Pemrograman Berbasis Kerangka Kerja

ASP.NET Core, Laravel, dan Spring Boot, dari Rute Pertama hingga Aplikasi



Irfan Subakti

Profitina

profitina.com

Pemrograman Berbasis Kerangka Kerja

ASP.NET Core, Laravel, dan Spring Boot, dari Rute Pertama hingga Aplikasi Siap Pakai

Irfan Subakti

Profitina

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 ..... **xxix**
Untuk siapa buku ini ditulis xxix
Satu aplikasi, tiga kerangka kerja xxix
Isi buku ini xxx
Bagaimana kode diperiksa xxxi
Cara membacanya xxxi
Tentang penulis xxxii
Ucapan terima kasih xxxii

BAB 1 Dari Pemrograman Web ke Kerangka Kerja **1**
1.1 Daftar Harga Bu Wati 1
1.2 Gagasannya dalam Kata-Kata Sederhana 2
1.2.1 Apa yang dikerjakan aplikasi web untuk setiap permintaan 2
1.2.2 Pustaka atau kerangka kerja: siapa memanggil siapa? 2
1.2.3 Model, View, Controller 4
1.2.4 Apa yang diberikan kerangka kerja, dan apa harganya 5
1.3 Mengapa ASP.NET Core, Laravel, dan Spring Boot 5
1.4 Menyiapkan Meja Kerja 7
1.4.1 Visual Studio Code 7
1.4.2 .NET 10 SDK 8
1.4.3 PHP 8.5 dan Composer 8
1.4.4 JDK 25 dan Maven 9
1.4.5 Basis data: SQLite dan MySQL 9
1.4.6 Memeriksa semuanya 9
1.5 Membangunnya Tanpa Kerangka Kerja 10
1.5.1 Dalam C#, dengan `HttpListener` 10
1.5.2 Dalam PHP, dengan server bawaan 13
1.5.3 Dalam Java, dengan `com.sun.net.httpserver` 15
1.6 Daftar Harga yang Sama dengan Kerangka Kerja 18
1.6.1 *Minimal API* ASP.NET Core 18
1.6.2 Menjalankannya 20
1.6.3 Proyek sungguhan pertama di setiap kerangka kerja 20
1.7 Menjalankan Keempat Program Berdampingan 21
1.8 Menjadikannya Lebih Baik: Yang Masih Kita Kerjakan Sendiri 23
1.9 Memakainya Sungguhan 23
1.9.1 Toko, klinik, atau kebun Anda sendiri 23
1.9.2 Memilih kerangka kerja untuk proyek sungguhan 23
1.9.3 Profitina dalam praktik 24
1.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 24
1.11 Ringkasan Bab 25
1.12 Soal Latihan 25
1.13 Penyelesaian Soal Latihan 26
Daftar Pustaka 27

BAB 2 Fondasi Bersama Kerangka Kerja Modern: MVC, Perutean, ORM, DI, Templat **29**
2.1 Posisi Kita Sekarang 29
2.2 Gagasannya dalam Kata-Kata Sederhana 30
2.3 Membangun Kerangka Kerja Mungil: MiniFrame 31
2.3.1 Perutean 31
2.3.2 Injeksi dependensi 33

2.3.3 Pemetaan objek-relasional 35
2.3.4 Templat 38
2.3.5 Aplikasi di atasnya 40
2.4 Menjalankannya 44
2.5 Fitur yang Sama di Tiga Kerangka Kerja Sungguhan 45
2.5.1 Perutean 45
2.5.2 Injeksi dependensi 46
2.5.3 Pemetaan objek-relasional 47
2.5.4 Templat 48
2.5.5 Satu fitur, tiga kerangka kerja 51
2.6 Menjadikannya Lebih Baik: Yang Ditinggalkan MiniFrame 52
2.7 Memakainya Sungguhan 53
2.7.1 Membaca kerangka kerja apa pun lewat empat fondasi 53
2.7.2 Profitina dalam praktik 53
2.7.3 Merencanakan fitur baru dengan empat pertanyaan 53
2.8 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 54
2.9 Ringkasan Bab 54
2.10 Soal Latihan 55
2.11 Penyelesaian Soal Latihan 55
Daftar Pustaka 56

BAB 3 ASP.NET Core MVC I: Proyek, Pipeline, Perutean, Controller, dan View Razor 58
3.1 Situs Web Bu Wati 58
3.2 Gagasannya dalam Kata-Kata Sederhana 59
3.2.1 *Host* dan *pipeline* 59
3.2.2 Satu permintaan melalui MVC 59
3.3 Membuat Proyeknya 60
3.3.1 Templat MVC 60
3.3.2 Yang kita ubah dari templat 61
3.4 `Program.cs`: Layanan, Lalu *Pipeline* 62
3.5 Model: Data dan Aturan 64
3.5.1 Layanan, gerai, dan uang 64
3.5.2 Aturan harga 65
3.5.3 *View model* dan model formulir 66
3.5.4 Katalog: akses data di balik antarmuka 67
3.6 Controller dan Perutean 69
3.6.1 Perutean konvensional: `HomeController` dan `OutletsController` 69
3.6.2 *Attribute routing*: `ServicesController` 71
3.6.3 Apa yang dikembalikan aksi 74
3.6.4 *Model binding*: formulir penawaran 74
3.7 View Razor 75
3.7.1 Pengaturan untuk semua view: `_ViewImports` dan `_ViewStart` 75
3.7.2 *Layout* 75
3.7.3 Halaman, *partial view*, dan *section* 76
3.7.4 *Tag helper* dan formulir penawaran 79
3.8 Menjalankannya 80
3.8.1 Di terminal atau di VS Code 80
3.8.2 Apa yang ditampilkan peramban 81
3.8.3 Setiap alamat, diperiksa 85
3.8.4 Log 86
3.9 Menjadikannya Lebih Baik 87
3.9.1 Versi controller gemuk, dan mengapa kita tidak menulisnya 87
3.9.2 Galat yang ramah yang tetap jujur statusnya 87
3.9.3 Yang masih kurang 87
3.10 Memakainya Sungguhan 88

3.10.1 Usaha Anda sendiri di atas LaundryMvc 88
3.10.2 Sekilas tentang *deployment* 88
3.10.3 Profitina dalam praktik 88
3.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 88
3.12 Ringkasan Bab 89
3.13 Soal Latihan 89
3.14 Penyelesaian Soal Latihan 90
Daftar Pustaka 91

BAB 4 Evaluasi dan Pembahasan I: Situs Etalase Profitina Laundry, Diselesaikan Tuntas 93

4.1 Posisi Kita Sekarang 93
4.2 Cara Menghadapi Soal Kerangka Kerja 94
4.3 Soal 4.1: Situs Etalase 95
4.3.1 Soalnya 95
4.3.2 Memahaminya secara manual 95
4.3.3 Merancangnya di atas kertas 96
4.3.4 Penyelesaiannya 97
4.3.5 Menjalankannya 107
4.3.6 Penyelesaian keliru yang menggoda 109
4.3.7 Uji diri 110
4.4 Soal 4.2: Di Mana Cucian Saya? 110
4.4.1 Soalnya 110
4.4.2 Memahaminya secara manual 110
4.4.3 Merancangnya di atas kertas 111
4.4.4 Penyelesaiannya 111
4.4.5 Menjalankannya 113
4.4.6 Penyelesaian keliru yang menggoda 114
4.4.7 Uji diri 115
4.5 Soal 4.3: Tabel Rute secara Manual 115
4.5.1 Soalnya 115
4.5.2 Memahaminya secara manual 116
4.5.3 Merancangnya di atas kertas 116
4.5.4 Penyelesaiannya: biarkan aplikasi menampilkan rutenya 116
4.5.5 Menjalankannya 117
4.5.6 Penyelesaian keliru yang menggoda 117
4.5.7 Uji diri 118
4.6 Soal 4.4: Header Waktu Respons 118
4.6.1 Soalnya 118
4.6.2 Memahaminya secara manual 118
4.6.3 Merancangnya di atas kertas 118
4.6.4 Penyelesaiannya 119
4.6.5 Menjalankannya 119
4.6.6 Penyelesaian keliru yang menggoda 119
4.6.7 Uji diri 120
4.7 Soal 4.5: Mode Pemeliharaan yang Tetap Bergaya 120
4.7.1 Soalnya 120
4.7.2 Memahaminya secara manual 121
4.7.3 Merancangnya di atas kertas 121
4.7.4 Penyelesaiannya 121
4.7.5 Menjalankannya 121
4.7.6 Penyelesaian keliru yang menggoda 122
4.7.7 Uji diri 123
4.8 Memakainya Sungguhan 123
4.8.1 Etalase untuk usaha Anda sendiri 123

4.8.2 Halaman pelacakan dan data pribadi 123
4.8.3 Profitina dalam praktik 123
4.9 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 124
4.10 Ringkasan Bab 124
4.11 Soal Latihan 125
4.12 Penyelesaian Soal Latihan 125
Daftar Pustaka 127

BAB 5 ASP.NET Core MVC II: Entity Framework Core dan CRUD Sungguhan 128
5.1 Posisi Kita Sekarang: Toko Tanpa Ingatan 128
5.2 Gagasannya dengan Kata-Kata Sederhana 129
5.2.1 Tabel di satu sisi, objek di sisi lain 129
5.2.2 Dua saat: saat desain dan saat berjalan 129
5.2.3 LINQ adalah bahasa yang sama di mana-mana 130
5.2.4 Mengapa SQLite 130
5.3 Merancang Data 130
5.4 LINQ Dulu, Tanpa Basis Data 132
5.5 Memasang Entity Framework Core 135
5.6 Kelas-Kelas Model 136
5.6.1 Service: dari *record* menjadi entitas 136
5.6.2 Customer, Order, dan Payment 136
5.7 DbContext 138
5.8 Mengisi Basis Data dan Menepati Janji Bab 3 141
5.8.1 Data awal 141
5.8.2 Katalog di atas basis data 143
5.8.3 Mendaftarkan semuanya di Program.cs 143
5.9 Migrasi: Dari Kelas ke Tabel 145
5.10 CRUD Sungguhan untuk Pelanggan 146
5.10.1 Mendapatkan DbContext 149
5.10.2 Read: *query*, AsNoTracking, dan Include 149
5.10.3 Create: mengikat, memeriksa, menyimpan 150
5.10.4 Update: ubah entitas yang dilacak 150
5.10.5 Delete: konfirmasi, lalu tolak atau hapus 150
5.10.6 *View*-nya 151
5.11 Pesanan, Pembayaran, dan Daftar Harga 153
5.11.1 Meja kasir 153
5.11.2 Daftar harga pemilik dan halaman kasir 158
5.12 Menjalankannya 160
5.13 Membuatnya Lebih Baik 161
5.14 Memakainya Sungguhan 161
5.15 Profitina dalam Praktik 162
5.16 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 162
5.17 Ringkasan Bab 163
5.18 Soal Latihan 163
5.19 Penyelesaian Soal Latihan 164
Daftar Pustaka 165

BAB 6 ASP.NET Core MVC III: Validasi, Identity, dan Penerapan 166
6.1 Posisi Kita Sekarang: Pintu yang Terbuka 166
6.2 Gagasannya dengan Kata-Kata Sederhana 166
6.3 Validasi dengan *Data Annotation* 168
6.3.1 Atribut pada model 168
6.3.2 Atribut validasi buatan sendiri 170
6.3.3 Menjalankan aturan tanpa server web 171
6.3.4 Apa yang dilakukan MVC dengan aturan-aturan itu 173

6.4 Kelas Formulir Alih-alih Entitas 174
6.4.1 Mengapa kelas tersendiri 174
6.4.2 Aturan tentang dua isian 175
6.4.3 *Controller* dengan kelas formulir 176
6.5 Pemeriksaan di Peramban Juga 180
6.6 Identity: Siapa di Balik Meja Kasir? 181
6.6.1 Apa yang disediakan Identity 181
6.6.2 Program.cs dengan Identity, otorisasi, dan pengaturan penerapan 183
6.6.3 Apa yang disimpan sebagai pengganti kata sandi 186
6.6.4 Peran dan akun pertama 186
6.7 Masuk dan Menjaga Pintu Tetap Tertutup 188
6.7.1 Halaman masuk 188
6.7.2 Siapa boleh melakukan apa 190
6.7.3 Kebijakan: aturan yang bernama 191
6.8 Penerapan: Dari Laptop ke Server 191
6.8.1 Lingkungan dan pengaturan 191
6.8.2 Menerbitkan 192
6.8.3 Migrasi di server 193
6.8.4 Server Linux dengan Nginx dan systemd 193
6.8.5 Windows dengan IIS 194
6.8.6 Hal-hal kecil yang membuatnya awet 194
6.8.7 Daftar periksa rilis 195
6.9 Menjalankannya 195
6.10 Membuatnya Lebih Baik 196
6.11 Memakainya Sungguhan 196
6.12 Profitina dalam Praktik 197
6.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 197
6.14 Ringkasan Bab 198
6.15 Soal Latihan 198
6.16 Penyelesaian Soal Latihan 198
Daftar Pustaka 200

BAB 7 Laravel I: Artisan, Routing, Blade, dan Eloquent 202
7.1 Posisi Kita Sekarang: Toko yang Sama, Bahasa yang Lain 202
7.2 Gagasannya dengan Kata-Kata Sederhana 202
7.3 Memasang PHP, Composer, dan Laravel 204
7.4 Artisan, Baris Perintah yang Menulis Kode 205
7.5 Migrasi: Tabel yang Ditulis dengan PHP 206
7.6 Model Eloquent 209
7.6.1 Enum untuk status dan cara bayar 209
7.6.2 Model-modelnya 211
7.6.3 Eloquent dan EF Core berdampingan 215
7.7 Mengisi Data dari Berkas Bab 3 216
7.8 *Routing* 218
7.9 *Controller* 220
7.9.1 Pelanggan: validasi dan CRUD 220
7.9.2 Pesanan, pembayaran, dan harga 223
7.9.3 Melihat SQL-nya 227
7.10 Templat Blade 228
7.10.1 *Layout* 228
7.10.2 *Escaping*, formulir, dan masukan lama 229
7.10.3 Halaman pesanan 230
7.10.4 Razor dan Blade berdampingan 232
7.11 Menjalankannya 233
7.12 Membuatnya Lebih Baik 234

7.13 Memakainya Sungguhan 235
7.14 Profitina dalam Praktik 235
7.15 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 235
7.16 Ringkasan Bab 236
7.17 Soal Latihan 236
7.18 Penyelesaian Soal Latihan 237
Daftar Pustaka 238

BAB 8 Evaluasi dan Pembahasan II: Rangkaian Fitur yang Sama di ASP.NET Core dan Laravel 240

8.1 Posisi Kita Sekarang 240
8.2 Cara Mendekati Soal Kerangka Kerja 240
8.2.1 Enam langkah 240
8.2.2 Tempat kode berada, dan apa yang dijalankan 241
8.3 Soal 8.1: Satu Aturan Harga, Dua Kerangka Kerja 242
8.3.1 Soalnya 242
8.3.2 Memahaminya dengan tangan 242
8.3.3 Merancangnya di atas kertas 242
8.3.4 Penyelesaian ASP.NET Core 243
8.3.5 Penyelesaian Laravel 245
8.3.6 Penyelesaian keliru yang menggoda 246
8.4 Soal 8.2: Alur Pesanan di Satu Tempat 247
8.4.1 Soalnya 247
8.4.2 Memahaminya dengan tangan 247
8.4.3 Merancangnya di atas kertas 247
8.4.4 Penyelesaian ASP.NET Core 248
8.4.5 Penyelesaian Laravel 248
8.4.6 Penyelesaian keliru yang menggoda 249
8.5 Soal 8.3: Daftar Pesanan Tanpa N+1 dan Tanpa Pemborosan 249
8.5.1 Soalnya 249
8.5.2 Memahaminya dengan tangan 249
8.5.3 Merancangnya di atas kertas 250
8.5.4 Penyelesaian ASP.NET Core 250
8.5.5 Penyelesaian Laravel 256
8.5.6 Penyelesaian keliru yang menggoda 259
8.6 Soal 8.4: Pencarian Pelanggan yang Menemukan Sari 259
8.6.1 Soalnya 259
8.6.2 Memahaminya dengan tangan 259
8.6.3 Merancangnya di atas kertas 260
8.6.4 Penyelesaian ASP.NET Core 260
8.6.5 Penyelesaian Laravel 263
8.6.6 Penyelesaian keliru yang menggoda 266
8.7 Soal 8.5: Tanpa Kelebihan Bayar 266
8.7.1 Soalnya 266
8.7.2 Memahaminya dengan tangan 266
8.7.3 Merancangnya di atas kertas 266
8.7.4 Penyelesaian ASP.NET Core 266
8.7.5 Penyelesaian Laravel 266
8.7.6 Penyelesaian keliru yang menggoda 267
8.8 Soal 8.6: Laporan Pendapatan Harian 267
8.8.1 Soalnya 267
8.8.2 Memahaminya dengan tangan 267
8.8.3 Merancangnya di atas kertas 268
8.8.4 Penyelesaian ASP.NET Core 268
8.8.5 Penyelesaian Laravel 270

8.8.6 Penyelesaian keliru yang menggoda 272
8.9 Soal 8.7: Rak untuk Setiap Kantong yang Selesai 273
8.9.1 Soalnya 273
8.9.2 Memahaminya dengan tangan 273
8.9.3 Merancangnya di atas kertas 273
8.9.4 Penyelesaian ASP.NET Core 273
8.9.5 Penyelesaian Laravel 275
8.9.6 Penyelesaian keliru yang menggoda 275
8.10 Soal-Soal Berdampingan 275
8.11 Profitina dalam Praktik 276
8.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 276
8.13 Ringkasan Bab 277
8.14 Soal Latihan 277
8.15 Penyelesaian Soal Latihan 278
Daftar Pustaka 279

BAB 9 Laravel II: Migrasi, Seeder, Middleware, dan Form Request 280
9.1 Gerai Ketiga Bu Wati 280
9.2 Gagasannya dengan Kata-Kata Sederhana 281
9.2.1 Migrasi: buku catatan perubahan basis data 281
9.2.2 Factory dan seeder: satu bulan yang masuk akal, sengaja direka 281
9.2.3 Middleware: pos pemeriksaan yang dilewati setiap permintaan 282
9.2.4 Form request: daftar periksa meja kasir, di satu tempat 282
9.3 Keadaan Proyek Setelah Bab 8 283
9.4 Migrasi Lebih Dalam 284
9.4.1 Apa yang diingat basis data 284
9.4.2 Mengubah tabel yang sudah berisi pesanan 285
9.4.3 Perintah-perintah yang Anda perlukan 286
9.4.4 Empat aturan untuk basis data yang dipakai bersama 287
9.5 Factory dan Seeder 287
9.5.1 Model yang dapat direka 287
9.5.2 Menulis factory 289
9.5.3 Menulis seeder 291
9.5.4 Menjalankannya 294
9.6 Middleware 294
9.6.1 Bawang, tanpa Laravel 294
9.6.2 Middleware “sebelum” untuk aturan toko 296
9.6.3 Middleware “sesudah” 297
9.6.4 Mendaftarkan middleware 297
9.6.5 Memasang middleware pada rute 298
9.7 Form Request 300
9.7.1 Daftar periksa pesanan 300
9.7.2 Aturan yang memerlukan pesanannya 301
9.7.3 Controller yang memercayai masukannya 302
9.8 Menjalankannya 306
9.9 Membuatnya Lebih Baik 306
9.10 Memakainya Sungguhan 307
9.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 307
9.12 Ringkasan Bab 308
9.13 Soal Latihan 309
9.14 Penyelesaian Soal Latihan 309
Daftar Pustaka 312

BAB 10 Laravel III: Autentikasi, API Resource, dan Pengujian 313
10.1 Daftar Harga yang Dapat Diubah Siapa Saja 313

10.2 Gagasannya dengan Kata-Kata Sederhana 314
10.2.1 Siapa Anda, dan apa yang boleh Anda lakukan? 314
10.2.2 Sesi untuk peramban, token untuk program 314
10.2.3 Resource: menentukan isi sebuah jawaban 315
10.2.4 Uji: sepasang tangan kedua yang tidak kenal lelah 316
10.3 Pengguna dan Peran 316
10.4 Masuk ke Halaman Staf 320
10.4.1 Controller untuk masuk 320
10.4.2 Menjaga rute 321
10.4.3 Gate untuk daftar harga 323
10.5 API untuk Program 325
10.5.1 Memasang Sanctum 325
10.5.2 Menerbitkan dan mencabut token 325
10.5.3 Rute API 326
10.5.4 Satu cara membuat pesanan 327
10.6 API Resource 333
10.7 Pengujian 335
10.7.1 Uji unit untuk aturan harga 335
10.7.2 Uji fitur untuk proses masuk 336
10.7.3 Uji fitur untuk API 338
10.7.4 Menjalankan uji 341
10.8 Menjalankannya 341
10.9 Membuatnya Lebih Baik 342
10.10 Memakainya Sungguhan 342
10.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 343
10.12 Ringkasan Bab 343
10.13 Soal Latihan 344
10.14 Penyelesaian Soal Latihan 344
Daftar Pustaka 346

BAB 11 Spring Boot I: Initializr, REST Controller, dan Spring Data JPA 348

11.1 Pertanyaan Raka 348
11.2 Gagasannya dengan Kata-Kata Sederhana 349
11.2.1 Container yang membangun objek Anda 349
11.2.2 Boot: bawaan yang wajar, dinyalakan oleh dependensi Anda 349
11.2.3 Entitas, repository, dan SQL yang tidak Anda tulis 349
11.2.4 Controller yang berbicara JSON 350
11.3 Membuat Proyek 351
11.3.1 Spring Initializr 351
11.3.2 Berkas build 351
11.3.3 Tata letak dan titik masuk 353
11.3.4 Pengaturan 353
11.4 Domain: Entitas 354
11.4.1 Empat tabel 354
11.4.2 Customer 354
11.4.3 Service, dan nama yang sudah dipakai 356
11.4.4 Order dan Payment 357
11.4.5 Status yang disimpan seperti di kerangka kerja lain 362
11.4.6 Aturannya, dalam Java biasa 363
11.5 Repository 365
11.6 REST Controller 366
11.6.1 Record untuk yang keluar 366
11.6.2 Record untuk yang masuk 368
11.6.3 Controller-nya 369
11.7 Memuat Daftar Harga 373

11.8 Menjalankannya 376
11.9 Menguji Aturannya 376
11.10 Membuatnya Lebih Baik 377
11.11 Memakainya Sungguhan 378
11.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 378
11.13 Ringkasan Bab 379
11.14 Soal Latihan 379
11.15 Penyelesaian Soal Latihan 380
Daftar Pustaka 382

BAB 12 Evaluasi dan Pembahasan III: Satu REST API di Laravel dan Spring Boot 383

12.1 Titik Berangkat Kita 383
12.2 Cara Mendekati Masalah API 383
12.3 Kontraknya 384
12.4 Masalah 12.1: Daftar Harga untuk Program 386
12.4.1 Masalahnya 386
12.4.2 Memahaminya dengan tangan 386
12.4.3 Membandingkan keduanya 386
12.4.4 Penyelesaian di kedua kerangka kerja 386
12.4.5 Penyelesaian keliru yang menggoda 388
12.5 Masalah 12.2: Satu Cara Menulis Waktu 389
12.5.1 Masalahnya 389
12.5.2 Memahaminya dengan tangan 389
12.5.3 Membandingkan keduanya 389
12.5.4 Penyelesaian di kedua kerangka kerja 389
12.5.5 Penyelesaian keliru yang menggoda 391
12.6 Masalah 12.3: Satu Bentuk untuk Setiap Galat 392
12.6.1 Masalahnya 392
12.6.2 Memahaminya dengan tangan 392
12.6.3 Membandingkan keduanya 392
12.6.4 Penyelesaian di Spring 392
12.6.5 Penyelesaian keliru yang menggoda 395
12.7 Masalah 12.4: Semua Galat Sekaligus 395
12.7.1 Masalahnya 395
12.7.2 Memahaminya dengan tangan 396
12.7.3 Membandingkan keduanya 396
12.7.4 Penyelesaian di Spring 396
12.7.5 Penyelesaian keliru yang menggoda 401
12.8 Masalah 12.5: Memajukan Pesanan, dan Pesanan yang Tertutup 401
12.8.1 Masalahnya 401
12.8.2 Memahaminya dengan tangan 401
12.8.3 Membandingkan keduanya 401
12.8.4 Penyelesaian di Spring 402
12.8.5 Penyelesaian keliru yang menggoda 402
12.9 Masalah 12.6: Pembayaran 402
12.9.1 Masalahnya 402
12.9.2 Memahaminya dengan tangan 402
12.9.3 Membandingkan keduanya 402
12.9.4 Penyelesaian di Spring 402
12.9.5 Penyelesaian keliru yang menggoda 403
12.10 Masalah 12.7: Laporan Harian 404
12.10.1 Masalahnya 404
12.10.2 Memahaminya dengan tangan 404
12.10.3 Membandingkan keduanya 404
12.10.4 Penyelesaian di kedua kerangka kerja 404

12.10.5 Penyelesaian keliru yang menggoda 408
12.11 Memeriksa Kontrak 408
12.11.1 Pemeriksanya 408
12.11.2 Satu jalan yang sungguhan 412
12.11.3 Uji di dalam setiap kerangka kerja 412
12.12 Memakainya Sungguhan 415
12.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 416
12.14 Ringkasan Bab 416
12.15 Soal Latihan 417
12.16 Penyelesaian Soal Latihan 417
Daftar Pustaka 418

BAB 13 Spring Boot II: Arsitektur Berlapis, Validasi, dan CRUD Lengkap dengan Thymeleaf 420
13.1 Titik Berangkat Kita 420
13.2 Gagasannya dalam Bahasa Sehari-hari 421
13.2.1 Satu kelas yang mengerjakan segalanya 421
13.2.2 Tiga lapisan, satu arah 421
13.2.3 Apa yang boleh diketahui setiap lapisan 422
13.2.4 Transaksi: semua atau tidak sama sekali 423
13.3 Di Mana Kode Ditempatkan 423
13.4 Lapisan Service 424
13.4.1 Tiga eksepsi 424
13.4.2 Pelanggan 425
13.4.3 Pesanan 430
13.4.4 Laporan harian 436
13.5 Controller API yang Tipis 437
13.6 Memvalidasi Masukan Formulir 444
13.6.1 Jakarta Bean Validation 444
13.6.2 *Record* formulir 445
13.6.3 Pesan aturan di field yang tepat 447
13.7 Halaman dengan Thymeleaf 447
13.7.1 Halaman pelanggan 447
13.7.2 Post/Redirect/Get 450
13.7.3 Templat 451
13.7.4 Pesanan dan pembayaran 454
13.8 Satu Tempat untuk Galat 462
13.9 SQLite, Konfigurasi, dan Menjalankannya 463
13.10 Menguji Lapisan Service 466
13.11 Membuatnya Lebih Baik 469
13.12 Memakainya Sungguhan 470
13.13 Profitina dalam Praktik 470
13.14 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 471
13.15 Ringkasan Bab 471
13.16 Soal Latihan 472
13.17 Penyelesaian Soal Latihan 472
Daftar Pustaka 474

BAB 14 Spring Boot III: Spring Security, Desain REST API, dan Pengujian Otomatis 475
14.1 Titik Berangkat Kita 475
14.2 Gagasannya dalam Bahasa Sehari-hari 475
14.3 Akun dan Kata Sandi 476
14.3.1 Jangan pernah menyimpan kata sandi 476
14.3.2 Staf dan peran 477
14.3.3 Memberi tahu Spring Security tentang staf kita 479

14.4 Token untuk API ... 480
14.4.1 Gagasan token Sanctum ... 480
14.4.2 Tabel token dan service token ... 481
14.4.3 Filternya ... 485
14.4.4 Masuk dan keluar ... 487
14.5 Dua Rantai Filter ... 488
14.5.1 Konfigurasinya ... 488
14.5.2 Mengikuti permintaan dengan tangan ... 491
14.5.3 Halaman login ... 492
14.5.4 CSRF: mengapa halaman memerlukannya dan API tidak ... 493
14.5.5 Garis pertahanan kedua di service ... 493
14.6 Mendesain REST API ... 496
14.6.1 *Resource*, URL, dan metode ... 497
14.6.2 Metode aman dan idempoten ... 497
14.6.3 Kode status dan badan galat ... 498
14.6.4 Amplop dan halaman ... 499
14.6.5 Yang masih kurang dari API: versi ... 499
14.7 Tes Otomatis pada Empat Tingkat ... 499
14.7.1 Piramidanya ... 499
14.7.2 Lapisan web saja ... 502
14.7.3 Kueri basis data ... 504
14.7.4 Seluruh aplikasi ... 505
14.8 Membuatnya Lebih Baik ... 507
14.9 Memakainya Sungguhan ... 507
14.10 Profitina dalam Praktik ... 508
14.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya ... 508
14.12 Ringkasan Bab ... 509
14.13 Soal Latihan ... 509
14.14 Penyelesaian Soal Latihan ... 510
Daftar Pustaka ... 511

BAB 15 Membandingkan Tiga Kerangka Kerja: Kinerja, Produktivitas, Ekosistem, Memilih ... 513

15.1 Titik Berangkat Kita ... 513
15.2 Gagasannya dalam Bahasa Sehari-hari ... 513
15.3 Satu Aplikasi, Tiga Kosakata ... 514
15.3.1 Konsep yang sama ... 514
15.3.2 Riwayat satu permintaan ... 516
15.3.3 Membandingkan pengujian ... 516
15.3.4 Endpoint yang sama tiga kali ... 517
15.4 Kinerja ... 519
15.4.1 Apa yang diukur benchmark publik ... 519
15.4.2 Mengukur sendiri ... 520
15.4.3 Memeriksa angka dengan hukum Little ... 521
15.4.4 Waktu nyala dan memori ... 522
15.5 Produktivitas ... 522
15.6 Ekosistem dan Komunitas ... 523
15.6.1 Apa yang dipakai pengembang ... 523
15.6.2 Paket, hosting, dan bantuan ... 524
15.7 Kalender Dukungan ... 525
15.8 Aman secara Bawaan ... 526
15.9 Memilih ... 527
15.10 Memakainya Sungguhan ... 528
15.11 Profitina dalam Praktik ... 528
15.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya ... 529

15.13 Ringkasan Bab 529
15.14 Soal Latihan 530
15.15 Penyelesaian Soal Latihan 530
Daftar Pustaka 531

BAB 16 Evaluasi dan Pembahasan IV 533
16.1 Titik Berangkat Kita 533
16.2 Cara Mendekati Capstone 534
16.3 Soal 16.1: Kontrak 535
16.3.1 Soalnya 535
16.3.2 Memahaminya dengan tangan 535
16.3.3 Kontraknya 535
16.3.4 Pemeriksanya 536
16.3.5 Membaca run yang gagal 541
16.3.6 Penyelesaian keliru yang menggoda 541
16.4 Soal 16.2: Aturan dalam Tiga Bahasa 541
16.4.1 Soalnya 541
16.4.2 Memahaminya dengan tangan 542
16.4.3 Penyelesaiannya 542
16.4.4 Penyelesaian keliru yang menggoda 543
16.5 Soal 16.3: Capstone ASP.NET Core 543
16.5.1 Soalnya 543
16.5.2 Merancangnya di atas kertas 543
16.5.3 Domain dan penyimpanan 544
16.5.4 Service-nya 546
16.5.5 API, galat, dan pintu-pintunya 551
16.5.6 Menjalankannya 562
16.5.7 Papan konter 564
16.5.8 Penyelesaian keliru yang menggoda 566
16.6 Soal 16.4: Capstone Laravel 566
16.6.1 Soalnya 566
16.6.2 Merancangnya di atas kertas 566
16.6.3 Penyelesaiannya 566
16.6.4 Menjalankannya 573
16.6.5 Penyelesaian keliru yang menggoda 573
16.7 Soal 16.5: Capstone Spring Boot 573
16.7.1 Soalnya 573
16.7.2 Penyelesaiannya 573
16.7.3 Menjalankannya 575
16.8 Membandingkan Ketiga Penyelesaian 576
16.9 Keputusan Bu Wati 577
16.10 Memakainya Sungguhan 577
16.11 Profitina dalam Praktik 578
16.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 578
16.13 Ringkasan Bab 579
16.14 Soal Latihan 579
16.15 Penyelesaian Soal Latihan 580
Daftar Pustaka 581

Daftar Gambar

Gambar 1.1 Pustaka dipanggil oleh program Anda; kerangka kerja memanggil program Anda. Dengan kerangka kerja, satu-satunya kode yang Anda tulis adalah bagian yang menyangkut bisnis Anda. 3

Gambar 1.2 Satu permintaan melalui Model–View–Controller: router memilih aksi controller, controller bertanya kepada model dan meneruskan hasilnya ke view, dan HTML dari view menjadi responsnya. 4

Gambar 1.3 Kerangka kerja dan teknologi web yang dipakai pengembang profesional dalam Stack Overflow Developer Survey 2025 (19.460 responden untuk pertanyaan ini); tiga kerangka kerja buku ini berwarna hijau toska. 6

Gambar 2.1 Empat fondasi di sepanjang satu permintaan: perutean memilih aksi, injeksi dependensi membangun controller, ORM mengubah objek menjadi SQL dan baris menjadi objek, dan templat mengubah objek menjadi HTML. 30

Gambar 2.2 Mencocokkan `GET /services/under/15000` dengan tabel rute: jumlah segmen menyingkirkan dua rute pertama, dan rute ketiga cocok karena parameter `{max:int}` menerima angka. 33

Gambar 2.3 Pendaftaran yang dibuat saat awal jalan, dan graf yang dibangun container ketika sebuah controller diminta. Hanya controller yang *transient* yang baru pada setiap permintaan. 35

Gambar 2.4 Di atas, sebuah baris tabel `service` menjadi objek `Service` karena nama kolom sama dengan nama properti. Di bawah, syarat `s => s.Price <= 15000` dibaca sebagai pohon ekspresi dan dicetak sebagai SQL. 38

Gambar 2.5 Dari model ke halaman: templat halaman diisi dengan model, hasilnya disisipkan ke dalam *layout*, dan seluruh halaman dikirim. 40

Gambar 3.1 *Pipeline* permintaan LaundryMvc: setiap permintaan melewati *middleware* pencatatan, kode status, dan perutean menuju *endpoint*, lalu respons kembali melalui lapisan yang sama. 59

Gambar 3.2 Satu permintaan, `GET /services/S0004`, melalui LaundryMvc: dari byte ke `HttpContext`, melalui *middleware* dan perutean, ke controller yang dibangun container, sebuah view, dan *layout*. 60

Gambar 3.3 Halaman beranda LaundryMvc: kepala halaman dan menu dari *layout*, teks view dengan hitungan dari *view model*, dan tiga kartu yang digambar *partial view* `_ServiceCard`. 81

Gambar 3.4 Daftar layanan di `/services`, dirender oleh `Views/Services/Index.cshtml`. 82

Gambar 3.5 Sebuah penawaran harga: *model binding* mengisi `QuoteForm` dari alamat, `PriceCalculator` menerapkan berat minimal dan biaya antar-jemput, dan view menampilkan hasilnya beserta penjelasannya. 83

Gambar 3.6 Formulir penawaran setelah `Quantity=abc`: galat konversi dari *model binding* muncul di ringkasan validasi, dan isiannya ditandai. 84

Gambar 3.7 Halaman 404 yang ramah untuk `/services/under/abc`, dihasilkan *middleware* kode status dan `HomeController.Error`. 84

Gambar 4.1 Enam langkah yang dipakai untuk setiap soal di bab ini; panah putus-putus adalah gelung antara menjalankan program dan memperbaikinya. 94

Gambar 4.2 Peta situs LaundryShowcase: setiap alamat adalah *attribute route* dari satu controller di bawah prefiks `/laundry`; rute pelacakan yang berbatasan milik Soal 4.2. 96

Gambar 4.3 Beranda etalase: slogannya berasal dari `site.json`, hitungannya dari data. 107

Gambar 4.4 Halaman perusahaan: setiap angka dihitung `LaundryData.Facts()` dari data pesanan. 108

Gambar 4.5 Halaman kerja sama: tawaran dan permintaan untuk investor, mitra, dan calon karyawan, dari `site.json`. 108

Gambar 4.6 Halaman pelacakan pesanan B0072: siap di rak A01, dengan layanan, gerai, tanggal, dan jumlahnya, tanpa apa pun tentang pelanggan. 113

Gambar 4.7 Halaman pemeliharaan, dijawab dengan status 503 dan diberi gaya oleh lembar gaya situs sendiri, yang dilolos kan mode pemeliharaan. 122

Gambar 5.1 Cara EF Core menghubungkan kelas dengan basis data: migrasi pada saat desain, LINQ yang diterjemahkan menjadi SQL dan pelacak perubahan pada saat berjalan. 130

Gambar 5.2 Basis data Profitina Laundry pada bab ini: setiap pesanan milik satu pelanggan dan satu layanan, dan sebuah pesanan dapat memiliki banyak pembayaran. 131
Gambar 6.1 Jalur pengiriman formulir melalui aplikasi Bab 6: autentikasi dan otorisasi sebelum *controller*, *model binding* dan validasi sebelum *action*, serta empat cara permintaan dapat berakhir. 168
Gambar 6.2 Aplikasi laundry di server Linux: Nginx di depan dengan sertifikatnya, Kestrel di belakangnya pada alamat *loopback*, systemd yang menjaganya tetap berjalan, serta folder data berisi basis data, kunci cookie, dan cadangan setiap malam. 193
Gambar 7.1 Satu permintaan melalui aplikasi Laravel: *front controller*, grup *middleware* web, *router*, *controller*, Eloquent, dan Blade. 203
Gambar 8.1 Tujuh soal di bab ini dan lapisan tempat masing-masing berada. 241
Gambar 8.2 Perintah SQL yang diperlukan untuk menampilkan satu halaman berisi 20 pesanan, menurut cara daftar dimuat; hitungannya mengikuti cara setiap pemanggilan didokumentasikan memuat data. 250
Gambar 9.1 Perjalanan satu permintaan, yaitu pembayaran pada pesanan 7, melalui *middleware*, *route model binding*, dan *form request*; kotak merah adalah tempat permintaan berhenti sebelum mencapai *controller*. 283
Gambar 9.2 Riwayat migrasi proyek: setiap kali `migrate` dijalankan terbentuk satu *batch*, tabel `migrations` mencatatnya, dan `migrate:rollback` hanya membatalkan *batch* terakhir. 285
Gambar 10.1 Dua cara membuktikan siapa yang bertanya: peramban masuk sekali lalu membawa *cookie* sesi; program menukar kata sandi dengan token lalu mengirimnya bersama setiap permintaan; keduanya berakhir di pemeriksaan otorisasi yang sama. 315
Gambar 10.2 Apa yang dilakukan `OrderResource` terhadap sebuah pesanan: isian diteruskan, diubah bentuknya, atau dihitung, relasi disarangkan bila dimuat, dan kolom internal ditinggalkan. 334
Gambar 11.1 Satu `POST /api/orders` melalui aplikasi Spring Boot: Tomcat tertanam, DispatcherServlet, Jackson, Bean Validation, *controller*, *repository* Spring Data, Hibernate, dan basis data. 350
Gambar 11.2 Empat tabel Profitina Laundry dan kelas entitas yang memetakannya; panah mengarah dari *foreign key* ke kunci primer yang dirujuknya. 354
Gambar 12.1 Satu kontrak, tiga implementasi, dan satu pemeriksa: klien hanya bergantung pada kontrak, dan `check_api.php` memeriksa setiap server terhadapnya. 384
Gambar 12.2 Alur kerja pesanan sebagaimana dinyatakan kontrak: satu langkah setiap kali, pergi hanya bila sisa tagihan 0, dan pesanan yang sudah pergi tertutup bagi setiap perubahan. 401
Gambar 13.1 Lapisan-lapisan aplikasi laundry Spring Boot: controller halaman dan API berbicara HTTP, service mengerjakan pekerjaan toko di dalam transaksi, repository berbicara SQL; aturannya sendiri adalah Java biasa dari Bab 11. 422
Gambar 13.2 Post/Redirect/Get: formulir yang tidak sah dijawab dengan halaman yang sama beserta pesannya; yang sah dijawab dengan *redirect*, sehingga menyegarkan halaman tidak pernah menerima pesanan yang sama dua kali. 451
Gambar 14.1 Dua rantai filter: permintaan di bawah `/api/` masuk ke rantai tanpa status dengan token *bearer* dan penolakan JSON; yang lain masuk ke rantai dengan formulir login, *cookie* sesi, dan perlindungan CSRF. 490
Gambar 14.2 Empat tingkat tes dalam proyek Bab 14, dari tes unit biasa di bawah sampai tes seluruh aplikasi di atas. 500
Gambar 15.1 TechEmpower Round 23 (Februari 2025), uji Fortunes: respons per detik dari implementasi ASP.NET Core, Spring, dan Laravel terbaik, dalam skala logaritmik. 519
Gambar 15.2 Pengukuran kami sendiri atas *endpoint* penawaran ASP.NET Core di mesin virtual 2 vCPU (25 September 2026): *throughput* mendatar setelah 8 pekerja sementara latensi p99 terus naik. 521
Gambar 15.3 Stack Overflow Developer Survey 2025: persentase responden yang memakai setiap kerangka kerja dalam setahun terakhir, dan persentase penggunanya yang ingin terus memakainya (“dikagumi”). 524
Gambar 15.4 Jendela dukungan versi-versi yang dipakai buku ini; garis putus-putus adalah 25 September 2026. 525

Gambar 16.1 Satu kontrak, tiga implementasi: pemeriksa `check_api.php` mengirim 25 permintaan yang sama ke setiap server dan membandingkan jawabannya dengan kontrak. 534
Gambar 16.2 Tabel-tabel *capstone*: layanan dari daftar harga (`S0001` sampai `S0012`), pelanggan, pesanan dengan antar-jemput, pembayaran, staf, dan token API. 543
Gambar 16.3 Papan konter *capstone* ASP.NET Core saat berjalan pada 25 September 2026: dua pesanan dalam antrean, satu siap diambil. 565

Daftar Tabel

Tabel 1.1 Pekerjaan yang diambil alih kerangka kerja web, dan bab-bab di buku ini tempat masing-masing dibangun untuk Profitina Laundry. 5
Tabel 1.2 Tiga kerangka kerja buku ini berdampingan, dengan versi yang berlaku pada September 2026. 7
Tabel 1.3 Ekstensi VS Code yang dipakai di buku ini. Jika komputer Anda lambat, pasang hanya ekstensi untuk kerangka kerja yang sedang Anda pelajari. 8
Tabel 1.4 Memasang ketiga rangkaian perkakas. Buka terminal baru setiap selesai memasang, agar perintah baru dikenali. 8
Tabel 2.1 Berkas-berkas MiniFrame di `Chapter02/miniframe`. Keempat fondasi bersama-sama berjumlah 226 baris. 31
Tabel 2.2 Keempat fondasi di MiniFrame dan di tiga kerangka kerja buku ini. 52
Tabel 3.1 Berkas-berkas yang dibuat `dotnet new mvc`, dan guna masing-masing. 61
Tabel 3.2 Cara rute konvensional `{controller=Home}/{action=Index}/{id?}` memetakan alamat ke aksi. Pencocokan tidak membedakan huruf besar dan kecil. 69
Tabel 3.3 *Attribute route* milik `ServicesController`. Templat yang diawali `/` mengabaikan prefiks controller. 73
Tabel 3.4 Kode status yang dikembalikan LaundryMvc yang sedang berjalan untuk setiap alamat bab ini, dari `verify.sh`. 85
Tabel 4.1 Kelima soal bab ini dan perkakas dari Bab 1 sampai 3 yang paling dilatih masing-masing. 94
Tabel 4.2 Tabel alamat Soal 4.1, termasuk tiga alamat yang menguji batas-batasnya. 96
Tabel 4.3 Tiga pesanan dari `Bill.csv`, satu untuk setiap status, dikerjakan secara manual. Jumlah untuk B0072 cocok: $7 \times 8000 = 56000$. 110
Tabel 4.4 Tabel alamat Soal 4.2. 111
Tabel 4.5 Tabel rute yang diramalkan secara manual untuk dua belas permintaan. Dua alasan berbeda bersembunyi di balik 404: tidak ada rute, atau aksi yang mengembalikan `NotFound()`. 116
Tabel 4.6 Kelima penyelesaian keliru yang menggoda di bab ini, apa yang dilakukan masing-masing, dan apa yang menangkapnya. 124
Tabel 5.1 Padanan antara model C# dan tabel SQLite. 132
Tabel 5.2 *Action* `CustomersController`, alamatnya, dan SQL yang ditimbulkannya. 146
Tabel 6.1 Atribut validasi yang dipakai di bab ini. 169
Tabel 6.2 Siapa boleh memakai halaman mana setelah bab ini. 190
Tabel 6.3 Daftar periksa rilis untuk server laundry. 195
Tabel 7.1 Tempat tinggal segala sesuatu dalam proyek Laravel 13, beserta padanannya di ASP.NET Core 205
Tabel 7.2 Perintah Artisan yang dipakai di bab ini. 205
Tabel 7.3 Operasi sehari-hari di EF Core dan di Eloquent. 216
Tabel 7.4 Tujuh rute dari `Route::resource('customers', …)`. 220
Tabel 7.5 Konstruksi Razor dan Blade untuk tugas yang sama. 233
Tabel 8.1 Tujuh soal, penyelesaiannya di kedua kerangka kerja, dan pelajaran dari masing-masing. 276
Tabel 9.1 Berkas yang ditambahkan atau diubah bab ini pada proyek Bab 8. 284
Tabel 9.2 Perintah migrasi Artisan dan kapan masing-masing dipakai. 287

Tabel 9.3 *Middleware* rute bawaan Laravel yang dipakai buku ini, beserta nama pendeknya. 298
Tabel 10.1 Penolakan dalam aplikasi bab ini: kode status, artinya, dan bagian yang menjawab. 324
Tabel 11.1 Pekerjaan yang sama di Laravel dan di Spring Boot. 351
Tabel 12.1 *Endpoint* kontrak API Profitina Laundry; tanda tanya menandai isian yang opsional. 385
Tabel 12.2 Penolakan dalam kontrak, dan badan yang dibawa masing-masing. 386
Tabel 13.1 Apa yang boleh diketahui setiap lapisan, dan apa yang tidak boleh. 423
Tabel 13.2 Paket-paket proyek Bab 13. 423
Tabel 13.3 Batasan Bean Validation yang dipakai formulir. 445
Tabel 13.4 URL halaman pelanggan. 450
Tabel 14.1 Enam permintaan diikuti melewati kedua rantai. 491
Tabel 14.2 API Profitina Laundry setelah Bab 14. 497
Tabel 14.3 Kode status API dan kapan masing-masing dipakai. 498
Tabel 14.4 Empat tingkat tes, dengan 24 run di bab ini. 500
Tabel 15.1 Apa yang dibandingkan, dan jenis bukti yang diperlukan setiap pertanyaan. 514
Tabel 15.2 Konsep yang sama dengan tiga nama. 515
Tabel 15.3 Baris yang diperlukan untuk dua fitur yang sama pada *capstone* Bab 16, dihitung pada 25 September 2026. 522
Tabel 15.4 Tanggal rilis dan dukungan, dihimpun dari endoflife.date pada 25 September 2026. 525
Tabel 15.5 Bawaan sekuriti dibandingkan. 526
Tabel 15.6 Empat keadaan dan kerangka kerja yang disarankan prosedur ini. 527
Tabel 16.1 Kontrak API Profitina Laundry setelah Bab 16. 536
Tabel 16.2 Sembilan harga yang dikerjakan dengan tangan. 542
Tabel 16.3 Ketiga *capstone* dibandingkan. 576

Daftar Kode

Kode 1.1 `nofw-csharp/Program.cs`: daftar harga tanpa kerangka kerja. Komentar (1) sampai (6) menandai pekerjaan yang biasanya dilakukan kerangka kerja. 11
Kode 1.2 `nofw-php/index.php`: daftar harga yang sama dalam PHP tanpa kerangka kerja. Skrip ini sekaligus router, controller, model, dan view. 14
Kode 1.3 `nofw-java/RawLaundry.java`: daftar harga dalam Java tanpa kerangka kerja, di atas server HTTP bawaan JDK. 15
Kode 1.4 `aspnet/HelloLaundry.csproj`. SDK `.Web` menjadikan ini proyek ASP.NET Core; `net10.0` menargetkan .NET 10. 18
Kode 1.5 `aspnet/Program.cs`: daftar harga dengan *minimal API* ASP.NET Core. Perutean, ekstraksi parameter, kode status, JSON, dan server adalah urusan kerangka kerja. 18
Kode 2.1 `Router.cs`: tabel rute, pencocokan templat, dan ekstraksi parameter. 31
Kode 2.2 `Container.cs`: pendaftaran, dua masa hidup, dan injeksi lewat konstruktor dengan refleksi. 34
Kode 2.3 `Mapper.cs`: baris menjadi objek dengan mencocokkan nama, dan syarat C# menjadi teks klausa SQL `WHERE`. 35
Kode 2.4 `Template.cs`: lubang, gelung, dan *layout*, dengan *escaping* sebagai bawaan. 38
Kode 2.5 `templates/layout.html`: bingkai yang dipakai bersama semua halaman; HTML milik halaman masuk di tempat `{{{ body }}}`. 39
Kode 2.6 `templates/services.html`: gelung atas baris-baris model, dengan setiap nilai di-*escape*. 40
Kode 2.7 `Laundry.cs`: model, *repository*, dan controller daftar harga. Tidak ada perutean, penguraian, SQL, atau *escaping* HTML di sini. 40
Kode 2.8 `Program.cs`: pendaftaran, rute, demo luring, dan server HTTP minimal. 42
Kode 2.9 `routes/web.php` di Laravel: tiga rute, masing-masing menunjuk metode controller; `whereNumber` adalah batasannya. 45

Kode 2.10 `app/Models/Service.php`: model Eloquent. Properti-propertinya hanya menyatakan apa yang berbeda dari konvensi Laravel. 47
Kode 2.11 `Service.java`: *entity* JPA yang dipetakan ke tabel `service`. 47
Kode 2.12 `ServiceRepository.java`: *repository* Spring Data. Kuerinya diturunkan dari nama metodenya. 48
Kode 2.13 `resources/views/layouts/app.blade.php`: *layout* Blade; `@yield` menandai slot-slotnya. 48
Kode 2.14 `resources/views/services/index.blade.php`: halaman daftar, yang mengisi dua slot milik *layout*. 48
Kode 2.15 `app/Http/Controllers/ServiceController.php`: controller Laravel. `findOrFail` menjawab 404 dengan sendirinya jika kodenya tidak ada. 49
Kode 2.16 `templates/layout.html`: dua *fragment* Thymeleaf, bagian *head* (dengan parameter) dan bagian kepala halaman. 50
Kode 2.17 `templates/services/index.html`: halaman daftar dalam Thymeleaf; `th:each` adalah gelungnya, `th:text` lubang yang di-*escape*. 50
Kode 2.18 `ServiceController.java`: controller Spring MVC, dengan *repository*-nya disuntikkan lewat konstruktor. 50
Kode 3.1 `Properties/launchSettings.json`: `dotnet run` menjalankan aplikasi di `http://localhost:4030` dalam lingkungan Development. 62
Kode 3.2 `appsettings.json`: tingkat log dan *host* yang diizinkan. 62
Kode 3.3 `Program.cs`: layanan didaftarkan pada *builder*, lalu *pipeline* dibangun pada aplikasi. 62
Kode 3.4 `Models/Service.cs`: sebuah layanan dengan kode, deskripsi, dan harganya, serta tiga properti yang diturunkan dari deskripsinya. 64
Kode 3.5 `Models/Outlet.cs`: sebuah gerai Profitina Laundry. 65
Kode 3.6 `Models/Money.cs`: jumlah rupiah dengan gaya Indonesia, diformat di satu tempat. 65
Kode 3.7 `Models/PriceCalculator.cs`: aturan harga toko, dengan penjelasan untuk setiap aturan yang diterapkan. 65
Kode 3.8 `Models/HomeViewModel.cs`: persis apa yang ditampilkan halaman beranda. 67
Kode 3.9 `Models/QuoteForm.cs`: masukan formulir penawaran harga, diisi oleh *model binding*. 67
Kode 3.10 `Data/ILaundryCatalog.cs`: apa yang boleh ditanyakan controller tentang layanan dan gerai. 67
Kode 3.11 `Data/CsvCatalog.cs`: katalog yang diimplementasikan di atas kedua berkas CSV, dibaca sekali saat awal jalan. 68
Kode 3.12 `Controllers/HomeController.cs`: halaman beranda, halaman tentang kami, dan halaman yang ditampilkan untuk kode status galat. 69
Kode 3.13 `Controllers/OutletsController.cs`: gerai-gerai, dicapai melalui rute konvensional. 70
Kode 3.14 `Controllers/ServicesController.cs`: *attribute route*, sebuah batasan, aksi formulir, dan aksi JSON. 71
Kode 3.15 `Views/_ViewImports.cshtml`: *namespace* dan *tag helper* untuk setiap view. 75
Kode 3.16 `Views/_ViewStart.cshtml`: setiap view memakai *layout* `_Layout`. 75
Kode 3.17 `Views/Shared/_Layout.cshtml`: bingkai bersama, dengan menu yang menandai halaman yang sedang dibuka. 75
Kode 3.18 `Views/Home/Index.cshtml`: halaman beranda, dengan *partial view* untuk setiap kartu. 76
Kode 3.19 `Views/Shared/_ServiceCard.cshtml`: satu kartu, ditulis sekali dan dipakai untuk setiap layanan populer. 77
Kode 3.20 `Views/Home/About.cshtml`: halaman yang menambahkan jam buka ke kaki halaman *layout* melalui sebuah *section*. 77
Kode 3.21 `Views/Services/Index.cshtml`: tabel layanan, dipakai bersama oleh aksi `Index` dan `Under`. 77
Kode 3.22 `Views/Services/Details.cshtml`: satu layanan, dengan tautan ke penawaran untuknya. 78
Kode 3.23 `Views/Outlets/Index.cshtml`: daftar gerai. 78
Kode 3.24 `Views/Outlets/Details.cshtml`: satu gerai. 78
Kode 3.25 `Views/Shared/Error.cshtml`: halaman ramah untuk 404 dan status galat lainnya. 79

Kode 3.26 `Views/Services/Quote.cshtml`: formulir penawaran, dibangun dengan *tag helper*, beserta tabel hasilnya. 80
Kode 4.1 `content/site.json`: setiap kata di etalase yang tidak dihitung dari data. 97
Kode 4.2 `Models/SiteContent.cs`: bentuk `site.json` sebagai *record* C#. 98
Kode 4.3 `Models/Records.cs`: layanan, gerai, pesanan tanpa pelanggan, dan fakta yang dihitung darinya. 99
Kode 4.4 `Data/LaundryData.cs`: data toko, dimuat sekali, dan fakta yang ditampilkan halaman. 99
Kode 4.5 `Program.cs` milik LaundryShowcase: konten dan data didaftarkan sebagai *singleton*, *pipeline middleware*, sebuah pengalihan, controller, dan daftar rute khusus pengembangan. 101
Kode 4.6 `Controllers/ShowcaseController.cs`: satu controller ber-*attribute route* untuk seluruh etalase. 103
Kode 4.7 `Views/_ViewImports.cshtml`: `@inject` menaruh konten situs ke setiap view. 104
Kode 4.8 `Views/Shared/_Layout.cshtml`: bingkai dan menu etalase. 104
Kode 4.9 `Views/Showcase/Home.cshtml`: slogan dan hitungan yang dihitung dari data. 105
Kode 4.10 `Views/Showcase/Profile.cshtml`: kisah dan tiga janji, dari `site.json`. 105
Kode 4.11 `Views/Showcase/Company.cshtml`: misi, angka dari data, dan tim. 106
Kode 4.12 `Views/Showcase/Services.cshtml`: daftar harga. 106
Kode 4.13 `Views/Showcase/Collaboration.cshtml`: apa yang ditawarkan dan diminta usaha ini dari setiap khalayak. 106
Kode 4.14 `Tempting/AbsoluteRoute/Program.cs`: satu garis miring di depan memindahkan profil keluar dari `/laundry`. 109
Kode 4.15 `Views/Showcase/TrackForm.cshtml`: formulir `GET` dengan satu isian. 112
Kode 4.16 `Views/Showcase/Track.cshtml`: status dalam kata-kata, lalu fakta pesanannya, tanpa data pribadi. 112
Kode 4.17 `Views/Shared/Error.cshtml`: satu kalimat untuk halaman atau pesanan yang tidak ada, beserta alamat yang diminta. 112
Kode 4.18 `Tempting/RegexBraces/Program.cs`: ekspresi reguler dengan kurung kurawal tunggal di dalam templat rute. 114
Kode 4.19 `Tempting/Ambiguous/Program.cs`: dua *attribute route* yang cocok dengan alamat yang persis sama. 117
Kode 4.20 `Middleware/ResponseTimeMiddleware.cs`: header diisi dalam *callback* `OnStarting`, tepat sebelum header dikirim. 119
Kode 4.21 `Tempting/LateHeader/Program.cs`: header diisi setelah sisa *pipeline* sudah memulai responsnya. 119
Kode 4.22 `appsettings.json` milik LaundryShowcase: sakelar pemeliharaan, mati secara bawaan. 121
Kode 4.23 `Tempting/MaintenanceFirst/Program.cs`: pemeriksaan pemeliharaan sebelum `UseRouting`, ketika belum ada *endpoint* yang dipilih. 122
Kode 5.1 `LinqDemo/Program.cs`: *query* bab ini pada *list* di memori, dengan kelas-kelas model milik aplikasi web. 132
Kode 5.2 `LaundryMvc.csproj` setelah dua paket EF Core ditambahkan. 135
Kode 5.3 `appsettings.json`: untai koneksi dan log SQL. 135
Kode 5.4 `Models/Service.cs`: *record* Bab 3 yang diubah menjadi kelas entitas dengan *setter* dan penanda `IsActive`. 136
Kode 5.5 `Models/Customer.cs`: seorang pelanggan dan properti navigasi ke pesanan-pesanannya. 137
Kode 5.6 `Models/Order.cs`: pesanan dengan kunci asing, status, cap waktu, dan pembayarannya; `Paid` dan `Balance` menerapkan rumus sisa tagihan. 137
Kode 5.7 `Models/Payment.cs`: satu pembayaran untuk satu pesanan. 137
Kode 5.8 `Data/LaundryDbContext.cs`: lima tabel dan aturan-aturan yang tidak dapat ditebak konvensi. 138
Kode 5.9 `Data/SeedData.cs`: layanan dan gerai dari berkas CSV Bab 3, lalu pelanggan, pesanan, dan pembayaran contoh. 141
Kode 5.10 `Data/EfCatalog.cs`: *interface* katalog Bab 3 yang dijawab oleh EF Core. 143

Kode 5.11 `Program.cs`: *pipeline* Bab 3 dengan DbContext yang didaftarkan dan katalog yang ditukar. 143
Kode 5.12 `Controllers/CustomersController.cs`: CRUD lengkap untuk pelanggan dengan EF Core 146
Kode 5.13 `Views/Customers/Index.cshtml`: daftar pelanggan dengan kotak pencarian. 151
Kode 5.14 `Views/Customers/_Form.cshtml`: isian yang dipakai bersama oleh halaman Create dan Edit 151
Kode 5.15 `Views/Customers/Details.cshtml`: satu pelanggan dan pesanan-pesanannya, yang terbaru di atas 151
Kode 5.16 `Views/Customers/Create.cshtml`: halaman pelanggan baru di sekeliling formulir bersama. 152
Kode 5.17 `Controllers/OrdersController.cs`: daftar pesanan, halaman pesanan, pesanan baru yang dihargai oleh kalkulator Bab 3, alur kerja, dan pembayaran. 153
Kode 5.18 `Views/Orders/Details.cshtml`: satu pesanan, tombol alur kerjanya, pembayarannya, dan formulir pembayaran. 156
Kode 5.19 `Views/Orders/Create.cshtml`: formulir kasir untuk pesanan baru. 158
Kode 5.20 `Controllers/PricesController.cs`: daftar harga pemilik, dengan kode layanan otomatis. 158
Kode 5.21 `Controllers/CounterController.cs`: empat angka untuk meja kasir, masing-masing satu perintah SQL. 160
Kode 5.22 Soal 5.3: tiga pelanggan terbaik, sebagai baris-baris yang ditambahkan ke `LinqDemo/Program.cs`. 164
Kode 5.23 Soal 5.5: *action* yang ditambahkan ke `OrdersController` untuk menghapus satu pembayaran dari satu pesanan. 165
Kode 6.1 `Models/Customer.cs` dengan atribut validasi: nama wajib sepanjang 2 sampai 80 karakter dan nomor seluler Indonesia yang wajib. 168
Kode 6.2 `Models/Service.cs` dengan aturan untuk deskripsi dan harga. 169
Kode 6.3 `Models/IndonesianMobileAttribute.cs`: atribut validasi untuk nomor seluler Indonesia, dengan pembantu yang menyimpan setiap nomor dalam satu ejaan. 170
Kode 6.4 `ValidationDemo/Program.cs`: aturan validasi milik aplikasi dan *password hasher* Identity, dijalankan dalam konsol. 171
Kode 6.5 `Models/OrderForm.cs`: persis apa yang boleh dikirim formulir pesanan baru, beserta aturannya, termasuk satu aturan yang melibatkan dua isian. 174
Kode 6.6 `Models/PaymentForm.cs`: formulir pembayaran dan aturannya. 176
Kode 6.7 `Controllers/OrdersController.cs` dengan `OrderForm` dan `PaymentForm` 176
Kode 6.8 `Views/Orders/Create.cshtml` yang terikat pada `OrderForm` 180
Kode 6.9 `Data/LaundryDbContext.cs` yang diturunkan dari `IdentityDbContext`, sehingga pengguna dan peran tinggal di basis data yang sama dengan pesanan. 181
Kode 6.10 `Program.cs`: Identity, kebijakan otorisasi cadangan, dan pengaturan yang diperlukan untuk penerapan. 183
Kode 6.11 `Models/AppRoles.cs`: dua peran di toko. 186
Kode 6.12 `Data/IdentitySeeder.cs`: migrasi, peran, dan akun pertama saat aplikasi dimulai. 187
Kode 6.13 `appsettings.Development.json`: akun demo hanya untuk komputer pengembang. 188
Kode 6.14 `Controllers/AccountController.cs`: masuk, keluar, dan halaman penolakan. 188
Kode 6.15 `Views/Account/Login.cshtml`: formulir masuk staf. 189
Kode 6.16 `Views/Account/AccessDenied.cshtml`: yang dilihat pengguna yang sudah masuk ketika sebuah halaman bukan untuk perannya. 190
Kode 6.17 `appsettings.Production.json`: basis data dan kunci cookie di folder tetap, serta log yang lebih tenang. 191
Kode 6.18 `deploy/laundrymvc.service`: unit systemd yang menjaga aplikasi tetap berjalan. 193
Kode 6.19 `deploy/nginx-laundrymvc.conf`: Nginx yang meneruskan ke Kestrel beserta *forwarded headers* 193
Kode 6.20 Soal 6.2: `Models/NotInFutureAttribute.cs`, atribut validasi buatan sendiri untuk tanggal. 199

Kode 6.21 Soal 6.3: kelas formulir dengan satu isian dan *action* yang memakainya, untuk ditambahkan ke `OrdersController`. 199
Kode 7.1 `2026_09_01_000001_create_customers_table.php`: tabel pelanggan. 206
Kode 7.2 `2026_09_01_000002_create_services_table.php`: layanan dan gerai, dengan kodenya sebagai kunci primer. 206
Kode 7.3 `2026_09_01_000003_create_orders_table.php`: pesanan dengan kunci asing, status, dan cap waktunya. 207
Kode 7.4 `2026_09_01_000004_create_payments_table.php`: pembayaran, yang terhapus bersama pesanannya. 208
Kode 7.5 `app/Enums/OrderStatus.php`: tiga status pesanan dan alur kerja di antaranya. 209
Kode 7.6 `app/Enums/PaymentMethod.php`: tunai, QRIS, dan transfer. 210
Kode 7.7 `app/Support/Money.php`: rupiah yang diformat ala Indonesia. 210
Kode 7.8 `tools/enum_demo.php`: enum dan pembantu uang, dijalankan dengan PHP biasa. 211
Kode 7.9 `app/Models/Customer.php`: seorang pelanggan dan relasi ke pesanan-pesanannya. 211
Kode 7.10 `app/Models/Service.php`: kunci berupa untai, *cast*, dan tiga atribut terhitung. 212
Kode 7.11 `app/Models/Order.php`: *cast*, relasi, dan sisa tagihan sebuah pesanan. 213
Kode 7.12 `app/Models/Payment.php`: sebuah pembayaran dan pesanannya. 214
Kode 7.13 `app/Models/Outlet.php`: tiga gerai dari Bab 3. 215
Kode 7.14 `database/seeders/DatabaseSeeder.php`: layanan dan gerai dari berkas CSV Bab 3, lalu pelanggan, pesanan, dan pembayaran contoh. 216
Kode 7.15 `routes/web.php`: halaman publik, halaman staf, dan dua *action* tambahan pada pesanan. 218
Kode 7.16 `app/Http/Controllers/CustomerController.php`: CRUD lengkap untuk pelanggan dengan Eloquent. 220
Kode 7.17 `app/Http/Controllers/OrderController.php`: daftar pesanan, halaman pesanan, pesanan baru yang dihargai dengan aturan Bab 3, dan alur kerjanya. 223
Kode 7.18 `app/Http/Controllers/PaymentController.php`: mencatat pembayaran untuk sebuah pesanan. 225
Kode 7.19 `app/Http/Controllers/PriceController.php`: daftar harga milik pemilik. 225
Kode 7.20 `app/Http/Controllers/CounterController.php`: empat angka untuk meja kasir. 227
Kode 7.21 `resources/views/layouts/app.blade.php`: *layout* yang dipakai bersama oleh setiap halaman. 228
Kode 7.22 `resources/views/customers/_form.blade.php`: isian yang dipakai bersama oleh halaman buat dan ubah. 229
Kode 7.23 `resources/views/customers/edit.blade.php`: formulir ubah dengan token CSRF dan isian metodenya. 229
Kode 7.24 `resources/views/customers/show.blade.php`: satu pelanggan, pesanan-pesanannya, dan tombol hapus. 230
Kode 7.25 `resources/views/orders/show.blade.php`: satu pesanan dengan tombol alur kerja, pembayaran, dan formulir pembayarannya. 230
Kode 7.26 Soal 7.3: rute dan metode *controller* untuk penawaran harga (tambahkan `use Illuminate\Http\Request;` ke *controller*). 237
Kode 7.27 Soal 7.5: metode `up` dan `down` dari migrasi yang menambahkan kolom rak. 238
Kode 8.1 `Models/PriceCalculator.cs`: aturan harga Bab 3 dalam C#, dengan aritmetika `decimal`. 243
Kode 8.2 `CoreCheck/Program.cs`: aturan harga dan alur kerja pada kasus-kasus tetap, dalam C#. 243
Kode 8.3 `app/Domain/PriceCalculator.php`: aturan yang sama dalam PHP, dengan aritmetika bilangan bulat yang tepat. 245
Kode 8.4 `tools/core_check.php`: kasus-kasus yang sama dengan `CoreCheck`, dalam PHP biasa. 245
Kode 8.5 `tools/wrong_price.php`: harga yang dihitung dengan *float*, dengan tiga cara. 246
Kode 8.6 `Domain/OrderWorkflow.cs`: alur kerja dan aturan barunya, sebagai kelas biasa. 248
Kode 8.7 `app/Domain/OrderWorkflow.php`: alur kerja dan aturan barunya dalam PHP. 248
Kode 8.8 `Models/OrderRow.cs`: persis kolom-kolom daftar pesanan. 250

Kode 8.9 `Domain/PagedList.cs`: satu halaman dari *query* terurut dan apa yang dibutuhkan penomor halaman. 250
Kode 8.10 `Controllers/OrdersController.cs`: daftar pesanan sebagai proyeksi berhalaman, alur kerja Soal 8.2, dan pemeriksaan pembayaran Soal 8.5. 251
Kode 8.11 `Views/Orders/Index.cshtml`: daftar pesanan dari *record* `OrderRow`, dengan penomor halaman. 255
Kode 8.12 `app/Http/Controllers/OrderController.php`: daftar pesanan dengan halaman dan jumlah, harga dari Soal 8.1, dan alur kerja Soal 8.2. 256
Kode 8.13 `Controllers/CustomersController.cs`: pencarian yang tidak membedakan huruf besar-kecil, dengan halaman. 260
Kode 8.14 `Views/Shared/_Pager.cshtml`: satu penomor halaman untuk setiap daftar, yang mempertahankan *query string*. 262
Kode 8.15 `app/Http/Controllers/CustomerController.php`: pencarian yang sama dengan halaman di Laravel. 263
Kode 8.16 `resources/views/_pager.blade.php`: penomor halaman dalam Blade. 265
Kode 8.17 `app/Http/Controllers/PaymentController.php`: pembayaran yang dibatasi sisa tagihan. 267
Kode 8.18 `Models/DailyRevenueRow.cs`: satu baris laporan. 268
Kode 8.19 `Controllers/ReportsController.cs`: uang yang diterima per hari dan cara bayar, dikelompokkan di basis data. 268
Kode 8.20 `Views/Reports/Daily.cshtml`: laporan pendapatan harian. 269
Kode 8.21 `app/Http/Controllers/ReportController.php`: laporan yang sama dengan *query builder* dan koleksi. 270
Kode 8.22 `resources/views/reports/daily.blade.php`: laporan dalam Blade. 271
Kode 8.23 `Models/OrderForm.cs` dengan rak dan aturan dua desimal. 273
Kode 8.24 `2026_09_20_000001_add_rack_to_orders_table.php`: kolom rak, ditambahkan dan dihapus. 275
Kode 9.1 `2026_09_27_000001_add_outlet_id_to_orders_table.php`: kolom baru dengan *foreign key*, dan pesanan lama diisi. 285
Kode 9.2 `app/Models/Order.php`: model Bab 8 dengan `HasFactory` dan relasi baru ke gerai. 287
Kode 9.3 `database/factories/CustomerFactory.php`: satu pelanggan rekaan. 289
Kode 9.4 `database/factories/OrderFactory.php`: satu pesanan rekaan, diberi harga dengan aturan yang sungguhan, dengan *state* untuk pesanan yang sudah selesai dan sudah pergi. 289
Kode 9.5 `database/seeders/PriceListSeeder.php`: daftar harga dan gerai yang sungguhan, aman dijalankan ulang. 291
Kode 9.6 `database/seeders/BusyMonthSeeder.php`: empat puluh pelanggan rekaan dan satu bulan pesanan yang bergerak melalui alur kerja seperti pesanan sungguhan. 292
Kode 9.7 `database/seeders/DatabaseSeeder.php`: data sungguhan selalu, data rekaan hanya di komputer pengembang atau komputer uji. 293
Kode 9.8 `tools/pipeline_demo.php`: *middleware pipeline* dalam PHP biasa, gagasan yang sama dengan dasar *pipeline* Laravel. 294
Kode 9.9 `app/Http/Middleware/EnsureOrderIsOpen.php`: pesanan yang sudah meninggalkan toko tidak boleh diubah. 296
Kode 9.10 `app/Http/Middleware/AddServerTiming.php`: *middleware* "sesudah" yang mengukur sisa *pipeline*. 297
Kode 9.11 `bootstrap/app.php`: `AddServerTiming` bergabung dengan grup `web`, dan `EnsureOrderIsOpen` mendapat nama pendek `order.open`. 297
Kode 9.12 `routes/web.php`: rute-rute Bab 8, dengan rute yang mengubah pesanan dikelompokkan di belakang `order.open`. 298
Kode 9.13 `app/Http/Requests/StoreOrderRequest.php`: daftar periksa untuk pesanan baru, dengan perapian, aturan, pesan, dan nama isian. 300
Kode 9.14 `app/Http/Requests/StorePaymentRequest.php`: aturan sederhana lebih dahulu, lalu pemeriksaan terhadap sisa tagihan. 301

Kode 9.15 `app/Http/Controllers/PaymentController.php`: ketika `store` berjalan, permintaannya sudah diperiksa. 302
Kode 9.16 `app/Http/Controllers/OrderController.php`: *controller* Bab 8 dengan `StoreOrderRequest` di `store` dan daftar gerai di `create`. 303
Kode 9.17 `2026_10_01_000001_add_promised_at_to_orders_table.php`: kolom baru yang diisi baris demi baris, per potongan. 310
Kode 9.18 `app/Http/Middleware/LogSlowRequests.php`: *middleware* rute dengan parameter. 310
Kode 10.1 `app/Enums/Role.php`: dua peran di toko. 316
Kode 10.2 `2026_10_04_000001_add_role_to_users_table.php`: setiap pengguna mendapat peran, kasir kecuali dinyatakan lain. 316
Kode 10.3 `app/Models/User.php`: pengguna dengan peran, kata sandi yang di-*hash*, dan token API. 317
Kode 10.4 `database/factories/UserFactory.php`: kasir secara bawaan, dan *state* `owner()`. 318
Kode 10.5 `database/seeders/DatabaseSeeder.php`: dua akun demo, hanya di komputer pengembang. 319
Kode 10.6 `app/Http/Controllers/Auth/LoginController.php`: menampilkan formulir, masuk, dan keluar. 320
Kode 10.7 `routes/web.php`: halaman publik, halaman masuk khusus tamu, halaman staf di belakang `auth`, dan daftar harga untuk pemilik. 321
Kode 10.8 `bootstrap/app.php`: rute API, pengukur waktu untuk kedua grup, dan halaman masuk bagi tamu. 323
Kode 10.9 `app/Providers/AppServiceProvider.php`: satu *gate*, `manage-prices`, hanya untuk pemilik. 323
Kode 10.10 `app/Http/Controllers/Api/TokenController.php`: kata sandi masuk, token keluar; dan cara mengembalikannya. 325
Kode 10.11 `routes/api.php`: token, pelanggan, pesanan, dan pembayaran, di belakang `auth:sanctum`. 327
Kode 10.12 `app/Actions/CreateOrder.php`: satu-satunya tempat masukan yang sudah diperiksa menjadi pesanan berharga. 327
Kode 10.13 `app/Http/Controllers/Api/OrderController.php`: mendaftar, membuat, menampilkan, dan memajukan pesanan, menjawab dengan *resource*. 328
Kode 10.14 `app/Http/Requests/UpdateOrderStatusRequest.php`: status yang diminta harus status berikutnya yang diizinkan alur kerja. 330
Kode 10.15 `app/Http/Controllers/Api/PaymentController.php`: mencatat pembayaran, menjawab dengan pesanan yang sudah diperbarui. 332
Kode 10.16 `app/Http/Requests/StoreCustomerRequest.php`: pelanggan baru, dengan telepon yang disederhanakan menjadi angka dan diperiksa agar tidak ganda. 332
Kode 10.17 `app/Http/Resources/OrderResource.php`: bentuk JSON sebuah pesanan. 334
Kode 10.18 `app/Http/Resources/ServiceResource.php`: satu baris daftar harga, sesuai kebutuhan ponsel. 335
Kode 10.19 `tests/Unit/PriceCalculatorTest.php`: enam kasus harga, satu metode uji. 335
Kode 10.20 `tests/Feature/AuthTest.php`: tamu, proses masuk, kata sandi yang salah, daftar harga milik pemilik, dan token API. 336
Kode 10.21 `tests/Feature/OrderApiTest.php`: harga, validasi, alur kerja, pesanan tertutup, pembayaran, dan penyaring, melalui API. 338
Kode 10.22 `tests/Feature/PriceListAccessTest.php`: daftar harga milik pemilik, di setiap rute. 345
Kode 11.1 `pom.xml`: Spring Boot 4.1.1 sebagai induk, Java 25, dan lima dependensi. 351
Kode 11.2 `LaundryApplication.java`: seluruh aplikasi dimulai di sini. 353
Kode 11.3 `application.properties`: basis data di memori, JSON snake_case, dan *problem details* untuk galat. 353
Kode 11.4 `Customer.java`: seorang pelanggan, dengan nomor telepon yang unik. 354
Kode 11.5 `Service.java`: satu baris daftar harga, dengan kodenya sendiri sebagai kunci, serta nama dan satuan yang diturunkan dari deskripsinya. 356
Kode 11.6 `Order.java`: sebuah pesanan, diberi harga ketika dibuat, dimajukan oleh alur kerja toko. 357

Kode 11.7 `Payment.java`: uang yang diterima untuk sebuah pesanan; pesanan boleh dibayar bertahap. 361
Kode 11.8 `OrderStatus.java`: tiga status, masing-masing dengan kata yang dipakai basis data dan JSON. 362
Kode 11.9 `OrderStatusConverter.java`: menyimpan `queued`, bukan `QUEUED`, di setiap kolom status. 362
Kode 11.10 `LaundryRules.java`: aturan harga dan alur kerja, dengan aritmetika bilangan bulat yang sama dengan versi PHP. 363
Kode 11.11 `CustomerRepository.java`: setiap operasi baku untuk pelanggan, ditambah dua metode yang diturunkan dari namanya. 365
Kode 11.12 `OrderRepository.java`: halaman-halaman pesanan, disaring menurut status, dan satu pesanan dengan semua yang diperlukannya. 366
Kode 11.13 `OrderView.java`: JSON satu pesanan, dibangun dari entitasnya. 367
Kode 11.14 `PageView.java`: satu halaman sebagai `data` dan `meta`, dengan halaman dihitung mulai dari 1 seperti di Laravel. 367
Kode 11.15 `CreateOrderRequest.java`: isian pesanan baru dan aturannya, termasuk satu aturan tentang dua isian. 368
Kode 11.16 `ServiceController.java`: daftar harga yang aktif, dan satu layanan menurut kodenya. 369
Kode 11.17 `OrderController.java`: mendaftar, menampilkan, membuat, dan memajukan pesanan, masing-masing di dalam transaksi. 370
Kode 11.18 `DemoData.java`: daftar harga sungguhan dari `Service.csv`, serta tiga pelanggan Bab 5 dan Bab 7 beserta pesanannya. 373
Kode 11.19 `LaundryRulesTest.java`: aturan harga dan alur kerja, diuji tanpa Spring. 376
Kode 11.20 `CustomerOrdersController.java`: pesanan seorang pelanggan, yang terbaru lebih dahulu, atau 404. 380
Kode 12.1 `Api/ServiceController.php` (Laravel): daftar harga yang aktif sebagai koleksi *resource*. 386
Kode 12.2 `routes/api.php` (Laravel): rute Bab 10, daftar harga, dan laporan harian. 387
Kode 12.3 `DataView.java` (Spring): amplop kontrak dalam satu baris. 388
Kode 12.4 `ServiceController.java` (Spring): daftar yang sama, kini di dalam amplop. 388
Kode 12.5 `OrderResource.php` (Laravel): setiap waktu dalam format kontrak. 389
Kode 12.6 `OrderView.java` (Spring): waktu dengan pola tetap, dan pembayaran hanya bila diminta. 390
Kode 12.7 `ApiValidationException.java` (Spring): aturan yang dilanggar, beserta pesan per isian. 392
Kode 12.8 `ApiExceptionHandler.java` (Spring): galat validasi, aturan yang dilanggar, *exception* status, dan badan yang tidak terbaca, masing-masing dalam bentuk kontrak. 393
Kode 12.9 `CreateOrderRequest.java` (Spring): hanya aturan satu isian. 396
Kode 12.10 `OrderController.java` (Spring): semua galat sekaligus, alur kerja sebagai 422, pesanan tertutup sebagai 409, dan pembayaran. 396
Kode 12.11 `CreatePaymentRequest.java` (Spring): jumlah dan metode pembayaran. 402
Kode 12.12 `PaymentView.java` (Spring): satu pembayaran, dengan waktunya dalam format kontrak. 403
Kode 12.13 `Api/ReportController.php` (Laravel): hitungan per status, total dan pembayaran hari itu, dengan setiap status hadir. 404
Kode 12.14 `OrderRepository.java` (Spring): kueri-kueri Bab 11 ditambah dua kueri yang diperlukan laporan. 406
Kode 12.15 `ReportController.java` (Spring): laporan yang sama, dengan `LinkedHashMap` yang mendaftar setiap status secara berurutan. 407
Kode 12.16 `contract/check_api.php`: enam belas pemeriksaan kontrak, terhadap server mana pun. 408
Kode 12.17 `tests/Feature/ContractTest.php` (Laravel): hal-hal baru dalam kontrak, dengan jam yang ditetapkan. 412

Kode 12.18 `OrderApiTest.java` (Spring): aturan harga, semua galat sekaligus, seluruh kehidupan sebuah pesanan, dan laporan, melalui MockMvc. 414
Kode 13.1 `RuleException.java`: masukan yang melanggar aturan, dengan pesan per field. 424
Kode 13.2 `NotFoundException.java`: record yang tidak ada. 424
Kode 13.3 `ConflictException.java`: permintaan yang tidak diizinkan oleh keadaan record saat ini. 424
Kode 13.4 `Customer.java`: pelanggan dari Bab 11 dengan satu metode baru, `change`. 425
Kode 13.5 `CustomerRepository.java`: kueri yang diturunkan dari nama metode. 426
Kode 13.6 `CustomerService.java`: mencari, membuat, mengubah, dan menghapus pelanggan, di dalam transaksi. 427
Kode 13.7 `LaundryApplication.java`: titik masuk, dan jam toko dalam Waktu Indonesia Barat. 430
Kode 13.8 `OrderRepository.java`: halaman-halaman pesanan, satu pesanan beserta rinciannya, dan kueri laporan. 430
Kode 13.9 `OrderService.java`: menerima pesanan, memajukannya, dan menerima uang, dengan setiap aturan di satu tempat. 431
Kode 13.10 `ReportService.java`: angka-angka satu hari. 436
Kode 13.11 `OrderApiController.java`: baca JSON, panggil `OrderService`, tulis JSON. 438
Kode 13.12 `CustomerApiController.java`: pelanggan dalam JSON. 440
Kode 13.13 `ServiceApiController.java`: daftar harga dalam JSON. 440
Kode 13.14 `ReportApiController.java`: laporan harian dalam JSON. 441
Kode 13.15 `ApiExceptionHandler.java`: setiap galat API dalam bentuk kontrak. 442
Kode 13.16 `CustomerForm.java`: formulir pelanggan, dengan batas yang sama seperti permintaan API. 445
Kode 13.17 `OrderForm.java`: formulir pesanan baru, yang tahu cara menjadi `NewOrder`. 446
Kode 13.18 `PaymentForm.java`: formulir pembayaran. 446
Kode 13.19 `Forms.java`: pesan aturan dari service menjadi galat pada field formulir. 447
Kode 13.20 `CustomerController.java`: CRUD lengkap untuk pelanggan sebagai halaman. 447
Kode 13.21 `fragments.html`: kepala halaman, navigasi, dan pesan *flash*. 451
Kode 13.22 `customers/list.html`: daftar pelanggan yang dapat dicari. 452
Kode 13.23 `customers/form.html`: satu formulir untuk menambah dan mengubah, dengan pesan di samping setiap field. 453
Kode 13.24 `OrderController.java`: antrean, pesanan baru, satu pesanan, langkah status, dan pembayaran. 454
Kode 13.25 `HomeController.java`: halaman depan adalah antrean. 458
Kode 13.26 `orders/list.html`: satu halaman pesanan, disaring menurut status. 458
Kode 13.27 `orders/form.html`: menerima sekantong cucian. 459
Kode 13.28 `orders/detail.html`: pesanan dengan total, pembayaran, dan sisa tagihannya, serta tombol untuk langkah berikutnya. 460
Kode 13.29 `WebErrorAdvice.java`: setiap `NotFoundException` di halaman menjadi halaman 404. 462
Kode 13.30 `error/404.html`: halaman untuk record yang tidak ada. 462
Kode 13.31 `pom.xml`: *build* Bab 12 ditambah Thymeleaf dan SQLite; H2 tetap untuk tes. 463
Kode 13.32 `application.properties`: SQLite dalam berkas, skema dijaga mutakhir, JSON snake_case. 464
Kode 13.33 `src/test/resources/application.properties`: basis data H2 baru untuk setiap kali tes dijalankan. 464
Kode 13.34 `OrderServiceTest.java`: lapisan service dengan repository tiruan dan jam yang tetap. 466
Kode 14.1 `Role.java`: dua peran sudah cukup bagi toko. 477
Kode 14.2 `Staff.java`: orang yang boleh login, dengan *hash* kata sandi dan peran. 477
Kode 14.3 `StaffRepository.java`: mencari staf menurut e-mail. 478
Kode 14.4 `StaffDetailsService.java`: dari e-mail ke pandangan Spring Security tentang seorang pengguna. 479

Kode 14.5 `DemoStaffSeeder.java`: pemilik dan satu kasir, dengan kata sandi yang di-*hash*.....480
Kode 14.6 `ApiToken.java`: kunci API milik satu perangkat.....481
Kode 14.7 `ApiTokenRepository.java`: metode-metode bakunya sudah cukup.....482
Kode 14.8 `TokenService.java`: menerbitkan, mencari, dan mencabut token, hanya menyimpan *hash*.....482
Kode 14.9 `BearerTokenFilter.java`: dari token *bearer* ke staf yang terautentikasi.....485
Kode 14.10 `TokenApiController.java`: `POST /api/tokens` memberi token, `DELETE /api/tokens/current` mengambilnya kembali.....487
Kode 14.11 `SecurityConfig.java`: satu rantai untuk API, satu untuk halaman.....488
Kode 14.12 `LoginController.java`: hanya `GET`; Spring Security menangani `POST`.....492
Kode 14.13 `login.html`: e-mail dan kata sandi, dengan pesan untuk login yang gagal dan untuk logout.....492
Kode 14.14 `fragments.html`: navigasi kini diakhiri formulir logout.....492
Kode 14.15 `CustomerService.java`: `delete` hanya untuk pemilik, siapa pun yang memanggilnya.....493
Kode 14.16 `CustomerApiController.java`: `DELETE /api/customers/{id}` yang baru.....495
Kode 14.17 `pom.xml`: sekuriti, dan satu *starter* tes per *slice*.....500
Kode 14.18 `CustomerApiControllerTest.java`: lapisan web dan aturan sekuriti, dengan service tiruan.....502
Kode 14.19 `OrderRepositoryTest.java`: kueri laporan dan `existsByCustomerId` terhadap H2.....504
Kode 14.20 `TokenFlowTest.java`: masuk, pakai token, keluar, lalu lihat token itu ditolak.....505
Kode 15.1 `PerfLab/Program.cs`: *endpoint* penawaran harga sebagai *minimal API* ASP.NET Core.....517
Kode 15.2 `routes/api.php`: *endpoint* penawaran harga sebagai rute Laravel.....518
Kode 15.3 `QuoteController.java`: *endpoint* penawaran harga sebagai controller Spring Boot.....518
Kode 15.4 `LoadGen/Program.cs`: pembangkit beban yang melaporkan *throughput* dan persentil latensi.....520
Kode 16.1 `check_api.php`: 25 pemeriksaan kontrak, untuk server mana pun dari ketiganya.....536
Kode 16.2 `LaundryCapstone.csproj`: sakelar `UseEf` memutuskan apakah EF Core ikut dikompilasi.....544
Kode 16.3 `LaundryRules.cs`: harga dan alur kerja dalam aritmetika bilangan bulat, seperti di Bab 8 dan 11.....544
Kode 16.4 `ILaundryStore.cs`: semua yang diperlukan aplikasi dari penyimpanan.....546
Kode 16.5 `LaundryService.cs`: pelanggan, pesanan, langkah status, pembayaran, dan angka harian, dengan setiap aturan di satu tempat.....546
Kode 16.6 `Exceptions.cs`: aturan yang dilanggar, record yang tidak ada, keadaan yang bertentangan.....551
Kode 16.7 `Dtos.cs`: *record* permintaan dengan atribut validasi, dan tampilan dalam bentuk kontrak.....551
Kode 16.8 `OrdersController.cs`: baca JSON, panggil `LaundryService`, tulis JSON.....554
Kode 16.9 `ApiErrors.cs`: setiap galat dalam bentuk kontrak.....555
Kode 16.10 `CustomersController.cs`: telepon disimpan sebagai angka, dan `DELETE` hanya untuk pemilik.....557
Kode 16.11 `BearerTokenHandler.cs`: dari token *bearer* ke staf yang sudah login, dengan 401 dan 403 milik kontrak.....558
Kode 16.12 `TokenService.cs`: token bergaya Sanctum, di-*hash* dengan SHA-256 dan dibandingkan dalam waktu konstan.....559
Kode 16.13 `Program.cs`: penyimpanan, JSON, galat, sekuriti, dan data awal, secara berurutan.....561
Kode 16.14 `BoardController.cs`: antrean dan pesanan yang sudah selesai, tanpa login.....564
Kode 16.15 `Views/Board/Index.cshtml`: nomor pesanan dan nama depan, disegarkan setiap 30 detik.....564
Kode 16.16 `routes/api.php`: rute `DELETE` baru, hanya untuk pemilik.....566
Kode 16.17 `AppServiceProvider.php`: *gate* Bab 10 dan `delete-customers` yang baru.....567
Kode 16.18 `CustomerController.php`: `destroy` menolak pelanggan yang punya pesanan dengan 409.....568

Kode 16.19 `routes/web.php`: papan adalah rute publik. 569
Kode 16.20 `BoardController.php`: antrean dan pesanan yang sudah selesai. 570
Kode 16.21 `board.blade.php`: papan yang sama dengan milik Razor, dalam Blade. 571
Kode 16.22 `CapstoneTest.php`: penghapusan oleh kasir, dua penghapusan oleh pemilik, dan privasi papan. 572
Kode 16.23 `BoardController.java`: papan membaca pesanannya melalui `OrderService`. 573
Kode 16.24 `board.html`: papan yang sama dalam Thymeleaf. 574

Daftar Persamaan

(5.1) 131
(8.1) 242
(8.2) 242
(12.1) 385
(12.2) 404
(13.1) 436
(13.2) 436
(14.1) 477
(14.2) 480
(15.1) 517
(15.2) 521
(16.1) 535
(16.2) 535

Kata Pengantar

Setiap aplikasi web menjawab pertanyaan yang sama ribuan kali sehari: sebuah peramban meminta sebuah alamat, dan program harus menemukan potongan kode yang tepat, mengumpulkan data yang tepat, mengubahnya menjadi halaman atau jawaban, lalu mengirimkannya kembali — dengan benar, aman, dan cepat. Anda bisa menulis semua itu sendiri, dan bab pertama buku ini melakukannya, sekali saja, agar Anda melihat betapa banyak pekerjaannya. Kerangka kerja (*framework*) adalah keputusan untuk tidak menulisnya lagi: seseorang sudah membangun perutean, akses basis data, formulir, proses masuk (*sign-in*), dan pengujian, dan Anda hanya mengisi apa yang khas bagi aplikasi Anda. Belajar bekerja *bersama* kerangka kerja seperti itu — tahu di mana setiap bagian kode Anda seharusnya berada, apa yang dikerjakan kerangka kerja untuk Anda, dan apa yang masih diserahkannya kepada Anda — itulah pemrograman berbasis kerangka kerja, dan itulah seluruh pokok bahasan buku ini. Pegang kalimat itu, dan tidak ada yang terasa sewenang-wenang di halaman-halaman berikut.

Buku ini tumbuh dari mata kuliah Pemrograman Berbasis Kerangka Kerja yang diajarkan di Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember (ITS), Surabaya, dan telah disusun ulang sehingga dapat berdiri sepenuhnya sendiri. Keenam belas babnya mengikuti irama satu semester enam belas minggu, tetapi Anda tidak memerlukan kelas, dosen, maupun dokumen lain untuk belajar darinya.

Untuk siapa buku ini ditulis

Anda diandaikan sudah menguasai dasar pemrograman web: apa itu HTML dan formulir, apa yang terjadi ketika peramban mengirim permintaan dan server menjawabnya, dan cara menulis program kecil dengan variabel, perulangan, fungsi, dan kelas dalam sedikitnya satu bahasa. Anda *tidak* diandaikan mengenal C#, PHP, atau Java, kerangka kerja apa pun, atau cara memasangnya. Setiap istilah diperkenalkan dengan kata-kata sederhana sebelum dipakai, setiap kerangka kerja dipasang langkah demi langkah di Windows, macOS, dan Linux, dan setiap gagasan diperlihatkan bekerja lebih dulu sebelum dijelaskan secara umum.

Prinsip itu membuat buku ini panjang. Buku yang lebih pendek akan mengandaikan Anda sudah memahami bagian-bagian yang sulit, dan buku seperti itu tidak berguna bagi orang yang belum memahaminya.

Satu aplikasi, tiga kerangka kerja

Kerangka kerja adalah cara menata sebuah program, bukan fitur milik satu bahasa, dan buku ini memperlakukannya demikian. Buku ini mengajarkan tiga kerangka kerja sisi server yang paling banyak dipakai, masing-masing dalam bahasanya sendiri: **ASP.NET Core** dengan C#, **Laravel** dengan PHP, dan **Spring Boot** dengan Java. Dan buku ini membangun aplikasi yang sama di ketiganya: **Profitina Laundry**, usaha binatu di Surabaya dengan pelanggannya, daftar harganya, pesanannya, dan pembayarannya. Empat tabel yang sama, aturan yang sama tentang harga dan status pesanan, dan pada akhirnya API JSON yang sama, yang diperiksa oleh skrip uji yang sama, muncul tiga kali, dalam tiga kosakata. Melihat satu aplikasi ditulis dengan tiga cara adalah jalan tercepat untuk memahami gagasan yang dimiliki bersama oleh setiap kerangka kerja — Model–View–Controller, perutean, pemetaan objek-relasional (ORM), injeksi dependensi, dan templat —

bukan sekadar kebiasaan satu perkakas, sehingga kerangka kerja apa pun yang dipakai dalam proyek atau pekerjaan Anda kelak, Anda sudah tahu apa yang ada di baliknya.

Isi buku ini

Bab 1 sampai 3 membangun landasan: mengapa kerangka kerja ada, diperlihatkan dengan menulis daftar harga kecil yang sama empat kali — tanpa kerangka kerja dalam C#, PHP, dan Java, lalu dengan ASP.NET Core — disertai penyiapan lengkap VS Code, .NET SDK, PHP dan Composer, JDK dan Maven, serta basis data di Windows, macOS, dan Linux; fondasi bersama kerangka kerja modern — MVC, perutean, ORM, injeksi dependensi, dan templat — yang dibangun sendiri dalam kerangka kerja mungil milik kita, MiniFrame, lalu ditemukan kembali di ketiga kerangka kerja sungguhan; dan aplikasi ASP.NET Core MVC sungguhan yang pertama: proyek, `Program.cs` dengan layanan dan *pipeline* permintaannya, model, *controller* dan perutean, serta *view* Razor. Bab 5 sampai 7 memberi aplikasi itu ingatan dan bahasa kedua: Entity Framework Core, LINQ, migrasi, dan CRUD sungguhan untuk pelanggan, pesanan, pembayaran, dan daftar harga; validasi, kelas formulir, ASP.NET Core Identity untuk proses masuk, dan penerapan dari laptop ke server; dan toko yang sama dalam Laravel, dengan Artisan, migrasi yang ditulis dalam PHP, model Eloquent, *routing*, *controller*, dan templat Blade. Bab 9 sampai 11 membawa aplikasi lebih jauh dan ke kerangka kerja ketiga: migrasi Laravel secara mendalam, *factory* dan *seeder*, *middleware* dan *form request*; pengguna dan peran, proses masuk, API untuk program lain dengan *API resource*, serta pengujian; dan Spring Boot, dengan Initializr, entitas, *repository* Spring Data JPA, dan REST *controller*. Bab 13 sampai 15 menuntaskan pekerjaan: aplikasi Spring Boot berlapis dengan lapisan layanan, validasi, halaman Thymeleaf, dan satu tempat untuk galat; Spring Security dengan token untuk API, desain REST API, dan pengujian otomatis di empat tingkat; serta perbandingan ketiga kerangka kerja — kinerja, produktivitas, ekosistem dan komunitas, kalender dukungan, dan keamanan bawaan — yang diakhiri dengan cara memilih di antara ketiganya.

Bab 4, 8, 12, dan 16 adalah bab **Evaluasi dan Pembahasan**. Masing-masing menutup seperempat buku dengan menyelesaikan soal-soal utuh secara lengkap — setiap soal dinyatakan ulang, dirancang, diprogram, diuji, dijalankan bila memungkinkan, dan dijelaskan — dengan penyelesaian keliru yang menggoda ditampilkan lalu dibedah. Soal-soalnya berkisar dari situs etalase dengan pelacakan pesanan, tabel rute yang ditulis sendiri, *header* waktu respons, dan mode pemeliharaan; lalu satu aturan harga dan satu alur kerja pesanan yang diselesaikan di ASP.NET Core maupun Laravel, daftar tanpa masalah N+1, pencarian yang menemukan pelanggannya, dan pembayaran yang tidak mungkin berlebih; sampai satu kontrak REST API — satu cara menulis waktu, satu bentuk untuk setiap galat, pesanan yang berpindah status dan ditutup, laporan harian — yang dipenuhi Laravel dan Spring Boot sekaligus; dan akhirnya proyek puncak: Profitina Laundry lengkap di ketiga kerangka kerja, dibandingkan berdampingan, lalu sebuah keputusan.

Sepanjang buku, satu usaha kecil tumbuh bab demi bab: **Profitina Laundry**, binatu yang bermula dari satu mesin cuci, dua pengering, dan sebuah buku catatan, lalu tumbuh menjadi tiga gerai di Surabaya. Situs lamanya terdiri atas dua puluh berkas yang tidak berani disentuh siapa pun; keinginannya sederhana — *satu perubahan dikerjakan di satu tempat* — dan buku ini mengabulkannya satu kerangka kerja demi satu kerangka kerja: situs publik dengan layanan dan penawaran harganya, aplikasi kasir dengan basis data sungguhan, staf yang harus masuk dulu, API untuk aplikasi ponsel para pengantar, dan akhirnya toko yang sama dalam tiga kerangka kerja, dengan orang-orang yang menjalankannya memilih di antara ketiganya. Setiap bab juga

memiliki subbab *Memakainya Sungguhan* yang menunjukkan bagaimana gagasan yang sama terbawa ke usaha kecil lain dan masalah sehari-hari, dan beberapa bab memiliki subbab *Profitina dalam Praktik* yang memperlihatkan gagasan itu bekerja dalam produk sungguhan.

Bagaimana kode diperiksa

Setiap program dalam buku ini dikumpulkan, bab demi bab, dalam folder `Code` yang menyertai buku ini, dengan proyek `aspnet`, `laravel`, atau `spring` di setiap bab (hanya kerangka kerja yang dipakai bab itu), beserta README yang menjelaskan cara membangun serta menjalankan setiap proyek di terminal dan di VS Code, pada Windows, macOS, dan Linux. Setiap kode dalam buku ini adalah berkas di folder itu, karakter demi karakter.

Kami perlu jujur tentang sejauh mana kode itu dijalankan. Mesin tempat buku ini ditulis memiliki .NET 10 SDK, PHP, dan JDK, tetapi tidak dapat mengunduh paket dari NuGet, Packagist, maupun Maven Central. Karena itu, semua yang hanya membutuhkan .NET itu sendiri — program Bab 1 sampai 4, MiniFrame, demonstrasi-demonstrasi kecil di bab-bab berikutnya, dan aplikasi ASP.NET Core pada proyek puncak dalam versi yang menyimpan datanya di memori — dibangun dan dijalankan di sana, dan keluaran serta tangkapan layarnya di buku ini nyata. Aplikasi Entity Framework Core, Laravel, dan Spring Boot tidak dapat dibangun di sana. Kodenya ditulis berdasarkan dokumentasi resmi terkini, setiap berkas PHP diperiksa dengan `php -l`, setiap berkas Java dikompilasi dengan `javac` terhadap tiruan (*stand-in*) tipe-tipe pustakanya, aturan bisnis bersama dijalankan dalam PHP dan Java biasa, dan pemeriksa kontrak API Bab 12 dan 16 dijalankan terhadap server rujukan. Di mana buku ini menampilkan keluaran aplikasi-aplikasi tersebut, keluaran itu ditandai sebagai keluaran yang diharapkan, dan tidak ada tangkapan layar aplikasi itu yang dikarang; diagram buatan kami sendiri menggantikannya. README setiap bab menyebutkan dengan tepat apa yang dijalankan dan apa yang tidak, serta menyebut perintah pertama yang sebaiknya Anda jalankan sendiri — `dotnet build`, `composer install` lalu `php artisan test`, atau `mvn test` — karena di mesin Anda, dengan paket-paketnya terunduh, perintah itulah pemeriksaan yang sesungguhnya.

Cara membacanya

Dengan editor terbuka dan kerangka kerja terpasang. Ketikkan contohnya, jangan hanya menyalinnya; saat mengetik itulah kesalahan terjadi, dan dari kesalahan itulah pelajaran didapat. Sebelum mengirim sebuah permintaan, tebaklah apa yang akan dilakukan kerangka kerja terhadapnya — rute mana yang cocok, *controller* mana yang berjalan, apa yang ditanyakan ke basis data, *view* mana yang menjawab — lalu jalankan dan bandingkan. Setelah sebuah contoh berjalan, rusakkan dengan sengaja lalu baca apa yang dikatakan halaman galat atau lognya. Bacalah ketiga kerangka kerja berdampingan ketika Anda sampai padanya: bila sesuatu tampak baru di Laravel atau Spring Boot, tanyakan di mana Anda sudah menjumpainya di ASP.NET Core. 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 5.3” 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 ITS sejak 1998. Ia membimbing dan menguji tugas akhir mahasiswa sarjana, mengajar MLOps dalam program IBM SkillsBuild, dan membangun produk: ia mendirikan Profitina, platform visibilitas AI, dan Profitina Laundry, contoh berjalan buku ini, dipinjam dari keluarga studi kasusnya. 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 mahasiswa yang mempelajari versi-versi awal materi ini dan mengajukan pertanyaan yang menyingkapkan di mana sebuah penjelasan masih tipis, serta kepada rekan-rekan di ITS yang bersama-sama mengajarkan dan menyempurnakan mata kuliah ini dari tahun ke tahun. Standar buku ini juga berutang pada dokumentasi resmi ASP.NET Core, Entity Framework Core, Laravel, Spring Boot, dan Spring Security, serta komunitas yang merawatnya, yang dikutip di setiap tempat penggunaannya.

Surabaya, September 2026



Irfan Subakti · yifana@gmail.com

Profitina

profitina.com

Bab 1

Dari Pemrograman Web ke Kerangka Kerja

Setiap aplikasi web menjawab pertanyaan yang sama ribuan kali sehari: peramban meminta sebuah alamat, lalu program harus menemukan potongan kode yang tepat, mengumpulkan data yang tepat, mengubahnya menjadi halaman, dan mengirimkannya kembali tanpa satu kesalahan pun. Semua itu bisa Anda tulis sendiri, dan pelajaran pemrograman web yang pertama memang sepatutnya meminta Anda melakukannya sekali. Bab ini memperlihatkan, lewat satu program kecil yang ditulis empat kali, apa yang berubah ketika sebuah *framework* (kerangka kerja) mengambil alih sebagian besar pekerjaan itu. Pada akhir bab ini Anda akan mampu menjelaskan apa itu kerangka kerja dan bedanya dengan pustaka (*library*); menguraikan pola Model–View–Controller yang dipakai bersama oleh ASP.NET Core, Laravel, dan Spring Boot; menjelaskan mengapa buku ini memilih tepat ketiganya; memasang seluruh perkakas kerja (VS Code, .NET 10 SDK, PHP 8.5 dengan Composer, JDK 25 dengan Maven, SQLite dan MySQL) di Windows, macOS, atau Linux; membangun dan menjalankan daftar harga Profitina Laundry secara manual dalam C#, PHP, dan Java; membangunnya lagi dengan ASP.NET Core dengan kode yang jauh lebih sedikit; serta menyebutkan dengan tepat pekerjaan apa saja yang berpindah dari kode Anda ke kerangka kerja.

1.1 Daftar Harga Bu Wati

Profitina Laundry membuka konter pertamanya di Rungkut, Surabaya, pada Januari 2019. Bu Wati Wijaya, pemiliknya, memulai dengan satu mesin cuci, dua pengering, sebuah timbangan yang menghadap ke pelanggan, dan sebuah buku catatan untuk pesanan. Kini ada tiga gerai (Rungkut, Keputih, dan sejak Januari 2026, Wiyung), empat belas orang karyawan, dua belas layanan mulai dari “cuci, kering, lipat” seharga Rp 8.000 per kilogram hingga “cuci sepatu menyeluruh” seharga Rp 45.000 per pasang, dan hampir tiga ribu pesanan setahun.

Beberapa tahun lalu, seorang mahasiswa yang membantu di konter membuatkan toko itu sebuah situs web kecil: halaman beranda, halaman daftar harga, dan satu halaman untuk tiap layanan. Situs itu persis jenis situs yang diajarkan pada pelajaran pemrograman web yang pertama, dan situs itu berjalan. Setiap halaman adalah berkas PHP tersendiri. Setiap berkas membuka basis data, menjalankan sebuah kueri, lalu mencetak HTML dengan `echo`. Bagian kepala halaman yang berisi nama toko dan menu disalin ke setiap berkas.

Kemudian toko itu tumbuh, dan situsnya mulai menyusahkan. Ketika gerai Wiyung dibuka, alamat baru harus diketik di lima berkas; satu terlupa, dan selama dua minggu halaman “kontak” mengarahkan pelanggan ke jalan yang salah. Ketika harga setrika naik, harga itu berubah di daftar harga tetapi tidak di halaman layanan, yang punya kuerinya sendiri. Seorang pelanggan mengetik alamat yang aneh di peramban, kira-kira `/service.php?id=<script>…`, dan halaman itu mencetaknya kembali ke dalam HTML tanpa di-*escape*; kali itu tidak terjadi apa-apa, tetapi mahasiswa yang memeriksanya langsung pucat. Dan ketika Bu Wati meminta hal sederhana, “bisakah pelanggan melihat apakah cucian mereka sudah siap?”, tidak ada yang berani

menyentuh kodenya, karena tidak ada yang bisa memastikan mana di antara dua puluh berkas itu yang akan rusak.

Keinginan Bu Wati, ketika akhirnya ia jelaskan, sangat bersahaja: *satu perubahan cukup dilakukan di satu tempat*. Alamat baru cukup diketik sekali. Sebuah harga tinggal di satu tempat. Setiap halaman tampil seragam tanpa ada yang menyalin bagian kepalanya. Setiap nilai yang dicetak ke halaman dibuat aman secara otomatis, bukan hanya ketika ada yang ingat. Dan fitur baru punya tempat yang jelas.

Keinginan itulah alasan keberadaan kerangka kerja. Buku ini mengikuti Profitina Laundry melalui tiga kerangka kerja. Toko dan datanya tetap sama dari bab pertama hingga terakhir: pelanggan, gerai, layanan beserta harganya, pesanan (basis data menyebutnya *bill*), pembayaran, dan status setiap pesanan. Yang berganti adalah kerangka kerjanya, sehingga Anda dapat melihat apa yang dikerjakan masing-masing untuk Anda dan apa harganya. Bab ini dimulai dari potongan terkecil, yaitu daftar harga, dan membangunnya empat kali.

1.2 Gagasannya dalam Kata-Kata Sederhana

1.2.1 Apa yang dikerjakan aplikasi web untuk setiap permintaan

Jika semua rincian dikupas, setiap permintaan ke setiap aplikasi web melewati delapan langkah yang sama. Bayangkan seorang pelanggan membuka `http://…/services/S0004` di ponselnya untuk melihat harga *dry cleaning* sebuah kemeja.

1. **Menunggu** koneksi dan membaca permintaan: metodenya (`GET`), path-nya (`/services/S0004`), dan header-nya.
2. **Merutekan**: memutuskan potongan kode mana yang bertanggung jawab atas path dan metode ini.
3. **Mengekstrak**: memotong bagian yang berubah-ubah dari path (di sini kode `S0004`) dan mengubahnya ke tipe yang tepat.
4. **Mengambil**: meminta layanan dengan kode itu dari penyimpanan data.
5. **Memutuskan**: menerapkan aturan bisnis (adakah layanan seperti itu? bolehkah pelanggan melihatnya?).
6. **Merender**: mengubah data menjadi HTML (atau JSON untuk program lain), dengan meng-*escape* setiap nilai agar teks dari luar tidak pernah bisa menjadi kode.
7. **Menjawab**: memilih kode status (`200 OK`, `404 Not Found`, …), mengatur header, mengirim byte.
8. **Bertahan**: mencatat apa yang terjadi, dan jika ada yang salah, mengirim halaman galat yang sopan alih-alih berhenti mendadak.

Hanya langkah 4 dan 5 yang berkaitan dengan *laundry*. Langkah 1, 2, 3, 6, 7, dan 8 sama saja untuk laundry, klinik, sekolah, atau bank. Program yang mengerjakan kedelapannya secara manual menghabiskan sebagian besar barisnya untuk hal yang tidak ada hubungannya dengan bisnisnya, dan justru di baris-baris itulah kebanyakan bug bersembunyi: `if` yang keliru pada perutean, id yang dipotong satu karakter terlalu awal, nilai yang dicetak tanpa di-*escape*.

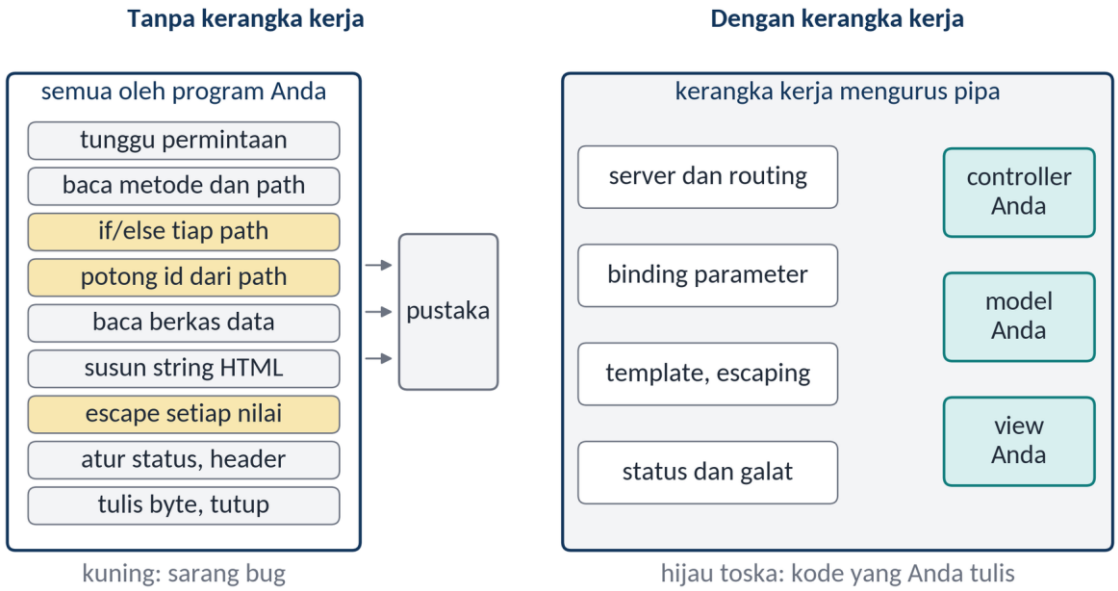
1.2.2 Pustaka atau kerangka kerja: siapa memanggil siapa?

Para pemrogram sejak dulu berbagi kode. **Pustaka** (*library*) adalah kumpulan rutin siap pakai yang dipanggil oleh program *Anda* ketika diperlukan: fungsi yang mengurai tanggal, kelas yang

membaca berkas CSV, rutin yang meng-*encode* HTML. Program Anda yang memegang kendali; program Anda yang menentukan urutan pekerjaan dan sesekali memanggil pustaka.

Kerangka kerja (*framework*) membalik hubungan itu. Kerangka kerja adalah program yang hampir lengkap tetapi berlubang, dan *Andalah* yang mengisi lubang-lubangnya. Kerangka kerja menunggu permintaan, merutekannya, mengekstrak parameter, memanggil kode Anda pada saat yang tepat, mengambil apa yang dikembalikan kode Anda, merendernya, lalu mengirimkannya. Ralph Johnson dan Brian Foote menjelaskan hal ini pada tahun 1988, jauh sebelum ada web: kerangka kerja adalah rancangan yang dapat dipakai ulang, yang alur kendalinya milik kerangka kerja, sementara aplikasi menyediakan bagian-bagian yang berbeda [1]. Gagasan yang sama sering diringkas sebagai *Hollywood principle* (prinsip Hollywood): "jangan hubungi kami, nanti kami yang menghubungi Anda" [2]. Nama bakunya adalah **inversion of control** (pembalikan kendali), karena alur kendalinya terbalik: kerangka kerja yang memanggil kode Anda, bukan kode Anda yang memanggil pustaka [3].

Gambar 1.1 menyandingkan keduanya. Di kiri, satu program mengerjakan sendiri kesembilan pekerjaan kecil sebuah permintaan dan sesekali memanggil pustaka; pekerjaan berwarna kuning adalah sumber bug situs lama Bu Wati. Di kanan, kerangka kerja mengurus "pipa-pipanya" dan memanggil tiga potongan kecil kode Anda: controller, model, dan view.



Gambar 1.1 Pustaka dipanggil oleh program Anda; kerangka kerja memanggil program Anda. Dengan kerangka kerja, satu-satunya kode yang Anda tulis adalah bagian yang menyangkut bisnis Anda.

> **DEFINISI.** *Pustaka* adalah kode yang dipanggil program Anda. *Kerangka kerja* adalah kode yang memanggil program Anda: ia memiliki gelung utama dan urutan kejadian, lalu memanggil potongan-potongan yang Anda sediakan (*handler*, controller, view) pada saat yang ia tentukan. Pembalikan ini disebut *inversion of control*. *Kerangka kerja web* adalah kerangka kerja yang gelung utamanya adalah siklus permintaan–respons HTTP.

Satu akibatnya sering mengejutkan pemula: dalam proyek berkerangka kerja, Anda sering tidak menemukan `main`. Di ASP.NET Core, `main` berupa `Program.cs` pendek yang membangun aplikasi lalu memanggil `app.Run()`, dan sesudah itu kerangka kerja mengambil alih; di Laravel berupa `public/index.php`, yang tidak pernah Anda sunting; di Spring Boot hanya satu baris, `SpringApplication.run(...)`. Kode Anda tersebar di controller dan view yang tampaknya tidak dipanggil oleh apa pun di berkas Anda sendiri. Kerangka kerjalah yang memanggilnya, mengikuti *konvensinya*: kelas bernama `ServicesController` di folder `Controllers`, view bernama

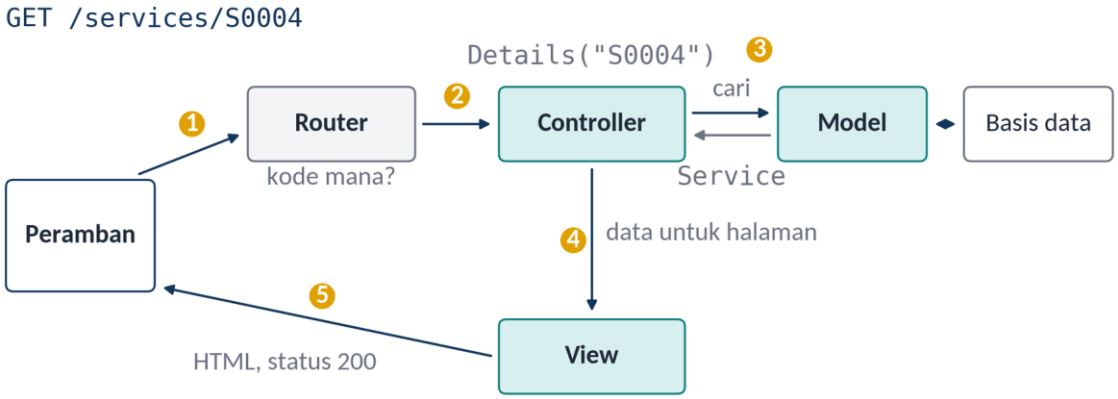
`Details.cshtml` di folder `Views/Services`. Mempelajari kerangka kerja, sebagian besar, adalah mempelajari konvensinya.

1.2.3 Model, View, Controller

Hampir setiap kerangka kerja web sisi server menata lubang-lubang yang disediakannya untuk Anda dengan cara yang sama, yang disebut **Model–View–Controller**, disingkat MVC. Pola ini ditemukan di Xerox PARC untuk antarmuka pengguna Smalltalk-80 dan pertama kali diuraikan secara tertulis oleh Krasner dan Pope pada tahun 1988 [4]; kerangka kerja web kemudian menyesuaikannya dengan siklus permintaan–respons.

> **DEFINISI.** Dalam aplikasi web yang ditata sebagai *Model–View–Controller*, **model** menyimpan data dan aturan bisnis (sebuah layanan dan harganya, aturan "minimal 3 kg"); **view** mengubah data menjadi apa yang dilihat pengguna, biasanya berupa templat HTML; dan **controller** menerima satu permintaan, meminta apa yang diperlukan kepada model, memilih view, dan menyerahkan data kepadanya. **Router**, bagian dari kerangka kerja, menentukan metode controller mana (disebut *action*, aksi) yang menangani setiap permintaan.

Gambar 1.2 mengikuti permintaan pelanggan untuk `/services/S0004` melalui lima langkah MVC. Peramban mengirim permintaan (1). Router menemukan bahwa metode `Details` pada controller layanan yang bertanggung jawab, lalu memanggilnya dengan `"S0004"` (2). Controller meminta layanan berkode itu kepada model, dan model mengambilnya dari basis data (3). Controller menyerahkan objek `Service` kepada sebuah view (4), dan HTML dari view kembali ke peramban dengan status `200 OK` (5).



Gambar 1.2 Satu permintaan melalui Model–View–Controller: router memilih aksi controller, controller bertanya kepada model dan meneruskan hasilnya ke view, dan HTML dari view menjadi responsnya.

Mengapa satu halaman dipecah menjadi tiga bagian? Karena ketiganya berubah dengan alasan dan kecepatan yang berbeda. Harga setrika adalah urusan model dan harus berubah di satu tempat, halaman mana pun yang menampilkannya. Warna kepala halaman adalah urusan view; mengubahnya tidak boleh membahayakan harga. Halaman mana yang menjawab alamat mana adalah urusan controller. Ketika setiap jenis perubahan punya satu rumah, keinginan Bu Wati, "satu perubahan di satu tempat", terwujud hampir dengan sendirinya.

> **PERINGATAN: CONTROLLER GEMUK, MODEL KURUS.** Kesalahan struktural yang paling umum pada proyek berkerangka kerja pertama adalah menaruh aturan bisnis di controller, karena di situlah permintaan datang. Controller yang menghitung harga, memeriksa berat minimum, dan menyusun string SQL tidak dapat dipakai ulang oleh halaman lain, oleh *endpoint* JSON untuk aplikasi seluler, maupun oleh pengujian. Jagalah controller tetap ramping: terima masukan, panggil satu atau dua metode model, pilih view. Bab 3 memperlihatkan bedanya lewat aturan penawaran harga Profitina Laundry.

1.2.4 Apa yang diberikan kerangka kerja, dan apa harganya

Tabel 1.1 mendaftar pekerjaan yang diambil alih oleh ketiga kerangka kerja di buku ini, beserta bab tempat Anda menjumpai masing-masing dalam praktik. Tidak ada yang ajaib di sini: setiap baris adalah kode biasa yang pernah ditulis sekali, dengan sangat cermat, dan kini dipakai bersama oleh jutaan aplikasi.

Tabel 1.1 Pekerjaan yang diambil alih kerangka kerja web, dan bab-bab di buku ini tempat masing-masing dibangun untuk Profitina Laundry.

Pekerjaan	Yang dikerjakan kerangka kerja untuk Anda	Di buku ini
Melayani HTTP	Server web yang teruji dan cepat, menangani banyak permintaan sekaligus	Bab 1, 3
Perutean	Memetakan metode dan pola URL ke satu aksi, mengekstrak parameter	Bab 2, 3, 7, 11
Model binding	Mengubah *query string*, isian formulir, dan JSON menjadi objek bertipe	Bab 3, 9, 13
Templat	*Layout*, *partial view*, dan *escaping* HTML otomatis	Bab 2, 3, 7, 13
Akses data	ORM yang memetakan tabel ke kelas dan menulis SQL-nya	Bab 5, 7, 11
Migrasi	Perubahan skema basis data yang berversi dan dapat diputar ulang di mana saja	Bab 5, 9, 11
Validasi	Aturan deklaratif pada model, pesan galat di samping isian	Bab 6, 9, 13
Keamanan	Login, aturan akses, *hashing* kata sandi, token CSRF	Bab 6, 10, 14
Konfigurasi	Pengaturan per lingkungan dari berkas dan variabel lingkungan	Bab 3, 4, 6
Pengujian	*Test host* dan alat bantu yang menjalankan permintaan tanpa peramban	Bab 10, 14

Harganya juga nyata, dan lebih baik disebutkan sejak awal. Kerangka kerja harus *dipelajari*: konvensinya, kosakatanya, cara kerjanya. Ia membuat beberapa hal begitu mudah sehingga Anda berhenti memahaminya; itulah sebabnya bab ini dan bab berikutnya membangun mesinnya secara manual sekali. Ia membawa *dependensi*, yaitu paket yang harus diunduh, dijaga tetap mutakhir, dan dipercaya. Dan ia terus bergerak: ASP.NET Core merilis versi mayor baru setiap November, Laravel setiap musim semi (di belahan bumi utara), Spring Boot dua kali setahun, dan setiap versi baru meminta sedikit pekerjaan. Subbab 1.3 menyebutkan versi yang berlaku saat ini dan berapa lama masing-masing didukung.

1.3 Mengapa ASP.NET Core, Laravel, dan Spring Boot

Ada puluhan kerangka kerja web. Buku ini mengajarkan tiga, masing-masing dalam bahasanya sendiri, karena bersama-sama ketiganya mencakup sebagian besar pekerjaan sisi server yang akan dijumpai seorang lulusan, dan karena melihat gagasan yang sama tiga kali adalah cara tercepat untuk membedakan gagasan dari sintaksisnya.

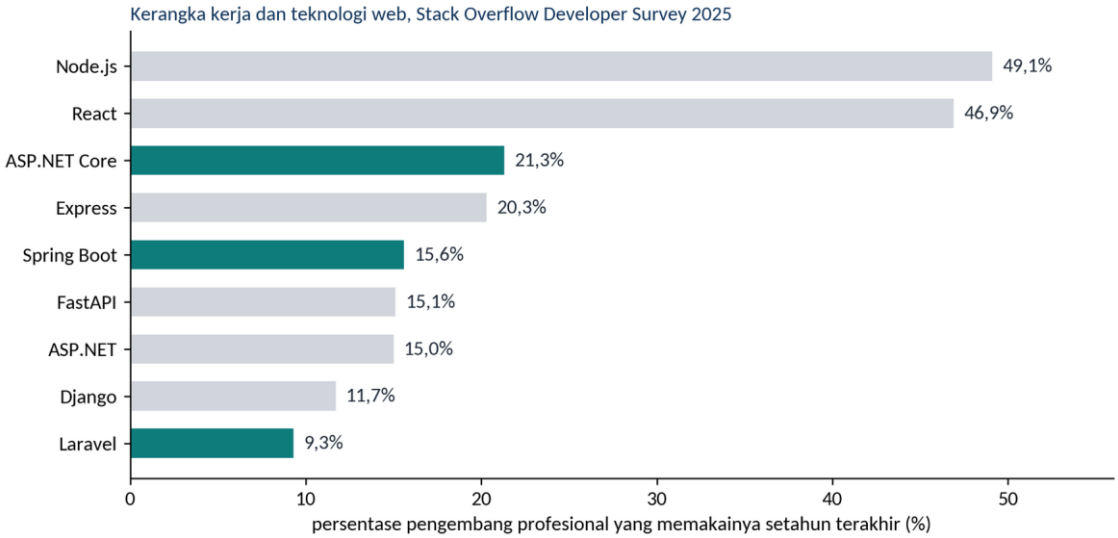
ASP.NET Core adalah kerangka kerja web sumber terbuka dan lintas platform dari Microsoft untuk C#. Ia berjalan di Windows, macOS, dan Linux, membawa server webnya sendiri (Kestrel), dan merupakan cara yang dianjurkan untuk membangun aplikasi web baru apa pun di .NET. Rilis dukungan jangka panjangnya (*long-term support*, LTS) adalah **.NET 10**, dirilis pada November 2025 dan didukung selama tiga tahun; pembaruan 8 September 2026 membawa *runtime*-nya ke versi 10.0.12 [5].

Laravel adalah kerangka kerja yang paling banyak dipakai untuk PHP, bahasa yang masih berada di balik sebagian besar web. Laravel 13 dirilis pada 17 Maret 2026; ia memerlukan PHP 8.3 atau yang lebih baru dan mendukung PHP 8.3, 8.4, dan 8.5, dengan perbaikan bug hingga kuartal ketiga 2027 dan perbaikan keamanan hingga Maret 2028 [6]. PHP 8.5 sendiri dirilis pada 20

November 2025; pembaruan terakhirnya saat buku ini ditulis adalah 8.5.11 tanggal 24 September 2026 [7].

Spring Boot adalah cara baku membangun aplikasi server di Java. Ia berdiri di atas Spring Framework dan menambahkan konfigurasi otomatis, server tertanam, dan "starter" siap pakai. Lini yang berlaku saat ini adalah Spring Boot 4.1 (4.1.1 tanggal 20 Agustus 2026), yang berjalan di Java 17 hingga 26; lini 4.0 didukung sampai akhir 2026 [8]. Buku ini memakai Java 25, rilis dukungan jangka panjang September 2025, yang pembaruan terakhirnya adalah 25.0.4.1 tanggal 18 Agustus 2026 [9].

Seberapa banyak ketiganya dipakai? Survei terbuka terbesar di kalangan pengembang, Stack Overflow Developer Survey, pada 2025 menanyakan kerangka kerja dan teknologi web mana yang dipakai secara intensif setahun terakhir. **Gambar 1.3** menampilkan jawaban 19.460 pengembang profesional yang menjawab pertanyaan itu [10]. ASP.NET Core dipakai oleh 21,3%, Spring Boot oleh 15,6%, dan Laravel oleh 9,3%; ASP.NET versi lama (pendahulu ASP.NET Core di .NET Framework) masih dipakai oleh 15,0%. Node.js dan React berada di puncak karena dipakai di sisi *front end* oleh hampir semua orang, termasuk banyak yang juga memakai salah satu dari tiga kerangka kerja kita di sisi server.



Gambar 1.3 Kerangka kerja dan teknologi web yang dipakai pengembang profesional dalam Stack Overflow Developer Survey 2025 (19.460 responden untuk pertanyaan ini); tiga kerangka kerja buku ini berwarna hijau toska.

Perlakukan angka semacam ini sebagai potret sesaat, bukan peringkat untuk dihafal: survei mengukur siapa yang menjawab, bukan seluruh dunia, dan hasilnya berubah setiap tahun. Yang penting bagi pemula adalah bahwa ketiganya matang, dikembangkan secara aktif, terdokumentasi dengan baik, dicari oleh pemberi kerja, dan masing-masing mewakili satu dari tiga keluarga besar pengembangan sisi server: .NET, PHP, dan Java Virtual Machine.

Tabel 1.2 membandingkan ketiganya secara sekilas. Setiap barisnya akan menjadi akrab pada bab-bab mendatang.

Tabel 1.2 Tiga kerangka kerja buku ini berdampingan, dengan versi yang berlaku pada September 2026.

	ASP.NET Core	Laravel	Spring Boot
Bahasa	C# 14	PHP 8.5	Java 25
Versi saat ini (Sept. 2026)	10.0 LTS (*runtime* 10.0.12)	13.x	4.1.1
Pembuat	Microsoft, sumber terbuka	Taylor Otwell, sumber terbuka	VMware, kini Broadcom, sumber terbuka
Perkakas proyek	CLI `dotnet`	Composer, `laravel`, `php artisan`	Maven atau Gradle, Spring Initializr
Sumber paket	NuGet	Packagist	Maven Central
Server web	Kestrel, bawaan	Server PHP, atau Nginx/Apache dengan PHP-FPM	Tomcat, tertanam
Perutean	Atribut atau pola terpusat	Berkas terpusat `routes/web.php`	Anotasi pada metode
Templat	Razor (`.cshtml`)	Blade (`.blade.php`)	Thymeleaf (`.html`)
ORM	Entity Framework Core	Eloquent	Spring Data JPA (Hibernate)
Bab	3–6	7–10	11–14

1.4 Menyiapkan Meja Kerja

Subbab ini memasang semua yang diperlukan buku ini. Lakukan sekali, dengan tenang, dan periksa setiap perkakas sebelum melanjutkan; kebanyakan masalah pada minggu-minggu pertama bersama kerangka kerja sebenarnya adalah masalah pemasangan yang menyamar. Versi yang disebutkan adalah versi yang berlaku pada September 2026; jika sudah ada *patch* yang lebih baru ketika Anda membaca ini, pakailah yang lebih baru.

1.4.1 Visual Studio Code

Ketiga kerangka kerja dipakai dengan penyunting yang sama di buku ini: **Visual Studio Code** (VS Code), gratis dan tersedia untuk Windows, macOS, dan Linux di https://code.visualstudio.com. Di Windows jalankan pemasangnya dan centang “Add to PATH”; di macOS seret aplikasinya ke *Applications*, buka, tekan **Cmd+Shift+P**, dan jalankan *Shell Command: Install ‘code’ command in PATH*; di Ubuntu atau Debian pasang paket `.deb`, dan di Fedora paket `.rpm`. Setelah itu perintah `code .` membuka folder yang sedang aktif.

VS Code mengenal setiap bahasa melalui *extension* (ekstensi). Buka tampilan Extensions (**Ctrl+Shift+X**, di macOS **Cmd+Shift+X**) dan pasang ekstensi pada **Tabel 1.3**, atau ketik perintah `code --install-extension` di terminal.

Tabel 1.3 Ekstensi VS Code yang dipakai di buku ini. Jika komputer Anda lambat, pasang hanya ekstensi untuk kerangka kerja yang sedang Anda pelajari.

Ekstensi (pengenal)	Untuk	Yang ditambahkannya
C# Dev Kit (`ms-dotnettools.csdevkit`)	ASP.NET Core	Dukungan bahasa C#, Solution Explorer, menjalankan dan *debug* [11]
PHP Intelephense (`bmewburn.vscode-intelephense-client`)	Laravel	Pelengkapan PHP, lompat ke definisi, galat saat mengetik
Laravel (`laravel.vscode-laravel`)	Laravel	Dukungan Blade, pelengkapan rute, view, dan konfigurasi [12]
Extension Pack for Java (`vscjava.vscode-java-pack`)	Spring Boot	Dukungan bahasa Java, Maven, menjalankan dan *debug* [13]
Spring Boot Extension Pack (`vmware.vscode-boot-dev-pack`)	Spring Boot	Spring Initializr di VS Code, dasbor Spring [13]
SQLite Viewer (`qwtel.sqlite-viewer`)	semua	Membuka berkas `.db` sebagai tabel

1.4.2 .NET 10 SDK

SDK (*software development kit*) berisi kompilator C#, perintah `dotnet`, templat proyek, dan kedua *runtime*: *runtime* .NET dan *runtime* ASP.NET Core, yang terakhir ini disebut juga *shared framework* (kerangka kerja bersama). Pasanglah SDK, bukan hanya *runtime*. **Tabel 1.4** memuat satu baris untuk setiap sistem; halaman unduhan resmi mendaftar semua pemasangnya [14].

Tabel 1.4 Memasang ketiga rangkaian perkakas. Buka terminal baru setiap selesai memasang, agar perintah baru dikenali.

Sistem	.NET 10 SDK	PHP 8.5 + Composer	JDK 25 + Maven
Windows 10/11	`winget install Microsoft.DotNet.SDK.10`	pemasang php.new di PowerShell (di bawah)	`winget install EclipseAdoptium.Temurin.25.JDK`, zip Maven dari maven.apache.org
macOS	`.pkg` dari dotnet.microsoft.com, atau `brew install --cask dotnet-sdk`	pemasang php.new (di bawah), atau `brew install php composer`	`brew install openjdk@25 maven`
Ubuntu 24.04 dan sesudahnya	`sudo apt install dotnet-sdk-10.0`	pemasang php.new (di bawah)	`sudo apt install openjdk-25-jdk maven`
Fedora	`sudo dnf install dotnet-sdk-10.0`	pemasang php.new (di bawah)	`sudo dnf install java-25-openjdk-devel maven`

Jika rilis Linux Anda belum menyediakan paket `openjdk-25-jdk`, pasanglah Eclipse Temurin 25 dari https://adoptium.net, yang menerbitkan paket untuk Debian, Ubuntu, Fedora, dan lainnya; jika belum menyediakan `dotnet-sdk-10.0`, skrip `dotnet-install.sh` dari Microsoft di halaman unduhan memasang SDK ke folder rumah Anda [14].

1.4.3 PHP 8.5 dan Composer

Laravel memerlukan PHP, pengelola paketnya **Composer**, dan, agar praktis, **Laravel installer**. Dokumentasi Laravel 13 menganjurkan satu perintah untuk setiap sistem yang memasang ketiganya dari https://php.new [15]:

```
# macOS
/bin/bash -c "$(curl -fsSL https://php.new/install/mac/8.5)"
# Linux
/bin/bash -c "$(curl -fsSL https://php.new/install/linux/8.5)"
# Windows PowerShell, run as administrator
Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://php.new/install/windows/8.5'))
```

Jika PHP dan Composer sudah terpasang (misalnya dari pelajaran pemrograman web Anda), cukup tambahkan pemasangnya dengan `composer global require laravel/installer` [15]. Composer juga dapat dipasang tersendiri dari https://getcomposer.org [16]. Di Windows, aplikasi desktop gratis dari tim Laravel, *Herd*, memasang PHP, Composer, pemasang Laravel, dan server web lokal dalam satu langkah; pilihan ini baik jika baris perintah masih terasa asing.

1.4.4 JDK 25 dan Maven

Spring Boot memerlukan Java Development Kit dan sebuah perkakas *build*. Buku ini memakai **JDK 25** (distribusi apa saja: Eclipse Temurin, Oracle, Microsoft, Amazon Corretto; semuanya dibangun dari kode sumber OpenJDK yang sama) dan **Apache Maven** 3.9, yang rilis terbarunya adalah 3.9.16 [17]. Dalam praktik Maven bersifat opsional, karena setiap proyek yang dibuat oleh **Spring Initializr** (https://start.spring.io) memuat skrip kecil `mvnw` (`mvnw.cmd` di Windows) yang mengunduh Maven yang tepat dengan sendirinya [18]. Setelah memasang, pastikan `java -version` menyebut versi 25; jika yang menjawab Java yang lebih lama, atur `JAVA_HOME` ke folder JDK yang baru dan letakkan folder `bin`-nya paling depan pada `PATH`.

1.4.5 Basis data: SQLite dan MySQL

Dua basis data dipakai di buku ini. **SQLite** adalah basis data SQL utuh dalam satu berkas, tanpa server yang perlu dipasang; SQLite adalah basis data bawaan proyek Laravel 13 yang baru [15], sedangkan ASP.NET Core dan Spring Boot berbicara dengannya melalui paket *driver*. Lini rilis SQLite saat ini adalah 3.53, yang pertama dirilis pada 9 April 2026 [19]; Anda tidak perlu memasangnya tersendiri, karena *driver* setiap kerangka kerja sudah membawanya, tetapi perkakas baris perintah `sqlite3` (dari https://sqlite.org/download.html, atau `brew install sqlite`, `sudo apt install sqlite3`) dan ekstensi *SQLite Viewer* berguna untuk melihat isi basis data.

MySQL adalah basis data server yang ditawarkan kebanyakan penyedia *hosting*, dan menjadi tujuan *deployment* pada Bab 6, 10, dan 14. Rilis dukungan jangka panjangnya saat ini adalah **MySQL 9.7 LTS**, lini LTS pertama sejak 8.4 [20]; server komunitasnya tersedia di https://dev.mysql.com/downloads/, dan di Linux `sudo apt install mysql-server` juga dapat dipakai. MariaDB, turunan yang kompatibel, sama baiknya untuk semua yang ada di buku ini.

1.4.6 Memeriksa semuanya

Buka terminal **baru** dan tanyakan versi setiap perkakas.

```
code --version
dotnet --list-sdks
dotnet --list-runtimes
php --version
composer --version
java -version
mvn -version
```

Pada mesin yang membangun buku ini, jawaban .NET, PHP, Composer, Java, dan Maven adalah sebagai berikut (baris JAVA_TOOL_OPTIONS khas mesin itu dihilangkan).

```
10.0.112 [/usr/lib/dotnet/sdk]
Microsoft.AspNetCore.App 10.0.12 [/usr/lib/dotnet/shared/Microsoft.AspNetCore.App]
Microsoft.NETCore.App 10.0.12 [/usr/lib/dotnet/shared/Microsoft.NETCore.App]
PHP 8.4.21 (cli) (built: May  7 2026 18:44:14) (NTS)
Composer version 2.8.12 2025-09-19 13:41:59
openjdk version "21.0.10" 2026-01-20
Apache Maven 3.9.11 (3e54c93a704957b63ee3494413a2b544fd3d825b)
```

Dua hal patut dibaca dari jawaban ini. Pertama, `Microsoft.AspNetCore.App 10.0.12` adalah **shared framework** ASP.NET Core: seluruh ASP.NET Core (server Kestrel, perutean, MVC, Razor, JSON) terpasang bersama SDK, sehingga proyek web tidak memerlukan apa pun dari internet. Kedua, mesin *build* itu menjalankan PHP 8.4 dan Java 21, satu versi di bawah yang dipasang di bab ini. Semua yang ada di Bab 1 sampai 4 hanya memakai fitur yang sama-sama dimiliki versi-versi itu dan PHP 8.5 serta Java 25, sehingga keluaran yang akan Anda lihat sama. Mesin *build* itu juga punya satu keterbatasan penting, yang dinyatakan terus terang oleh buku ini di setiap tempat yang relevan: mesin itu **tidak dapat mengunduh paket** dari NuGet, Packagist, atau Maven Central. Karena itu setiap program di Bab 1 sampai 4 ditulis agar hanya memerlukan apa yang sudah ada di dalam SDK, dan setiap program itu dikompilasi dan dijalankan; keluaran yang dicetak di bab-bab ini adalah keluaran sesungguhnya. Laravel dan Spring Boot sendiri berupa paket, sehingga proyek pertama keduanya (Subbab 1.6.3) tidak dapat dibuat di mesin itu; perintah-perintahnya diberikan, dan Bab 7 serta Bab 11 membangunnya secara utuh.

1.5 Membangunnya Tanpa Kerangka Kerja

Sebelum membiarkan kerangka kerja bekerja, kita mengerjakannya sendiri sekali, agar tahu persis apa yang kita serahkan. Tugasnya adalah daftar harga Bu Wati dalam bentuk terkecil, dengan empat alamat:

- `/` menjawab dengan halaman beranda singkat yang menautkan ke daftar harga;
- `/services` menjawab dengan tabel kedua belas layanan beserta harganya;
- `/services/S0004` menjawab dengan satu layanan, atau dengan `404 Not Found` jika kodenya tidak ada;
- alamat lain mana pun menjawab `404`, dan setiap metode selain `GET` menjawab `405 Method Not Allowed`.

Datanya berasal dari `data/Service.csv`, hasil ekspor tabel `Service` dari basis data Profitina Laundry. Baris-baris pertamanya terlihat di bawah; perhatikan tanda kutip di sekitar deskripsi yang mengandung koma.

```
id,description,price
S0001,"Wash, dry & fold (per kg)",8000.00
S0002,"Express wash & fold, 1 day (per kg)",12000.00
S0003,Ironing only (per kg),6000.00
S0004,Dry cleaning - shirt/blouse (per item),20000.00
```

1.5.1 Dalam C#, dengan `HttpListener`

.NET sudah lama memiliki kelas server HTTP kecil, `System.Net.HttpListener`, jauh sebelum ASP.NET Core ada [21]. Kelas ini bukan kerangka kerja: ia menerima koneksi dan menyerahkan permintaan mentah kepada kita, dan sisanya urusan kita. **Kode 1.1** adalah seluruh programnya.

Ini proyek *console* (`Sdk="Microsoft.NET.Sdk"`, bukan `.Web`), jadi ASP.NET Core sama sekali tidak terlibat.

Kode 1.1 `nofw-csharp/Program.cs`: daftar harga tanpa kerangka kerja. Komentar (1) sampai (6) menandai pekerjaan yang biasanya dilakukan kerangka kerja.

```
// Profitina Laundry's price list with NO framework: only HttpListener.
// Everything a framework would do is written out by hand, and marked.
using System.Globalization;
using System.Net;
using System.Text;

var services = LoadServices("../data/Service.csv");
var listener = new HttpListener();
listener.Prefixes.Add("http://127.0.0.1:4010/");
listener.Start();
Console.WriteLine("Listening on http://127.0.0.1:4010/ (Ctrl+C stops)");

while (true)
{
    var context = listener.GetContext();         // wait for ONE request
    string method = context.Request.HttpMethod;
    string path = context.Request.Url!.AbsolutePath;
    int status = 200;
    string body;

    // (1) Routing by hand: compare the path with every page we know.
    if (method != "GET")
    {
        status = 405;
        body = Page("Not allowed", "<p>This site only answers GET.</p>");
    }
    else if (path == "/")
    {
        body = Page("Profitina Laundry",
            "<p>See <a href=\"/services\">our services and prices</a>.</p>");
    }
    else if (path == "/services")
    {
        var rows = new StringBuilder();
        foreach (var s in services)
        {
            rows.Append($"<tr><td><a href=\"/services/{s.Id}\">{s.Id}</a>"
                + $"</td><td>{Escape(s.Description)}</td>"
                + $"<td>{Rp(s.Price)}</td></tr>");
        }
        body = Page("Our services", $"<table>{rows}</table>");
    }
    else if (path.StartsWith("/services/") && path.Count(c => c == '/') == 2)
    {
        // (2) Parameter extraction by hand: cut the id out of the path.
        string id = Uri.UnescapeDataString(path["/services/".Length..]);
```

Kode 1.1 (Lanjutan) *nofw-csharp/Program.cs: daftar harga tanpa kerangka kerja. Komentar (1) sampai (6) menandai pekerjaan yang biasanya dilakukan kerangka kerja.*

```
        var found = services.FirstOrDefault(s => s.Id == id);
        if (found is null)
        {
            status = 404;
            body = Page("Not found", $"<p>No service {Escape(id)}.</p>");
        }
        else
        {
            body = Page(found.Id, $"<p>{Escape(found.Description)}: "
                + $"{Rp(found.Price)}</p>");
        }
    }
    else
    {
        status = 404;
        body = Page("Not found", $"<p>Nothing at {Escape(path)}.</p>");
    }

    // (3) The response by hand: status, header, bytes, close.
    byte[] bytes = Encoding.UTF8.GetBytes(body);
    context.Response.StatusCode = status;
    context.Response.ContentType = "text/html; charset=utf-8";
    context.Response.ContentLength64 = bytes.Length;
    context.Response.OutputStream.Write(bytes);
    context.Response.Close();
    Console.WriteLine($"{method} {path} -> {status}");
}

// (4) The page frame, repeated for every page, by hand.
static string Page(string title, string content) =>
    "<!DOCTYPE html><html><head><meta charset=\"utf-8\">"
    + $"<title>{Escape(title)}</title></head><body>"
    + $"<h1>{Escape(title)}</h1>{content}</body></html>";

// (5) HTML escaping by hand: forget it once and the site is attackable.
static string Escape(string text) => WebUtility.HtmlEncode(text);

static string Rp(decimal amount) =>
    "Rp " + amount.ToString("N0", CultureInfo.InvariantCulture)
        .Replace(",", ".");
```

Kode 1.1 (Lanjutan) *nofw-csharp/Program.cs: daftar harga tanpa kerangka kerja. Komentar (1) sampai (6) menandai pekerjaan yang biasanya dilakukan kerangka kerja.*

```
// (6) Reading the data by hand: a CSV line with "quoted, commas".
static List<Service> LoadServices(string file)
{
    var list = new List<Service>();
    foreach (var line in File.ReadLines(file).Skip(1))
    {
        var f = new List<string>();
        var cur = new StringBuilder();
        bool quoted = false;
        foreach (char c in line)
        {
            if (c == '"') quoted = !quoted;
            else if (c == ',' && !quoted) { f.Add(cur.ToString()); cur.Clear(); }
            else cur.Append(c);
        }
        f.Add(cur.ToString());
        list.Add(new Service(f[0], f[1],
            decimal.Parse(f[2], CultureInfo.InvariantCulture)));
    }
    return list;
}

record Service(string Id, string Description, decimal Price);
```

Bacalah listing itu sekali dari atas ke bawah dan hitung pekerjaannya. (1) Perutean berupa rantai `if`/`else if` yang membandingkan path dengan setiap halaman yang kita kenal. (2) Kode `S0004` dipotong dari path dengan aritmetika string, `path["/services/".Length..]`, setelah memeriksa bahwa path mengandung tepat dua garis miring. (3) Respons dirakit secara manual: status, jenis konten, panjang, byte, tutup. (4) Bingkai halaman, dari `<!DOCTYPE html>` sampai `</html>`, adalah fungsi yang dipanggil setiap halaman. (5) Setiap nilai dari luar melewati `Escape`, yang memanggil `WebUtility.HtmlEncode`; lupa sekali saja, di satu halaman, dan begitulah situs lama Bu Wati mencetak `<script>` kembali ke pengunjung. (6) Berkas CSV diurai karakter demi karakter, karena deskripsi bisa mengandung koma di dalam tanda kutip.

Ada juga pekerjaan yang sama sekali tidak dilakukan listing itu: gelung `while (true)` melayani **satu permintaan pada satu waktu**. Selama ia membangun satu halaman, setiap pengunjung lain menunggu. Untuk dua belas layanan Bu Wati hal itu tidak berbahaya; untuk situs sungguhan, berbahaya. Server milik kerangka kerja melayani ribuan permintaan sekaligus, dan bagian itu tidak akan kita tulis sendiri.

Untuk menjalankannya, buka terminal di `Chapter01/nofw-csharp` dan ketik:

```
dotnet run
Listening on http://127.0.0.1:4010/ (Ctrl+C stops)
```

Buka `http://127.0.0.1:4010/services` di peramban; hasilnya tabel polos tanpa gaya. Subbab 1.7 mengajukan pertanyaan yang sama kepada keempat program dan membandingkan jawabannya.

1.5.2 Dalam PHP, dengan server bawaan

PHP dibuat untuk web, sehingga versi PHP lebih pendek: PHP sendiri mengurai permintaan ke dalam larik `$_SERVER`, `$_GET`, dan `$_POST`, dan `echo` menulis ke respons. Meski begitu, setiap pekerjaan dari Subbab 1.2.1 tetap ada. **Kode 1.2** memakai server pengembangan bawaan PHP, yang mengirim *setiap* permintaan ke satu skrip yang disebut pada baris perintahnya; karena itu skrip tersebut sekaligus menjadi router.

Kode 1.2 `nofw-php/index.php`: daftar harga yang sama dalam PHP tanpa kerangka kerja. Skrip ini sekaligus router, controller, model, dan view.

```php
<?php
// Profitina Laundry's price list with NO framework: PHP's built-in server.
// Run from this folder:  php -S 127.0.0.1:4011 index.php
declare(strict_types=1);

function page(string $title, string $content): string
{
    $t = htmlspecialchars($title);
    return "<!DOCTYPE html><html><head><meta charset=\"utf-8\">"
        . "<title>$t</title></head><body><h1>$t</h1>$content</body></html>";
}

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

// Reading the data by hand (fgetcsv understands "quoted, commas").
$services = [];
$file = fopen(__DIR__ . '/../data/Service.csv', 'r');
fgetcsv($file, escape: '');                          // skip the header
while (($f = fgetcsv($file, escape: '')) !== false) {
    $services[$f[0]] = ['id' => $f[0], 'description' => $f[1],
                        'price' => (float) $f[2]];
}
fclose($file);

// Routing by hand.
$method = $_SERVER['REQUEST_METHOD'];
$path = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
$status = 200;
if ($method !== 'GET') {
    $status = 405;
    $body = page('Not allowed', '<p>This site only answers GET.</p>');
} elseif ($path === '/') {
    $body = page('Profitina Laundry',
        '<p>See <a href="/services">our services and prices</a>.</p>');
} elseif ($path === '/services') {
    $rows = '';
    foreach ($services as $s) {
        $rows .= "<tr><td><a href=\"/services/{$s['id']}\">{$s['id']}</a></td>"
            . '<td>' . htmlspecialchars($s['description']) . '</td>'
            . '<td>' . rp($s['price']) . '</td></tr>';
    }
    $body = page('Our services', "<table>$rows</table>");
} elseif (preg_match('#^/services/([^/]+)$#', $path, $m)) {
```

Kode 1.2 (Lanjutan) *nofw-php/index.php: daftar harga yang sama dalam PHP tanpa kerangka kerja. Skrip ini sekaligus router, controller, model, dan view.*

```
    $id = urldecode($m[1]);
    if (!isset($services[$id])) {
        $status = 404;
        $body = page('Not found', '<p>No service ' . htmlspecialchars($id)
            . '.</p>');
    } else {
        $s = $services[$id];
        $body = page($s['id'], '<p>' . htmlspecialchars($s['description'])
            . ': ' . rp($s['price']) . '</p>');
    }
} else {
    $status = 404;
    $body = page('Not found', '<p>Nothing at ' . htmlspecialchars($path)
        . '.</p>');
}

// The response by hand.
http_response_code($status);
header('Content-Type: text/html; charset=utf-8');
echo $body;
```

Dua baris patut diberi catatan. `parse_url(..., PHP_URL_PATH)` memotong *query string* dari URI yang diminta, karena `$_SERVER['REQUEST_URI']` memuat keduanya. Dan `preg_match('#^/services/([^/]+)$#', ...)` adalah perutean dengan ekspresi reguler: ia mencocokkan `/services/` diikuti satu atau lebih karakter yang bukan garis miring, lalu menangkapnya di `$m[1]`. Satu baris itu adalah router kecil; router milik kerangka kerja adalah gagasan yang sama, diperumum dan teruji. Jalankan dari `Chapter01/nofw-php`:

```
php -S 127.0.0.1:4011 index.php
```

1.5.3 Dalam Java, dengan `com.sun.net.httpserver`

JDK telah menyertakan server HTTP kecil sejak Java 6 dalam paket `com.sun.net.httpserver`. Paket ini bagian resmi dari modul `jdk.httpserver`, dan, seperti `HttpListener`, merupakan pustaka, bukan kerangka kerja. **Kode 1.3** mendaftarkan satu *handler* untuk semua path dan melakukan perutean di dalamnya.

Kode 1.3 `nofw-java/RawLaundry.java`: daftar harga dalam Java tanpa kerangka kerja, di atas server HTTP bawaan JDK.

```java
// Profitina Laundry's price list with NO framework: the JDK's HttpServer.
// Run from this folder:  java RawLaundry.java
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.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;

public class RawLaundry {
    record Service(String id, String description, long price) {}

    static List<Service> services = new ArrayList<>();

    public static void main(String[] args) throws IOException {
        List<String> lines = Files.readAllLines(Path.of("../data/Service.csv"));
        for (String line : lines.subList(1, lines.size())) {   // skip header
            List<String> f = splitCsv(line);
            services.add(new Service(f.get(0), f.get(1),
                Math.round(Double.parseDouble(f.get(2)))));
        }
        HttpServer server = HttpServer.create(
            new InetSocketAddress("127.0.0.1", 4012), 0);
        server.createContext("/", RawLaundry::handle);  // one handler for all
        server.start();
        System.out.println("Listening on http://127.0.0.1:4012/ (Ctrl+C stops)");
    }

    static void handle(HttpExchange ex) throws IOException {
        String path = ex.getRequestURI().getPath();
        int status = 200;
        String body;
        if (!ex.getRequestMethod().equals("GET")) {             // routing by hand
            status = 405;
            body = page("Not allowed", "<p>This site only answers GET.</p>");
        } else if (path.equals("/")) {
            body = page("Profitina Laundry",
                "<p>See <a href=\"/services\">our services and prices</a>.</p>");
        } else if (path.equals("/services")) {
            StringBuilder rows = new StringBuilder();
            for (Service s : services) {
```

Kode 1.3 (Lanjutan) *nofw-java/RawLaundry.java: daftar harga dalam Java tanpa kerangka kerja, di atas server HTTP bawaan JDK.*

```
                rows.append("<tr><td><a href=\"/services/" + s.id() + "\">")
                    .append(s.id()).append("</a></td><td>")
                    .append(escape(s.description())).append("</td><td>")
                    .append(rp(s.price())).append("</td></tr>");
            }
            body = page("Our services", "<table>" + rows + "</table>");
        } else if (path.matches("/services/[^/]+")) {
            String id = URLDecoder.decode(path.substring(10),
                StandardCharsets.UTF_8);
            Service found = services.stream().filter(s -> s.id().equals(id))
                .findFirst().orElse(null);
            if (found == null) {
                status = 404;
                body = page("Not found", "<p>No service " + escape(id) + ".</p>");
            } else {
                body = page(found.id(), "<p>" + escape(found.description())
                    + ": " + rp(found.price()) + "</p>");
            }
        } else {
            status = 404;
            body = page("Not found", "<p>Nothing at " + escape(path) + ".</p>");
        }
        byte[] bytes = body.getBytes(StandardCharsets.UTF_8);  // response by hand
        ex.getResponseHeaders().set("Content-Type", "text/html; charset=utf-8");
        ex.sendResponseHeaders(status, bytes.length);
        try (OutputStream out = ex.getResponseBody()) {
            out.write(bytes);
        }
        System.out.println(ex.getRequestMethod() + " " + path + " -> " + status);
    }

    static String page(String title, String content) {
        return "<!DOCTYPE html><html><head><meta charset=\"utf-8\"><title>"
            + escape(title) + "</title></head><body><h1>" + escape(title)
            + "</h1>" + content + "</body></html>";
    }

    static String escape(String s) {
        return s.replace("&", "&amp;").replace("<", "&lt;")
            .replace(">", "&gt;").replace("\"", "&quot;");
    }

    static String rp(long amount) {
        return "Rp " + String.format("%,d", amount).replace(',', '.');
    }
```

Kode 1.3 (Lanjutan) *nofw-java/RawLaundry.java: daftar harga dalam Java tanpa kerangka kerja, di atas server HTTP bawaan JDK.*

```
    static List<String> splitCsv(String line) {       // "quoted, commas" too
        List<String> fields = new ArrayList<>();
        StringBuilder cur = new StringBuilder();
        boolean quoted = false;
        for (char c : line.toCharArray()) {
            if (c == '"') quoted = !quoted;
            else if (c == ',' && !quoted) {
                fields.add(cur.toString());
                cur.setLength(0);
            }
            else cur.append(c);
        }
        fields.add(cur.toString());
        return fields;
    }
}
```

Sejak Java 11, satu berkas sumber dapat dijalankan tanpa dikompilasi lebih dulu, sehingga dari `Chapter01/nofw-java` perintahnya cukup:

```
java RawLaundry.java
```

Berbeda dengan `HttpListener`, server JDK dapat memakai kumpulan *thread* (`server.setExecutor(...)`); kita membiarkan bawaannya, yang lagi-lagi melayani satu permintaan pada satu waktu. *Escaping* HTML di `escape` ditulis manual dengan empat panggilan `replace`; pustaka standar Java tidak punya pengode HTML, dan itu satu lagi pekerjaan yang diambil alih mesin templat kerangka kerja.

1.6 Daftar Harga yang Sama dengan Kerangka Kerja

1.6.1 *Minimal API* ASP.NET Core

Sekarang keempat alamat yang sama dengan ASP.NET Core. Berkas proyeknya berbeda satu kata saja: `Sdk="Microsoft.NET.Sdk.Web"` alih-alih `Sdk="Microsoft.NET.Sdk"`. Kata itu membawa masuk *shared framework* ASP.NET Core. **Kode 1.4** adalah seluruh berkasnya.

Kode 1.4 `aspnet/HelloLaundry.csproj`. SDK `.Web` menjadikan ini proyek ASP.NET Core; `net10.0` menargetkan .NET 10.

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

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

</Project>
```

ASP.NET Core menawarkan dua gaya untuk menangani permintaan: **controller** dengan view (MVC, mulai Bab 3), dan **minimal API**, tempat setiap rute berupa satu baris yang memetakan metode dan pola URL ke sebuah fungsi [22]. *Minimal API* adalah cara terpendek untuk memperlihatkan apa yang dikerjakan kerangka kerja, maka **Kode 1.5** memakainya.

Kode 1.5 `aspnet/Program.cs`: daftar harga dengan *minimal API* ASP.NET Core. Perutean, ekstraksi parameter, kode status, JSON, dan server adalah urusan kerangka kerja.

```
// The same price list WITH a framework: ASP.NET Core minimal APIs.
using System.Globalization;
using System.Text.RegularExpressions;

var app = WebApplication.CreateBuilder(args).Build();

// A CSV field is either "quoted, maybe with commas" or plain text.
var field = new Regex("\"([^\"]*)\"|([^,]+)");
var services = File.ReadLines("../data/Service.csv").Skip(1)
    .Select(line => field.Matches(line)
        .Select(m => m.Groups[1].Success ? m.Groups[1].Value : m.Value)
        .ToArray())
    .Select(f => new Service(f[0], f[1],
        decimal.Parse(f[2], CultureInfo.InvariantCulture)))
    .ToList();

app.MapGet("/", () => "Profitina Laundry. Prices at /services");
app.MapGet("/services", () => services);
app.MapGet("/services/{id}", (string id) =>
    services.FirstOrDefault(s => s.Id == id) is { } found
        ? Results.Ok(found)
        : Results.NotFound(new { error = $"No service {id}" }));

app.Run();

record Service(string Id, string Description, decimal Price);
```

Bandingkan dengan **Kode 1.1**, baris demi baris.

- `WebApplication.CreateBuilder(args).Build()` membuat aplikasi dengan server Kestrel, pencatatan (*logging*), dan konfigurasi yang sudah siap. Tidak ada `HttpListener`, tidak ada prefiks, tidak ada gelung: `app.Run()` menjalankan gelung milik kerangka kerja, yang melayani banyak permintaan sekaligus.
- `app.MapGet("/services/{id}", (string id) => ...)` adalah perutean *sekaligus* ekstraksi parameter dalam satu baris. `{id}` adalah **parameter rute**: router mencocokkan satu segmen path apa pun di posisi itu dan menyerahkan nilainya ke parameter fungsi yang bernama sama. Tidak ada yang dipotong manual, dan permintaan untuk `/services/S0004/extra` tidak cocok dengan rute mana pun.
- `app.MapGet("/services", () => services)` mengembalikan sebuah daftar. Kerangka kerja melihat bahwa hasilnya bukan string lalu menserialisasikannya menjadi **JSON**, dengan header `Content-Type: application/json; charset=utf-8`.
- `Results.Ok(found)` dan `Results.NotFound(...)` memilih kode status berdasarkan nama, bukan angka.
- Tidak ada halaman 404 untuk path yang tak dikenal dan tidak ada 405 untuk `POST`: router menjawab keduanya sendiri.
- Penguraian CSV masih urusan kita (kerangka kerja tidak mengenal berkas kita), tetapi menyusut menjadi satu ekspresi reguler, `"([^"]*)"|([^,]+)`, yang berarti "entah isian bertanda kutip, entah deretan karakter tanpa koma". Bab 5 mengganti berkas itu dengan basis data dan ORM.

Selain bagian CSV, program ini berjumlah **26 baris**, dibandingkan **110** untuk **Kode 1.1**, **66** untuk versi PHP, dan **108** untuk versi Java. Yang lebih penting daripada jumlahnya adalah baris *mana* yang hilang: tepat baris-baris kuning pada **Gambar 1.1**, tempat bug bersembunyi.

Satu hal yang *tidak* dilakukan versi *minimal API* adalah menghasilkan HTML. Ia menjawab dengan JSON, yang diinginkan aplikasi seluler atau program lain, dan yang paling cocok dikerjakan

minimal API. Mengubah data menjadi halaman HTML dengan *layout*, tautan, dan formulir adalah tugas MVC dan Razor, dan Bab 3 membangun situs lengkap Bu Wati dengan cara itu.

1.6.2 Menjalankannya

```
cd Chapter01/aspnet
dotnet run --urls http://127.0.0.1:4013
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://127.0.0.1:4013
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /home/claude/kit/FBP/Code_new/Chapter01/aspnet
```

Empat fakta terbaca dari log ini. Aplikasi ini **adalah** server webnya: Kestrel ada di dalamnya. `--urls` memilih alamatnya; tanpa itu, ASP.NET Core memakai `http://localhost:5000` atau alamat di `Properties/launchSettings.json`. **Hosting environment** (lingkungan hosting) adalah `Production`, karena tidak ada yang menyatakan lain; Bab 3 memperlihatkan bagaimana `Development` mengubah halaman galat. Dan **content root** adalah folder tempat program membaca berkas-berkasnya, itulah sebabnya path relatif `../data/Service.csv` berfungsi ketika Anda menjalankan program dari foldernya sendiri. Berkas kecil `appsettings.json` di samping program mengatur tingkat log pesan-pesan ASP.NET Core sendiri ke `Warning`, sehingga hanya pesan kita dan baris-baris awal yang muncul.

Tanyakan satu layanan kepadanya dengan `curl`, klien HTTP baris perintah yang tersedia di Windows 10 dan 11, macOS, dan hampir setiap Linux (`-i` menampilkan baris status dan header):

```
curl -i http://127.0.0.1:4013/services/S0004
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Server: Kestrel
Transfer-Encoding: chunked

{"id":"S0004","description":"Dry cleaning - shirt/blouse (per item)","price":20000.00}
```

(Header `Date`, yang hanya menyebut waktu permintaan, dihilangkan di sini dan di seluruh buku.) Nama-nama properti diubah menjadi *camelCase* (`Id` menjadi `id`), konvensi JSON di web; harga bertipe `decimal` mempertahankan dua desimalnya. Tak satu pun dari ini kita tulis.

1.6.3 Proyek sungguhan pertama di setiap kerangka kerja

Keempat program di atas sengaja dibuat mungil. Proyek kerangka kerja sungguhan dimulai dari **templat** atau **generator** yang membuat folder, konfigurasi, dan halaman beranda yang berfungsi. Inilah perintah-perintahnya; jalankan di folder tempat Anda menyimpan proyek.

ASP.NET Core MVC, dengan templat yang menjadi titik awal Bab 3:

```
dotnet new mvc -o FirstMvc
cd FirstMvc
dotnet run
```

```
The template "ASP.NET Core Web App (Model-View-Controller)" was created successfully.
This template contains technologies from parties other than Microsoft, see
https://aka.ms/aspnetcore/10.0-third-party-notices for details.

Processing post-creation actions...
Restoring …/FirstMvc/FirstMvc.csproj:
  Determining projects to restore...
  Restored …/FirstMvc/FirstMvc.csproj (in 135 ms).
Restore succeeded.
```

Perintah ini sungguh dijalankan di mesin *build*: templat itu tidak memerlukan paket di luar *shared framework*, sehingga *restore* berhasil tanpa internet. `dotnet run` lalu menjalankan situsnya; halaman berandanya berjudul *Home Page - FirstMvc*. Bab 3 membongkar proyek ini berkas demi berkas.

Laravel 13, dengan pemasang dari Subbab 1.4.3 [15]:

```
laravel new first-laravel
cd first-laravel
php artisan serve
```

Pemasang itu mengajukan beberapa pertanyaan (*starter kit*, kerangka pengujian; untuk sekarang pilih *None* dan *Pest* atau *PHPUnit*), mengunduh Laravel beserta paket-paketnya dari Packagist, membuat basis data SQLite `database/database.sqlite`, dan menjalankan migrasi pertama. `php artisan serve` menjalankan server pengembangan PHP dengan Laravel di belakangnya, secara bawaan di `http://localhost:8000`. Langkah ini memerlukan internet, sehingga tidak dapat dijalankan di mesin *build*; Bab 7 membangun Profitina Laundry di Laravel langkah demi langkah.

Spring Boot 4, dengan Spring Initializr [18]. Buka https://start.spring.io, pilih *Maven*, *Java*, versi Spring Boot 4.1 yang berlaku, Java 25, nama artefak `first-spring`, dan dependensi *Spring Web* serta *Thymeleaf*, lalu tekan *Generate*; atau minta zip yang sama dari terminal:

```
curl -o first-spring.zip "https://start.spring.io/starter.zip?type=maven-
project&language=java&javaVersion=25&artifactId=first-spring&dependencies=web,thymeleaf"
```

Ekstrak zip itu, buka terminal di foldernya, dan jalankan dengan *Maven wrapper*: `./mvnw spring-boot:run` di macOS dan Linux, `mvnw.cmd spring-boot:run` di Windows. Jalankan pertama mengunduh Spring dan dependensinya dari Maven Central, lalu menyajikan aplikasi di `http://localhost:8080`. Langkah ini juga memerlukan internet, dan Bab 11 membangunnya secara utuh. Di VS Code, Spring Boot Extension Pack menawarkan generator yang sama lewat perintah *Spring Initializr: Create a Maven Project*.

1.7 Menjalankan Keempat Program Berdampingan

Perbandingan hanya adil jika setiap program diberi pertanyaan yang persis sama. Skrip `Chapter01/verify.sh` menjalankan keempat server bergantian, mengirim lima permintaan kepada masing-masing dengan `curl`, mencetak kode status, path, dan awal jawabannya (dengan tag HTML dibuang agar muat satu baris), lalu menghentikannya lagi. Kelima permintaan itu adalah halaman beranda, layanan yang ada, layanan yang tidak ada, path yang mengandung HTML (`%3Cb%3EHi%3C%2Fb%3E` adalah `<b>Hi</b>` yang ditulis dengan *percent-encoding*, cara peramban mengirim karakter semacam itu), dan sebuah `POST`. Inilah keluaran sesungguhnya.

```
===== C#, no framework (HttpListener), port 4010
200  /                              Profitina Laundry: See our services and prices
200  /services/S0004                S0004: Dry cleaning - shirt/blouse (per item):
404  /services/S0099                Not found: No service S0099.
404  /services/%3Cb%3EHi%3C%2Fb%3E  Not found: No service &lt;b&gt;Hi&lt;/b&gt;.
411  POST /services
===== PHP, no framework (built-in server), port 4011
200  /                              Profitina Laundry: See our services and prices
200  /services/S0004                S0004: Dry cleaning - shirt/blouse (per item):
404  /services/S0099                Not found: No service S0099.
404  /services/%3Cb%3EHi%3C%2Fb%3E  Not found: No service &lt;b&gt;Hi&lt;/b&gt;.
405  POST /services
===== Java, no framework (com.sun.net.httpserver), port 4012
200  /                              Profitina Laundry: See our services and prices
200  /services/S0004                S0004: Dry cleaning - shirt/blouse (per item):
404  /services/S0099                Not found: No service S0099.
404  /services/%3Cb%3EHi%3C%2Fb%3E  Not found: Nothing at /services/&lt;b&gt;Hi&lt
405  POST /services
===== C# with ASP.NET Core (minimal APIs), port 4013
200  /                              Profitina Laundry. Prices at /services
200  /services/S0004                {"id":"S0004","description":"Dry cleaning - sh
404  /services/S0099                {"error":"No service S0099"}
404  /services/%3Cb%3EHi%3C%2Fb%3E  {"error":"No service <b>Hi<%2Fb>"}
405  POST /services
```

Pada pertanyaan yang mudah keempatnya sepakat, dan memang itu yang kita inginkan: halaman yang sama untuk `/`, layanan yang sama untuk `S0004`, `404` untuk `S0099`. Baris yang menarik adalah baris tempat mereka *berbeda*, karena setiap perbedaan adalah keputusan yang harus diambil seseorang, dan yang diambilkan oleh kerangka kerja untuk Anda, secara konsisten, di mana pun.

`POST` tanpa badan. Program C# kita tidak pernah melihat permintaan itu. `curl -X POST` tidak mengirim badan maupun header `Content-Length`, dan `HttpListener` sendiri menolaknya dengan `411 Length Required` sebelum `if (method != "GET")` milik kita sempat menjawab `405`. Log server menegaskannya: program C# mencetak satu baris untuk setiap `GET` dan tidak satu pun untuk `POST`. Bahkan pustaka server yang paling tipis pun mengambil keputusan sendiri, dan ketika Anda menulis semuanya secara manual, Anda harus tahu semuanya.

Garis miring yang di-*encode*. Program Java menjawab "Nothing at …" sementara yang lain menjawab "No service …". Server JDK men-*decode* seluruh path sebelum kode kita melihatnya, sehingga `%2F` menjadi `/` sungguhan, path-nya berisi tiga garis miring, dan perutean kita mengirimnya ke cabang "tidak ditemukan" untuk path tak dikenal. Program C# dan PHP memotong id-nya lebih dulu dan men-*decode* sesudahnya. ASP.NET Core bertindak lebih cermat lagi: ia men-*decode* segmen path tetapi membiarkan garis miring yang di-*encode* tetap ter-*encode* (`<b>Hi<%2Fb>`), karena garis miring di dalam sebuah nilai tidak boleh mengubah rute mana yang cocok. Tiga server buatan tangan, tiga perilaku; kerangka kerja punya satu, dan terdokumentasi.

***Escaping*.** Ketiga program buatan tangan mencetak `<b>`, bentuk yang sudah di-*escape*, sehingga peramban menampilkan teks `<b>Hi</b>` alih-alih menebalkan "Hi"; masing-masing melakukannya hanya karena kita memanggil fungsi *escaping* pada nilai yang satu itu. ASP.NET Core mencetak kurung sudutnya apa adanya, dan itu benar untuk JSON: jawaban dikirim sebagai `application/json`, yang tidak pernah dijalankan peramban sebagai HTML. Ketika ASP.NET Core memang menghasilkan HTML (Bab 3), view Razor-nya meng-*escape* setiap nilai secara otomatis, kecuali Anda secara tegas memintanya tidak. OWASP, komunitas keamanan web, menempatkan "*escape* semua keluaran yang tidak tepercaya, secara otomatis, di templat" sebagai pertahanan pertama terhadap *cross-site scripting* [23]; kerangka kerja menjadikannya bawaan, bukan sesuatu yang harus diingat.

Bentuk jawaban. Server buatan tangan menjawab HTML, kerangka kerja menjawab JSON. Itu bukan sifat kerangka kerja, melainkan gaya yang kita pilih: *minimal API* ditujukan untuk program; MVC di Bab 3 ditujukan untuk manusia.

1.8 Menjadikannya Lebih Baik: Yang Masih Kita Kerjakan Sendiri

Bahkan versi kerangka kerja masih menyisakan beberapa pekerjaan untuk kita, dan menyebutkannya sekarang memberi peta bab-bab mendatang.

- **Berkas data.** Kita mengurai berkas CSV dengan ekspresi reguler. Toko sungguhan menyimpan datanya di basis data; Bab 5 menghubungkan ASP.NET Core ke SQLite melalui Entity Framework Core, sebuah ORM, dan penguraian itu lenyap.
- **Halaman HTML.** *Minimal API* menjawab JSON. Bab 3 membangun halaman dengan controller, view Razor, *layout* bersama, dan *tag helper*, dan *escaping* menjadi otomatis.
- **Masukan dari formulir.** Tidak ada yang menerima masukan di sini kecuali path. Bab 3 mengikat isian formulir ke objek C#; Bab 6 menambahkan validasi.
- **Login dan peran.** Siapa pun dapat melihat semuanya. Bab 6 menambahkan ASP.NET Core Identity; Bab 10 dan 14 melakukan hal yang sama di Laravel dan Spring Boot.
- **Pengujian.** Kita memeriksa program dengan `curl` secara manual, lewat sebuah skrip. Bab 10 dan 14 menulis pengujian otomatis yang berjalan setiap kali ada perubahan.

Bab 2 melakukan sesuatu yang tidak lazim lebih dulu: sebelum memakai kerangka kerja sungguhan, bab itu membangun kerangka kerja mungil sendiri, sekitar tiga ratus baris, dengan router, *container* injeksi dependensi, pemeta objek-relasional, dan mesin templat. Kerangka kerja itu tidak dimaksudkan untuk dipakai sungguhan; ia dimaksudkan untuk menghapus misterinya, sehingga ketika ASP.NET Core, Laravel, atau Spring Boot melakukan hal yang sama dalam skala besar, Anda tahu apa yang terjadi di dalamnya.

1.9 Memakainya Sungguhan

1.9.1 Toko, klinik, atau kebun Anda sendiri

Setiap usaha kecil yang mengingatkan Anda pada kisah Bu Wati punya daftar harga, dan kebanyakan punya keinginan yang sama: satu perubahan di satu tempat. Jika Anda ingin menyesuaikan program-program bab ini dengan usaha yang Anda kenal, mulailah dari berkas datanya. `Service.csv` milik seorang penjahit mungkin berisi "potong kelim celana" dan "seragam sekolah, satu set"; klinik kecil mungkin mendaftar biaya konsultasi dan laboratoriumnya; kebun buah dan durian yang berjualan di gerbangnya mungkin mendaftar buah-buahannya per kilogram menurut musim. Ubah berkasnya, pertahankan ketiga kolomnya (`id`, `description`, `price`), dan keempat program berjalan tanpa diubah. Lalu ubah kata-kata di halaman beranda. Program mana yang paling mudah disesuaikan? Hampir pasti program dengan baris paling sedikit yang tidak berkaitan dengan bisnisnya.

1.9.2 Memilih kerangka kerja untuk proyek sungguhan

Untuk proyek sungguhan, pilihan di antara ketiganya biasanya ditentukan oleh orang dan tempatnya, bukan oleh kerangka kerjanya. Tim yang sudah menguasai C# dan bekerja dengan produk Microsoft memilih ASP.NET Core; agensi kecil yang pelanggannya memakai *shared hosting* PHP memilih Laravel; bank atau perusahaan besar dengan lingkungan Java memilih Spring Boot. Ketiganya mampu membangun apa pun yang ada di buku ini. Bab 15 membandingkan ketiganya

secara mendalam, dengan pengukuran, setelah Anda membangun fitur yang sama di masing-masing.

1.9.3 Profitina dalam praktik

Profitina, platform visibilitas AI yang namanya dipakai laundry di buku ini, juga dibangun di atas kerangka kerja web, tetapi kerangka kerja keempat: **FastAPI**, kerangka kerja Python, yang dilayani server Uvicorn [24]. Fakta ini mendidik, karena FastAPI memiliki setiap fitur bab ini dengan nama yang berbeda. Rutenya adalah fungsi Python yang diberi dekorator berisi metode dan path; parameter rutenya adalah parameter fungsi, diubah otomatis ke tipe yang dideklarasikan; injeksi dependensinya (`Depends`) menyediakan sesi basis data dan pengguna yang sedang masuk; jawabannya JSON secara bawaan. Dasbor seorang pengguna Profitina dikumpulkan oleh satu rute, `GET /api/me/dashboard`, yang *handler*-nya meminta notifikasi, kanal yang diikuti, dan peringkat kepada beberapa fungsi model, lalu mengembalikan satu dokumen JSON; halaman di peramban merendernya. Itulah *minimal API* pada **Kode 1.5** yang sudah dewasa. Ketika Anda berpindah antarkerangka kerja, carilah delapan langkah yang sama dari Subbab 1.2.1 dengan nama-nama baru; Anda akan selalu menemukannya.

1.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Memasang *runtime* alih-alih SDK. Halaman unduhan .NET menawarkan SDK, *runtime* .NET, dan *runtime* ASP.NET Core. Hanya SDK yang memiliki perintah `dotnet new`, `dotnet build`, dan `dotnet run` untuk pengembang. Jika `dotnet --list-sdks` tidak mencetak apa pun, pasanglah SDK.

Terminal lama. Terminal yang dibuka *sebelum* pemasangan tidak mengenal perintah baru. Setelah memasang apa pun, tutup terminal (termasuk terminal di VS Code) dan buka yang baru sebelum menyimpulkan bahwa pemasangan gagal.

Dua Java. `java -version` menjawab 17 atau 21 padahal Anda memasang 25: JDK yang lebih lama berada lebih depan di `PATH`. Atur `JAVA_HOME` ke folder JDK 25 dan letakkan `%JAVA_HOME%\bin` (Windows) atau `$JAVA_HOME/bin` (macOS, Linux) di *depan* `PATH`.

Menjalankan program dari folder yang salah. Keempat program membaca `../data/Service.csv`, path yang relatif terhadap folder tempat Anda menjalankannya. Jika dijalankan dari tempat lain, program berhenti dengan `FileNotFoundException` (C#), peringatan PHP, atau `NoSuchFileException` (Java). Jalankan masing-masing dari foldernya sendiri, seperti perintah-perintah di bab ini.

Port yang masih terpakai. Jika server berkata alamatnya sudah dipakai, jalankan sebelumnya masih hidup, sering kali di terminal lain atau di VS Code. Hentikan dengan **Ctrl+C** di terminalnya, atau pilih port lain.

Mengira kerangka kerja itu sihir. Ketika sebuah rute tidak cocok atau sebuah nilai tidak terikat, pemula mencoba perubahan acak. Sebaliknya, tanyakan langkah mana dari delapan langkah Subbab 1.2.1 yang gagal, dan lihat apa yang dicatat kerangka kerja untuk langkah itu. Setiap kerangka kerja dapat diminta menjelaskan keputusan peruteannya; Bab 4 memperlihatkan caranya di ASP.NET Core.

Lupa meng-*escape* pada HTML buatan tangan. Setiap kali Anda menyusun HTML dengan penggabungan string, setiap nilai dari luar harus di-*escape*, setiap kali. Lebih baik lagi: biarkan mesin templat melakukannya (Bab 3), dan perlakukan setiap kode yang mematikan *escaping*

(`Html.Raw` di Razor, `{!! !!}` di Blade, `th:utext` di Thymeleaf) sebagai tempat yang harus diperiksa dua kali.

1.11 Ringkasan Bab

- Setiap permintaan ke aplikasi web melewati langkah-langkah yang sama: menunggu, merutekan, mengekstrak, mengambil, memutuskan, merender, menjawab, bertahan. Hanya mengambil dan memutuskan yang berkaitan dengan bisnis.
- Pustaka dipanggil oleh kode Anda; kerangka kerja memanggil kode Anda. Pembalikan kendali inilah yang memungkinkan kerangka kerja mengurus "pipa-pipa" sekali, dengan benar, untuk setiap aplikasi.
- Kerangka kerja web menata kode Anda sebagai Model–View–Controller: model menyimpan data dan aturan, view merender, controller menangani satu permintaan. Router memilih aksi controller. Jagalah controller tetap ramping.
- Buku ini memakai ASP.NET Core 10 (C#), Laravel 13 (PHP 8.5), dan Spring Boot 4 (Java 25), yang bersama-sama mewakili tiga keluarga besar pengembangan sisi server dan yang menurut survei Stack Overflow 2025 dipakai oleh 21,3%, 9,3%, dan 15,6% pengembang profesional.
- Meja kerjanya adalah VS Code dengan beberapa ekstensi, .NET 10 SDK, PHP 8.5 dengan Composer dan pemasang Laravel, JDK 25 dengan Maven (atau *Maven wrapper*), SQLite, dan MySQL 9.7 LTS.
- Ditulis manual, daftar harga itu memerlukan 110 baris dalam C#, 66 dalam PHP, dan 108 dalam Java, dan setiap versi mengambil keputusan kecilnya sendiri tentang garis miring, metode, dan *escaping*. Dengan *minimal API* ASP.NET Core cukup 26 baris, dan kerangka kerja mengambil keputusan itu secara konsisten.

1.12 Soal Latihan

1. Golongkan masing-masing berikut sebagai pustaka atau kerangka kerja, dan berikan alasannya dalam satu kalimat: `HttpListener`; `fgetcsv` milik PHP; ASP.NET Core; `System.Text.Json`; Laravel; `com.sun.net.httpserver` milik JDK; Spring Boot.

2. Ikuti permintaan `GET /services/S0099` melalui **Kode 1.1**. Untuk setiap langkah dari delapan langkah Subbab 1.2.1, sebutkan baris yang melaksanakannya, atau katakan bahwa program tidak melaksanakannya.

3. Tambahkan alamat `/services/under/{max}` pada *minimal API* di **Kode 1.5** yang mengembalikan, sebagai JSON, setiap layanan yang harganya paling banyak `max` rupiah. `/services/under/cheap` harus menjawab `404`. Tunjukkan jawaban untuk `/services/under/7000`.

4. Tambahkan alamat yang sama pada program C# buatan tangan di **Kode 1.1**, yang menjawab dengan tabel HTML. Berapa baris yang Anda tambahkan, dibandingkan Soal 3?

5. Pada Subbab 1.7, `HttpListener` menjawab `411` untuk `curl -X POST`, sedangkan tiga lainnya menjawab `405`. Opsi `curl` mana yang membuat permintaan sampai ke kode kita, dan apa jawabannya kemudian? Mengapa rincian ini menjadi alasan yang baik untuk memakai kerangka kerja?

6. Buang panggilan `Escape` dari pesan 404 pada **Kode 1.1** (baris yang berisi `No service`), jalankan programnya, dan buka `http://127.0.0.1:4010/services/%3Cb%3EHi%3C%2Fb%3E` di peramban. Apa yang Anda lihat, dan mengapa hal ini berbahaya? Kembalikan `Escape` sesudahnya.

7. Catat jawaban ketujuh perintah versi dari Subbab 1.4.6 di komputer Anda sendiri. Baris mana yang membuktikan bahwa ASP.NET Core sendiri terpasang, bukan hanya .NET?

1.13 Penyelesaian Soal Latihan

1. `HttpListener` adalah pustaka: gelung kita memanggil `GetContext()` kapan pun ia menginginkan permintaan berikutnya. `fgetcsv` adalah fungsi pustaka yang dipanggil skrip kita. ASP.NET Core adalah kerangka kerja: setelah `app.Run()` ia memiliki gelungnya dan memanggil *handler* rute kita. `System.Text.Json` adalah pustaka yang kita (atau kerangka kerja) panggil untuk menserialisasi. Laravel adalah kerangka kerja: `public/index.php` miliknya menerima setiap permintaan dan memanggil rute serta controller kita. `com.sun.net.httpserver` adalah pustaka, meskipun ia memanggil balik *handler* kita: ia hanya menerima koneksi serta mengurai baris permintaan dan header, lalu menyerahkan perutean, perenderan, dan setiap keputusan lain kepada kita. Batas keduanya tidak selalu tajam; penentunya adalah siapa yang memiliki alur sebuah permintaan. Spring Boot adalah kerangka kerja: `SpringApplication.run` menjalankan server dan memanggil controller kita.

2. *Menunggu*: `listener.GetContext()`. *Merutekan*: rantai `if`/`else if` pada `path`; cabang `path.StartsWith("/services/") && path.Count(...) == 2` yang diambil. *Mengekstrak*: `Uri.UnescapeDataString(path["/services/".Length..])` menghasilkan `"S0099"`. *Mengambil*: `services.FirstOrDefault(s => s.Id == id)` mengembalikan `null`. *Memutuskan*: `if (found is null)` mengisi `status = 404`. *Merender*: `Page("Not found", ...)` dengan `Escape(id)`. *Menjawab*: lima baris di bawah komentar (3). *Bertahan*: hanya `Console.WriteLine` yang mencatat hasilnya; tidak ada penanganan galat, sehingga sebuah *exception* (misalnya berkas CSV yang hilang) akan menghentikan seluruh server.

3. Satu rute, diletakkan sesudah panggilan `MapGet` lainnya:

```
app.MapGet("/services/under/{max:int}", (int max) =>
    services.Where(s => s.Price <= max));
```

`{max:int}` adalah *route constraint* (batasan rute): rute hanya cocok jika segmennya bilangan bulat, dan nilainya tiba sudah diubah menjadi `int`. Untuk `cheap` tidak ada rute yang cocok, sehingga kerangka kerja menjawab `404` dengan badan kosong. Jawaban untuk `/services/under/7000`, dari menjalankan persis perubahan ini:

```
[{"id":"S0003","description":"Ironing only (per kg)","price":6000.00}]
```

4. Satu versi yang berfungsi, disisipkan tepat sebelum cabang untuk `/services/{id}`. (Di sini urutannya kebetulan tidak berpengaruh, karena cabang itu mensyaratkan tepat dua garis miring; secara umum, uji yang lebih khusus harus mendahului uji yang lebih umum dalam rantai `if`.)

```
    else if (path.StartsWith("/services/under/")
             && int.TryParse(path["/services/under/".Length..], out int max))
    {
        var rows = new StringBuilder();
        foreach (var s in services.Where(s => s.Price <= max))
        {
            rows.Append($"<tr><td>{s.Id}</td><td>{Escape(s.Description)}</td>"
                + $"<td>{Rp(s.Price)}</td></tr>");
        }
        body = Page($"Up to {Rp(max)}", $"<table>{rows}</table>");
    }
```

Itu sebelas baris dibandingkan dua, dan di dalamnya kembali muncul setiap pekerjaan Subbab 1.2.1: uji perutean, ekstraksi dengan `TryParse`, *escaping*, bingkai halaman. Sebuah percobaan menjawab `/services/under/7000` dengan satu baris tabel, `S0003 Ironing only (per kg) Rp 6.000`, dan `/services/under/cheap` dengan `404`, karena `TryParse` gagal dan tidak ada cabang lain yang cocok.

5. `curl -X POST -d "" http://127.0.0.1:4010/services` mengirim badan kosong dengan `Content-Length: 0`. Kali ini `HttpListener` meneruskan permintaan, dan program kita menjawab `405`, seperti yang lain; sebuah percobaan memberi tepat `405` dengan `-d ""` dan `411` tanpanya. Rincian ini penting karena perilaku server buatan tangan bergantung pada aturan pustaka di bawahnya yang mungkin tidak pernah Anda baca. Router kerangka kerja menjawab `405 Method Not Allowed` untuk setiap rute yang ada dengan metode lain, apa pun yang dikirim klien.

6. Peramban menampilkan "No service **Hi**." dengan "Hi" tercetak tebal: teks dari alamat berubah menjadi HTML. Dengan `<script>` sebagai ganti `<b>`, peramban akan menjalankan skrip itu, atas nama situs Bu Wati; itulah serangan *reflected cross-site scripting*, dan tautan yang memuatnya dapat dikirim ke pelanggan mana pun [23]. *Escaping* mengubah `<` menjadi `<`, sehingga peramban menampilkan karakternya alih-alih mematuhinya.

7. Jawabannya bergantung pada komputer Anda; bandingkan dengan Subbab 1.4.6. Baris yang membuktikan ASP.NET Core terpasang adalah baris `dotnet --list-runtimes` yang diawali `Microsoft.AspNetCore.App 10.0.`: itulah *shared framework*-nya. Komputer yang hanya memiliki `Microsoft.NETCore.App` dapat menjalankan program *console*, tetapi tidak aplikasi web.

Daftar Pustaka

[1] R. E. Johnson and B. Foote, "Designing reusable classes," *Journal of Object-Oriented Programming*, vol. 1, no. 2, pp. 22–35, 1988.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA, USA: Addison-Wesley, 1994.

[3] M. Fowler, "Inversion of control containers and the dependency injection pattern," martinfowler.com, Jan. 2004. [Online]. Available: https://martinfowler.com/articles/injection.html

[4] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80," *Journal of Object-Oriented Programming*, vol. 1, no. 3, pp. 26–49, 1988.

[5] Microsoft, ".NET 10.0 Update — September 8, 2026," Microsoft Support, Sep. 2026. [Online]. Available: https://support.microsoft.com/en-us/servicing/dotnet/net-10/2026/net-10-0-update-september-8-2026

[6] B. Crozat, "Laravel versions: latest release and support status," accessed Sep. 2026. [Online]. Available: https://benjamincrozat.com/laravel-versions

[7] PHP.Watch, "PHP 8.5 releases," accessed Sep. 2026. [Online]. Available: https://php.watch/versions/8.5/releases

[8] eosl.date, "Spring Boot end of life (EOL) dates," Aug. 2026 (4.1.1 release). [Online]. Available: https://eosl.date/eol/product/spring-boot/

[9] Oracle, "Consolidated JDK 25 release notes," accessed Sep. 2026. [Online]. Available: https://www.oracle.com/java/technologies/javase/25all-relnotes.html

[10] Stack Overflow, "2025 Developer Survey: Technology — Web frameworks and technologies," 2025. [Online]. Available: https://survey.stackoverflow.co/2025/technology

[11] Microsoft, "Getting started with C# in VS Code," *Visual Studio Code Documentation*, accessed Sep. 2026. [Online]. Available: https://code.visualstudio.com/docs/csharp/get-started

[12] Laravel, "Laravel," Visual Studio Code Marketplace, accessed Sep. 2026. [Online]. Available: https://marketplace.visualstudio.com/items?itemName=laravel.vscode-laravel

[13] Microsoft, "Spring Boot in Visual Studio Code," *Visual Studio Code Documentation*, accessed Sep. 2026. [Online]. Available: https://code.visualstudio.com/docs/java/java-spring-boot

[14] Microsoft, "Download .NET 10.0 (Linux, macOS, and Windows)," accessed Sep. 2026. [Online]. Available: https://dotnet.microsoft.com/en-us/download/dotnet/10.0

[15] Laravel, "Installation," *Laravel 13.x Documentation*, 2026. [Online]. Available: https://laravel.com/docs/13.x/installation

[16] Composer, "Download Composer," accessed Sep. 2026. [Online]. Available: https://getcomposer.org/download/

[17] The Apache Software Foundation, "Download Apache Maven," accessed Sep. 2026. [Online]. Available: https://maven.apache.org/download.cgi

[18] Spring, "Spring Initializr," accessed Sep. 2026. [Online]. Available: https://start.spring.io

[19] SQLite, "SQLite release 3.53.0 on 2026-04-09," 2026. [Online]. Available: https://sqlite.org/releaselog/3_53_0.html

[20] InfoQ, "MySQL 9.7: first major LTS since 8.4 brings enterprise features to Community Edition," May 2026. [Online]. Available: https://www.infoq.com/news/2026/05/mysql-97-lts/

[21] Microsoft, "HttpListener class," *.NET API Reference*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/dotnet/api/system.net.httplistener

[22] Microsoft, "Minimal APIs overview," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/fundamentals/minimal-apis

[23] OWASP Foundation, "Cross Site Scripting prevention cheat sheet," *OWASP Cheat Sheet Series*, accessed Sep. 2026. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

[24] S. Ramírez, "FastAPI," documentation, accessed Sep. 2026. [Online]. Available: https://fastapi.tiangolo.com

Bab 2

Fondasi Bersama Kerangka Kerja Modern: MVC, Perutean, ORM, DI, Templat

Sepintas, ASP.NET Core, Laravel, dan Spring Boot tampak sangat berbeda: bahasanya lain, foldernya lain, kosakatanya lain. Di baliknya, ketiganya dibangun di atas empat gagasan yang sama, yang disusun mengelilingi pola Model–View–Controller yang sama: **router** yang memetakan sebuah alamat ke satu aksi, **container injeksi dependensi** yang membangun objek-objek yang diperlukan setiap aksi, **pemeta objek-relasional** (ORM) yang mengubah baris tabel menjadi objek dan syarat menjadi SQL, serta **mesin templat** yang mengubah objek menjadi HTML di dalam *layout* bersama. Bab ini membangun keempatnya dari nol, dalam sekitar tiga ratus lima puluh baris C# biasa, sebagai kerangka kerja mungil bernama MiniFrame, lalu menulis daftar harga Profitina Laundry di atasnya. Dengan mesinnya terbuka di depan Anda, bab ini kemudian memperlihatkan fitur yang sama di ASP.NET Core, Laravel, dan Spring Boot, sehingga Anda belajar mengenali setiap gagasan di bawah tiga namanya. Pada akhir bab ini Anda akan mampu menjelaskan bagaimana router mencocokkan URL dengan templat rute dan mengekstrak parameter, termasuk batasan bertipe; menjelaskan apa yang dikerjakan container DI, arti masa hidup (*lifetime*) *singleton* dan *transient*, serta mengapa injeksi lewat konstruktor membuat kode lebih mudah diubah dan diuji; menjelaskan bagaimana ORM memetakan baris ke objek dan bagaimana syarat yang ditulis dalam kode menjadi klausa `WHERE`; menjelaskan *layout*, gelung, dan *escaping* otomatis pada templat; serta membaca controller, model, dan view di ketiga kerangka kerja.

2.1 Posisi Kita Sekarang

Bab 1 menulis daftar harga Profitina Laundry empat kali. Tiga versi buatan tangan memerlukan 66 sampai 110 baris masing-masing, dan setiap versi mengambil keputusan kecilnya sendiri tentang perutean, parameter, dan *escaping*. Versi ASP.NET Core memerlukan 26 baris, karena kerangka kerja mengambil alih perutean, ekstraksi parameter, kode status, JSON, dan server itu sendiri.

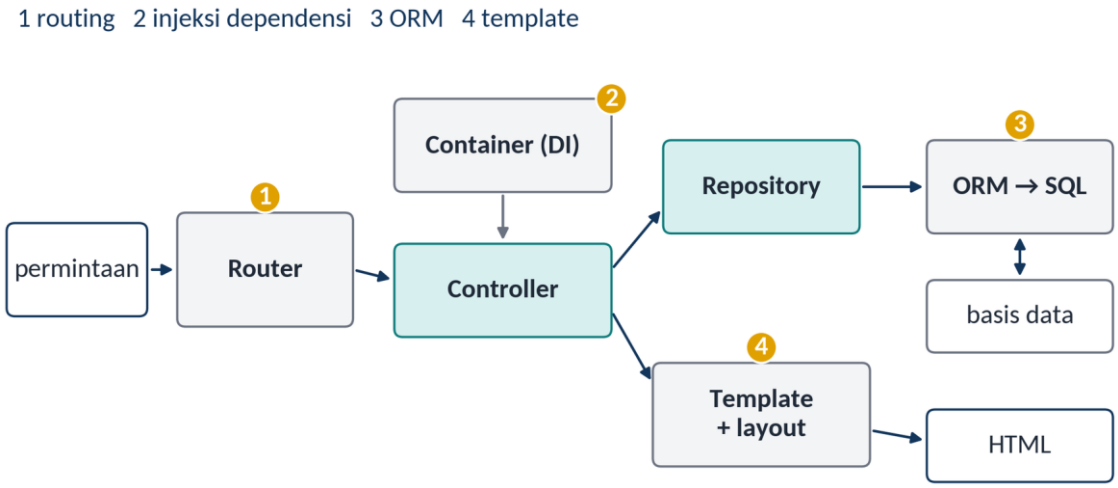
Perbandingan itu meninggalkan pertanyaan yang mengganjal. *Bagaimana* kerangka kerja melakukannya? Ketika `app.MapGet("/services/{id}", (string id) => …)` menerima `/services/S0004`, sesuatu harus memecah path, membandingkannya dengan templat, menyadari bahwa `{id}` adalah parameter, dan meneruskan `"S0004"` ke fungsi yang belum pernah ia lihat. Ketika controller Laravel mendeklarasikan sebuah *repository* di konstruktornya, sesuatu harus membangun *repository* itu lebih dulu. Ketika *repository* Spring punya metode bernama `findByPriceLessThanEqual`, sesuatu harus menulis SQL-nya. Tanpa jawaban, kerangka kerja terasa seperti sihir, dan sihir mustahil di-*debug*.

Keponakan Bu Wati, Dimas, mengalami persis hal ini ketika pertama kali membuka ketiga kerangka kerja. Di templat ASP.NET Core ia menemukan `[HttpGet("{id}")]`; di Laravel `Route::get('/services/{id}', …)`; di Spring `@GetMapping("/{id}")`. Tiga sintaksis, satu

gagasan. Di ASP.NET Core sebuah controller menerima `ILaundryCatalog` di konstruktornya; di Laravel sebuah controller meminta `ServiceRepository` dengan cara yang sama; di Spring konstruktornya menerima `ServiceRepository` padahal tidak ada kode di mana pun yang membuatnya. Lagi-lagi tiga sintaksis, satu gagasan. Bab ini menamai keempat gagasan itu sekali, membangun masing-masing secara manual, lalu menunjuknya di ketiga kerangka kerja.

2.2 Gagasannya dalam Kata-Kata Sederhana

Gambar 2.1 menempatkan keempat fondasi di sepanjang jalur satu permintaan. Router (1) memilih aksinya. Container (2) membangun controller dan semua yang diperlukan controller. Controller meminta data kepada *repository*, dan ORM (3) mengubah permintaan itu menjadi SQL dan baris-barisnya menjadi objek. Mesin templat (4) memasukkan data ke templat halaman, memasukkan halaman ke dalam *layout*, dan mengirim HTML kembali.



Gambar 2.1 Empat fondasi di sepanjang satu permintaan: perutean memilih aksi, injeksi dependensi membangun controller, ORM mengubah objek menjadi SQL dan baris menjadi objek, dan templat mengubah objek menjadi HTML.

DEFINISI. Rute (*route*) adalah aturan yang memasangkan metode HTTP dan *templat* URL seperti `/services/{id}` dengan sebuah aksi. **Router** menyimpan rute-rute dalam *tabel rute* dan, untuk setiap permintaan, mencari rute yang templatnya cocok dengan path, sambil mengekstrak nilai *parameter*-nya (bagian di dalam kurung kurawal). **Batasan** (*constraint*) seperti `{max:int}` membatasi apa yang boleh dicocokkan sebuah parameter.

DEFINISI. Injeksi dependensi (*dependency injection*, DI) berarti sebuah kelas tidak membuat sendiri objek-objek yang ia perlukan; ia mendeklarasikannya, biasanya sebagai parameter konstruktor, lalu menerimanya dari luar. **Container DI** adalah bagian kerangka kerja yang tahu, dari *pendaftaran* (*registration*) yang dibuat saat aplikasi mulai, kelas konkret mana yang harus dibuat untuk setiap tipe yang diminta, membangun seluruh graf dependensi dengan urutan yang benar, dan menentukan *masa hidup* setiap objek: satu objek untuk seluruh aplikasi (*singleton*), objek baru setiap kali (*transient*), atau satu per permintaan (*scoped*).

DEFINISI. Pemeta objek-relasional (*object-relational mapper*, ORM) menerjemahkan antara baris dan kolom basis data relasional dengan objek dan properti sebuah program. Ia membaca sebuah baris menjadi objek dengan mencocokkan nama kolom dan nama properti, dan menerjemahkan kueri yang ditulis dalam bahasa pemrograman (sebuah *syarat* seperti `s.Price <= 15000`) menjadi SQL. **Migrasi** adalah deskripsi berversi atas satu perubahan skema basis data, ditulis dalam kode agar dapat diputar ulang pada setiap salinan basis data.

> **DEFINISI. Templat** adalah berkas teks, biasanya HTML, yang berisi *lubang* untuk diisi oleh **mesin templat** dengan nilai-nilai dari sebuah *model*. Mesin templat yang baik meng-*escape* setiap nilai secara bawaan, sehingga teks dari luar tidak dapat menjadi *markup*, dan mendukung **layout**, yaitu templat bingkai yang dipakai bersama semua halaman, tempat isi masing-masing halaman disisipkan.

Keempat definisi ini adalah seluruh teori bab ini. Sisanya adalah menyaksikan semuanya bekerja.

2.3 Membangun Kerangka Kerja Mungil: MiniFrame

MiniFrame adalah kerangka kerja untuk belajar: cukup kecil untuk dibaca sekali duduk, cukup lengkap untuk menyajikan daftar harga Bu Wati lewat HTTP sungguhan, dan jujur tentang apa yang ditinggalkannya. MiniFrame adalah proyek *console* tanpa satu pun rujukan paket; ia hanya memakai pustaka kelas .NET. **Tabel 2.1** mendaftar berkas-berkasnya.

Tabel 2.1 Berkas-berkas MiniFrame di `Chapter02/miniframe`. Keempat fondasi bersama-sama berjumlah 226 baris.

Berkas	Baris	Fondasi	Isinya
`Router.cs`	55	Perutean	Tabel rute, pencocokan templat, `{name}` dan `{name:int}`
`Container.cs`	45	Injeksi dependensi	Pendaftaran, masa hidup *singleton* dan *transient*, injeksi lewat konstruktor
`Mapper.cs`	90	ORM	Baris ke objek lewat nama; syarat C# diterjemahkan menjadi teks SQL
`Template.cs`	36	Templat	`{{ x }}` di-*escape*, `{{{ x }}}` mentah, gelung `{{#each}}`, *layout*
`Laundry.cs`	66	(aplikasi)	Model, *repository*, dan controller daftar harga
`Program.cs`	63	(awal jalan)	Pendaftaran, rute, demo luring, server HTTP kecil
`templates/*.html`		(view)	`layout.html`, `services.html`, `service.html`, `missing.html`

2.3.1 Perutean

Tugas router adalah mengubah `GET /services/under/15000` menjadi "panggil *handler* ini dengan `max = 15000`". **Kode 2.1** melakukannya dengan cara paling sederhana yang masih memiliki ciri-ciri pokok: daftar rute yang berurutan, templat yang dipecah menjadi segmen, parameter dalam kurung kurawal, dan satu batasan.

Kode 2.1 `Router.cs`: tabel rute, pencocokan templat, dan ekstraksi parameter.

```
namespace MiniFrame;

// Foundation 1, ROUTING: from "GET /services/S0004" to one handler
// plus the values cut out of the address ({ id = "S0004" }).
public class Router
{
    private record Route(string Method, string[] Parts,
        Func<Dictionary<string, string>, Response> Handler);

    private readonly List<Route> routes = [];

    public void Add(string method, string template,
        Func<Dictionary<string, string>, Response> handler) =>
        routes.Add(new Route(method, template.Trim('/').Split('/'), handler));

    public (Func<Dictionary<string, string>, Response>? Handler,
            Dictionary<string, string> Values) Match(string method, string path)
    {
        string[] parts = path.Trim('/').Split('/');
        foreach (var route in routes)
        {
            var values = new Dictionary<string, string>();
            if (route.Method == method && Fits(route.Parts, parts, values))
            {
                return (route.Handler, values);
            }
        }
        return (null, []);
    }

    // "services/{id}" fits "services/S0004"; "{max:int}" needs digits.
    private static bool Fits(string[] template, string[] parts,
        Dictionary<string, string> values)
    {
        if (template.Length != parts.Length) return false;
        for (int i = 0; i < parts.Length; i++)
        {
            string t = template[i];
            if (t.StartsWith('{') && t.EndsWith('}'))
            {
                string[] nameAndRule = t[1..^1].Split(':');
                if (nameAndRule.Length == 2 && nameAndRule[1] == "int"
                    && !parts[i].All(char.IsAsciiDigit)) return false;
                values[nameAndRule[0]] = Uri.UnescapeDataString(parts[i]);
            }
            else if (!t.Equals(parts[i], StringComparison.OrdinalIgnoreCase))
```

Kode 2.1 (Lanjutan) *Router.cs: tabel rute, pencocokan templat, dan ekstraksi parameter.*

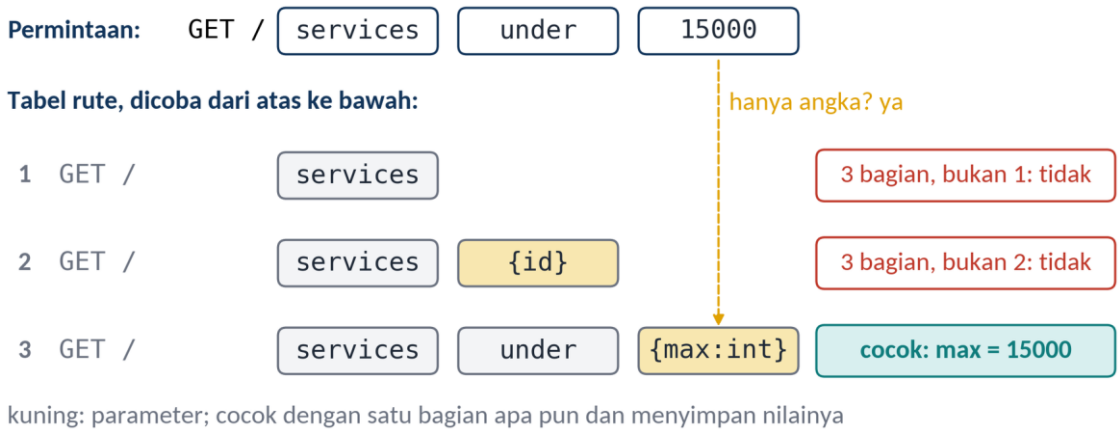
```
            {
                return false;
            }
        }
        return true;
    }
}

public record Response(int Status, string ContentType, string Body);
```

`Add` menyimpan setiap rute sebagai metodenya, templatnya yang dipecah di garis miring, dan sebuah *handler*, yaitu fungsi yang menerima nilai-nilai hasil ekstraksi dalam bentuk kamus (*dictionary*) dan mengembalikan `Response`. `Match` memecah path yang diminta dengan cara yang sama dan mencoba rute-rute **sesuai urutan penambahannya**. `Fits` membandingkan templat dengan path segmen demi segmen. Keduanya harus punya jumlah segmen yang sama. Segmen

literal seperti `services` harus sama dengan segmen path (tanpa membedakan huruf besar dan kecil). Segmen dalam kurung kurawal cocok dengan apa saja, dan nilainya disimpan dengan namanya; jika nama itu membawa batasan `:int`, segmennya harus terdiri atas angka saja, atau rutenya tidak cocok. `Uri.UnescapeDataString` mengubah `%3Cb%3E` kembali menjadi `<b>`, sehingga *handler* menerima nilai seperti yang dimaksud pengguna.

Gambar 2.2 menelusuri permintaan `GET /services/under/15000` melalui tiga rute daftar harga Bu Wati. Templat pertama punya satu segmen sedangkan path punya tiga, jadi tidak mungkin cocok. Templat kedua punya dua. Templat ketiga punya tiga segmen; `services` dan `under` sama dengan segmen path, dan `{max:int}` menerima `15000` karena seluruhnya angka. *Handler* dipanggil dengan `max = 15000`.



Gambar 2.2 Mencocokkan `GET /services/under/15000` dengan tabel rute: jumlah segmen menyingkirkan dua rute pertama, dan rute ketiga cocok karena parameter `{max:int}` menerima angka.

Router sungguhan menambahkan banyak hal: ia mengompilasi seluruh tabel menjadi pohon keputusan agar pencocokan tidak perlu mencoba setiap rute; ia mengizinkan parameter opsional (`{id?}`), nilai bawaan (`{action=Index}`), dan parameter penampung sisa (`{**path}`); ia menawarkan banyak batasan (`int`, `guid`, `min(1000)`, `regex(...)`); dan, yang terpenting, ia tidak bergantung pada urutan pendaftaran. ASP.NET Core mengurutkan templat menurut *precedence* (prioritas): segmen literal mengalahkan parameter, parameter berbatasan mengalahkan parameter tanpa batasan [1]. Aturan "cocok pertama" milik MiniFrame lebih sederhana dan lebih berbahaya, seperti diperlihatkan Soal 1.

2.3.2 Injeksi dependensi

Daftar harga memerlukan tiga jenis objek: *tabel* yang mengenal layanan-layanan, *repository* yang menjawab pertanyaan tentangnya, dan *controller* yang mengubah jawaban menjadi halaman. Masing-masing bergantung pada yang sebelumnya. Tanpa container, kode awal jalan harus membangunnya secara manual, dengan urutan yang benar: `new ServicesController(new CsvServiceRepository(new Table<Service>(...)), new Templates(...))`. Cara itu berjalan untuk tiga kelas, dan menjadi beban pada tiga puluh kelas. Lebih buruk lagi, setiap kelas yang membuat dependensinya sendiri dengan `new` terpatri pada satu kelas konkret, sehingga mengganti berkas CSV dengan basis data, atau dengan tiruan untuk pengujian, berarti menyunting setiap kelas yang membangun *repository*.

Container DI membalik keadaan ini, seperti diuraikan Martin Fowler dalam artikel yang memberi nama pola ini [2]. Setiap kelas *mendeklarasikan* kebutuhannya di konstruktor. Saat awal jalan, aplikasi *mendaftarkan*, sekali saja, kelas konkret mana yang menjawab setiap kebutuhan. Ketika sebuah objek diminta, container melihat konstruktornya, me-*resolve* setiap parameternya dengan

cara yang sama, dan membangun seluruh grafnya. **Kode 2.2** adalah container yang lengkap, meskipun kecil.

Kode 2.2 `Container.cs`: pendaftaran, dua masa hidup, dan injeksi lewat konstruktor dengan refleksi.

```
namespace MiniFrame;

// Foundation 2, DEPENDENCY INJECTION: classes say what they need in their
// constructor; the container builds them, and what they need, in order.
public class Container
{
    private readonly Dictionary<Type, (Type Impl, bool Single)> map = [];
    private readonly Dictionary<Type, object> singles = [];

    public void Singleton<TService, TImpl>() where TImpl : TService =>
        map[typeof(TService)] = (typeof(TImpl), true);

    public void Transient<TService, TImpl>() where TImpl : TService =>
        map[typeof(TService)] = (typeof(TImpl), false);

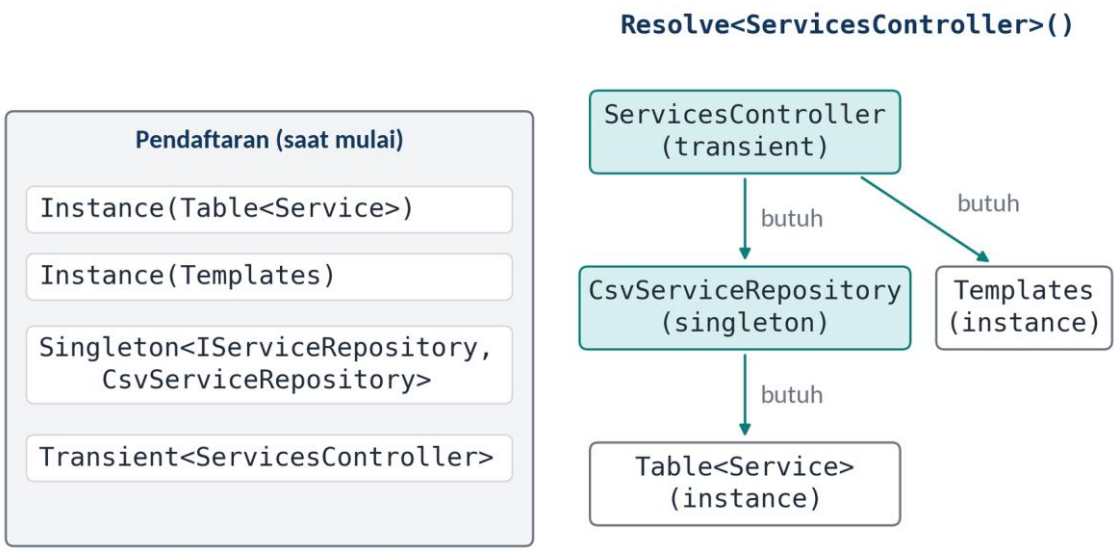
    public void Instance<T>(T obj) where T : notnull
    {
        map[typeof(T)] = (obj.GetType(), true);
        singles[typeof(T)] = obj;
    }

    public T Resolve<T>() => (T)Resolve(typeof(T));

    private object Resolve(Type type)
    {
        if (!map.TryGetValue(type, out var entry))
        {
            throw new InvalidOperationException(
                $"Nothing registered for {type.Name}.");
        }
        if (entry.Single && singles.TryGetValue(type, out var made))
        {
            return made;
        }
        // Take the constructor with the most parameters, resolve each one.
        var ctor = entry.Impl.GetConstructors()
            .OrderByDescending(c => c.GetParameters().Length).First();
        object[] args = ctor.GetParameters()
            .Select(p => Resolve(p.ParameterType)).ToArray();
        object obj = ctor.Invoke(args);
        Console.WriteLine($"  [container] built {entry.Impl.Name}");
        if (entry.Single) singles[type] = obj;
        return obj;
    }
}
```

Kamus `map` mencatat, untuk setiap tipe yang diminta, kelas yang harus dibuat dan apakah ia *singleton*. `Instance` mendaftarkan objek yang sudah ada. `Resolve` adalah jantungnya: ia mencari pendaftarannya, mengembalikan objek tersimpan jika tipenya *singleton* yang sudah pernah dibangun, dan jika tidak, mengambil konstruktor dengan parameter terbanyak, me-*resolve* setiap tipe parameter secara *rekursif*, lalu memanggil konstruktor itu lewat refleksi. Baris yang mencetak `[container] built …` membuat kita dapat menyaksikannya bekerja.

Gambar 2.3 memperlihatkan pendaftaran untuk daftar harga dan graf yang dibangun `Resolve<ServicesController>()` darinya. Controller membutuhkan `IServiceRepository` dan `Templates`; pendaftaran menyatakan bahwa `IServiceRepository` adalah `CsvServiceRepository`, yang membutuhkan `Table<Service>`; tabel dan templat didaftarkan sebagai *instance*.



Gambar 2.3 Pendaftaran yang dibuat saat awal jalan, dan graf yang dibangun container ketika sebuah controller diminta. Hanya controller yang *transient* yang baru pada setiap permintaan.

Dua gagasan pada gambar ini penting di setiap kerangka kerja. Yang pertama, controller bergantung pada *antarmuka* `IServiceRepository`, bukan pada kelas `CsvServiceRepository`. Bab 5 mengganti berkas CSV dengan basis data; satu-satunya baris yang berubah adalah pendaftarannya. Yang kedua adalah **masa hidup**. *Repository* menyimpan data dan aman dipakai bersama, jadi ia *singleton*: dibangun sekali, dipakai oleh setiap permintaan. Controller bersifat *transient*: yang baru untuk setiap permintaan, agar tidak ada yang dilakukan seorang pengunjung bocor ke halaman pengunjung berikutnya. Container sungguhan menambahkan masa hidup ketiga, *scoped*, yaitu satu objek per permintaan yang dipakai bersama oleh semua yang disentuh permintaan itu; itulah masa hidup alami sebuah koneksi basis data, dan Bab 5 memakainya.

2.3.3 Pemetaan objek-relasional

Data Profitina Laundry tinggal di tabel; program berpikir dalam objek. ORM berdiri di antara keduanya. ORM sungguhan berukuran besar (Entity Framework Core, Eloquent, dan Hibernate masing-masing ratusan ribu baris), tetapi dua trik utamanya muat dalam satu halaman, dan **Kode 2.3** memperlihatkan keduanya.

Kode 2.3 `Mapper.cs`: baris menjadi objek dengan mencocokkan nama, dan syarat C# menjadi teks klausa SQL `WHERE`.

```
using System.Globalization;
using System.Linq.Expressions;
using System.Reflection;

namespace MiniFrame;

// Foundation 3, OBJECT-RELATIONAL MAPPING: rows become objects, and a
// condition written in C# becomes the WHERE clause of an SQL query.
public class Table<T>(string name, List<Dictionary<string, string>> rows)
{
    // Row -> object: match each constructor parameter with a column.
    public List<T> Where(Expression<Func<T, bool>> condition)
    {
        Console.WriteLine($"  [mapper] SQL: SELECT * FROM {name} "
            + $"WHERE {Sql(condition.Body)}");
        var test = condition.Compile();
        return rows.Select(ToObject).Where(test).ToList();
    }

    public List<T> All()
    {
        Console.WriteLine($"  [mapper] SQL: SELECT * FROM {name}");
        return rows.Select(ToObject).ToList();
    }

    private static T ToObject(Dictionary<string, string> row)
    {
        var ctor = typeof(T).GetConstructors()[0];
        object[] args = ctor.GetParameters().Select(p => Convert.ChangeType(
            row[p.Name!.ToLowerInvariant()], p.ParameterType,
            CultureInfo.InvariantCulture)).ToArray();
        return (T)ctor.Invoke(args);
    }

    // Translates s => s.Price <= 20000 && s.Id != "S0003" into SQL text.
    private static string Sql(Expression e) => e switch
    {
        BinaryExpression b => $"{Sql(b.Left)} {Op(b.NodeType)} {Sql(b.Right)}",
        MemberExpression { Expression: ParameterExpression } m =>
            m.Member.Name.ToLowerInvariant(),
        MemberExpression m => Literal(Expression.Lambda(m).Compile()
            .DynamicInvoke()),                  // a captured local variable
        ConstantExpression c => Literal(c.Value),
        _ => throw new NotSupportedException(e.NodeType.ToString())
    };
```

Kode 2.3 (Lanjutan) *Mapper.cs: baris menjadi objek dengan mencocokkan nama, dan syarat C# menjadi teks klausa SQL WHERE.*

```
    private static string Op(ExpressionType t) => t switch
    {
        ExpressionType.Equal => "=", ExpressionType.NotEqual => "<>",
        ExpressionType.LessThan => "<", ExpressionType.LessThanOrEqual => "<=",
        ExpressionType.GreaterThan => ">",
        ExpressionType.GreaterThanOrEqual => ">=",
        ExpressionType.AndAlso => "AND", ExpressionType.OrElse => "OR",
        _ => throw new NotSupportedException(t.ToString())
    };

    private static string Literal(object? v) => v switch
    {
        string s => "'" + s.Replace("'", "''") + "'",
        IFormattable f => f.ToString(null, CultureInfo.InvariantCulture),
        _ => "NULL"
    };
}

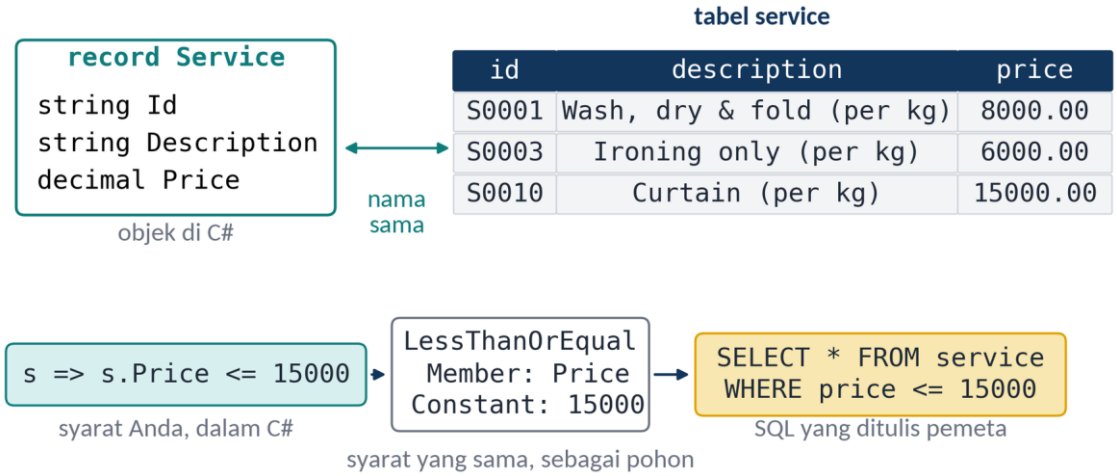
public static class Csv
{
    // Reads a CSV file into rows keyed by the (lower-case) header names.
    public static List<Dictionary<string, string>> Read(string path)
    {
        var lines = File.ReadAllLines(path);
        string[] header = Split(lines[0]);
        return lines.Skip(1).Select(Split)
            .Select(f => header.Zip(f).ToDictionary(
                p => p.First.ToLowerInvariant(), p => p.Second))
            .ToList();
    }

    private static string[] Split(string line)
    {
        var fields = new List<string> { "" };
        bool quoted = false;
        foreach (char c in line)
        {
            if (c == '"') quoted = !quoted;
            else if (c == ',' && !quoted) fields.Add("");
            else fields[^1] += c;
        }
        return fields.ToArray();
    }
}
```

Baris ke objek. `ToObject` mengambil konstruktor pertama kelasnya, di sini *record* `Service(string Id, string Description, decimal Price)`, dan untuk setiap parameter mencari kolom bernama sama dalam huruf kecil (`id`, `description`, `price`), sambil mengubah teksnya ke tipe parameter dengan `Convert.ChangeType`. *Pemetaan berdasarkan konvensi* ini persis yang dilakukan ORM sungguhan secara bawaan: properti `Price` dipetakan ke kolom `price` kecuali Anda menyatakan lain.

Syarat ke SQL. Metode `Where` tidak menerima fungsi biasa; ia menerima `Expression<Func<T, bool>>`. Ketika sebuah parameter bertipe demikian, kompilator C# tidak hanya mengompilasi lambda `s => s.Price <= 15000` menjadi kode mesin; ia juga membangun struktur data yang menggambarkannya, yaitu **pohon ekspresi** (*expression tree*) [3]: sebuah simpul `LessThanOrEqual` yang anak kirinya adalah anggota `Price` dari parameter `s` dan anak kanannya adalah konstanta `15000`. `Sql` menelusuri pohon itu secara rekursif dan mencetak setiap simpul dalam SQL: simpul biner sebagai “kiri operator kanan”, anggota parameter sebagai nama

kolomnya, konstanta atau variabel yang ditangkap sebagai literal SQL. **Gambar 2.4** memperlihatkan kedua arah pemetaan.



Gambar 2.4 Di atas, sebuah baris tabel `service` menjadi objek `Service` karena nama kolom sama dengan nama properti. Di bawah, syarat `s => s.Price <= 15000` dibaca sebagai pohon ekspresi dan dicetak sebagai SQL.

MiniFrame berbuat curang di satu tempat: setelah mencetak SQL-nya, ia tidak mengirimkannya ke basis data. Basis datanya memang tidak ada (datanya berkas CSV), sehingga ia mengompilasi syarat yang sama menjadi kode biasa, `condition.Compile()`, lalu menyaring baris-barisnya di memori. Entity Framework Core melakukan penerjemahan yang sama, dengan jauh lebih banyak jenis simpul (pemanggilan metode seperti `Contains`, *join*, pengurutan, pengelompokan), lalu mengirim SQL-nya ke SQLite, SQL Server, atau MySQL [4]. Gagasannya, kueri yang ditulis dalam C# dan diterjemahkan dari pohon ekspresinya, adalah LINQ, dan Bab 5 memakainya setiap hari.

2.3.4 Templat

Fondasi terakhir mengubah data menjadi HTML. **Kode 2.4** adalah mesin templat dengan tiga ciri yang dimiliki setiap mesin sungguhan: lubang yang di-*escape* secara bawaan, gelung, dan *layout*.

Kode 2.4 `Template.cs`: lubang, gelung, dan *layout*, dengan *escaping* sebagai bawaan.

```
using System.Net;
using System.Text.RegularExpressions;

namespace MiniFrame;

// Foundation 4, TEMPLATES: HTML files with holes. {{ name }} is filled
// with an escaped value, {{{ name }}} with raw HTML, {{#each list}}...
// {{/each}} repeats, and every page goes inside layout.html.
public partial class Templates(string folder)
{
    public string Page(string file, Dictionary<string, object> model)
    {
        string page = Fill(File.ReadAllText(Path.Combine(folder, file)), model);
        model["body"] = page;                  // layout.html says {{{ body }}}
        return Fill(File.ReadAllText(Path.Combine(folder, "layout.html")),
            model);
    }

    private static string Fill(string text, Dictionary<string, object> model)
    {
        text = EachBlock().Replace(text, m =>
            string.Concat(((IEnumerable<Dictionary<string, object>>)
                model[m.Groups[1].Value]).Select(item => Fill(m.Groups[2].Value,
                    item))));
        return Hole().Replace(text, m => m.Groups[1].Success
            ? model[m.Groups[1].Value].ToString()!          // {{{ raw }}}
            : WebUtility.HtmlEncode(model[m.Groups[2].Value].ToString() ?? ""));
    }

    [GeneratedRegex(@"\{\{#each (\w+)\}\}(.*?)\{\{/each\}\}",
        RegexOptions.Singleline)]
    private static partial Regex EachBlock();

    [GeneratedRegex(@"\{\{\{ *(\w+) *\}\}\}|\{\{ *(\w+) *\}\}")]
    private static partial Regex Hole();
}
```

`Page` mengisi templat halaman dengan model, lalu menyimpan hasilnya sebagai `body` dan mengisi `layout.html` dengan model yang sama. `Fill` bekerja dalam dua putaran dengan ekspresi reguler. Putaran pertama mengganti setiap blok `{{#each list}}…{{/each}}` dengan isi bloknya, yang diisi sekali untuk setiap butir daftar. Putaran kedua mengisi setiap lubang. Perbedaan kedua jenis lubang adalah baris terpenting berkas ini: `{{ name }}` diisi dengan `WebUtility.HtmlEncode(value)`, nilai yang sudah di-*escape*, sedangkan `{{{ name }}}` diisi dengan nilai apa adanya. Hanya *layout* yang memakai bentuk mentah, untuk `{{{ body }}}`, karena *body* adalah HTML yang dihasilkan templat kita sendiri. Semua yang berasal dari data atau dari URL melewati kurung kurawal ganda dan di-*escape* secara otomatis; kekhawatiran Bab 1, "setiap nilai, setiap kali", kini menjadi bawaan. **Kode 2.5** dan **Kode 2.6** adalah *layout* dan halaman daftarnya.

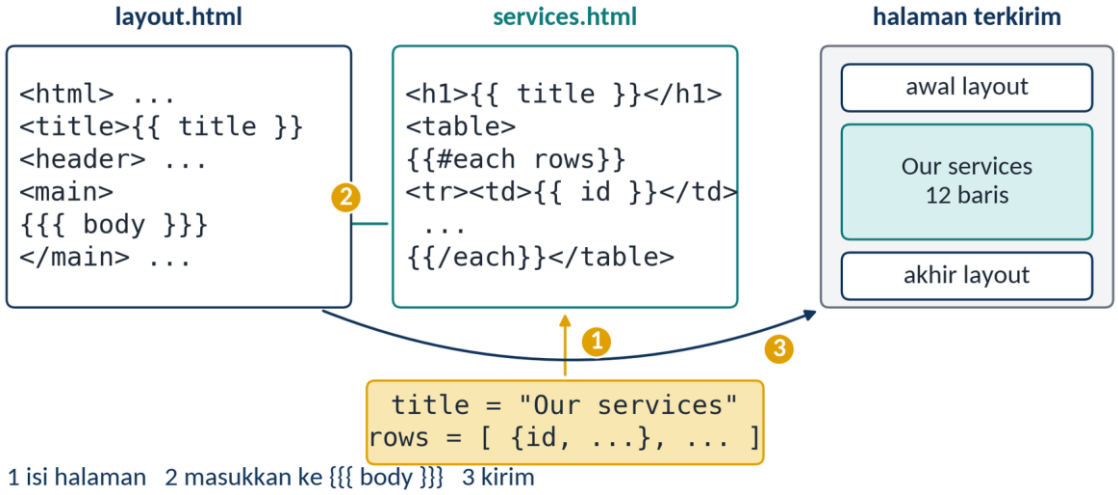
Kode 2.5 `templates/layout.html`: bingkai yang dipakai bersama semua halaman; HTML milik halaman masuk di tempat `{{{ body }}}`.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>{{ title }} - Profitina Laundry</title>
</head>
<body>
  <header><a href="/services">Profitina Laundry</a></header>
  <main>
{{{ body }}}
  </main>
</body>
</html>
```

Kode 2.6 `templates/services.html`: gelung atas baris-baris model, dengan setiap nilai di-*escape*.

```
<h1>{{ title }}</h1>
<table>
{{#each rows}}  <tr><td><a href="/services/{{ id }}">{{ id }}</a></td>
      <td>{{ description }}</td><td>{{ price }}</td></tr>
{{/each}}</table>
```

Gambar 2.5 mengikuti halaman daftar dari templat ke peramban: model dari controller mengisi `services.html` (1), hasilnya dimasukkan ke *layout* di tempat `{{{ body }}}` (2), dan halaman lengkapnya dikirim (3). Tiga langkah yang sama terjadi di Razor, Blade, dan Thymeleaf, seperti diperlihatkan Subbab 2.5.4.



Gambar 2.5 Dari model ke halaman: templat halaman diisi dengan model, hasilnya disisipkan ke dalam *layout*, dan seluruh halaman dikirim.

2.3.5 Aplikasi di atasnya

Dengan keempat fondasi di tempatnya, aplikasinya sendiri pendek. **Kode 2.7** memuat modelnya (*record* `Service`), antarmuka *repository* beserta implementasi CSV-nya, dan controller-nya.

Kode 2.7 `Laundry.cs`: model, *repository*, dan controller daftar harga. Tidak ada perutean, penguraian, SQL, atau *escaping* HTML di sini.

```
using System.Globalization;
using MiniFrame;

// The application code: the only part a framework user writes.
public record Service(string Id, string Description, decimal Price);

public interface IServiceRepository
{
    List<Service> All();
    Service? Find(string id);
    List<Service> UpTo(decimal max);
}

// The M of MVC: data access through the mapper, no SQL strings here.
public class CsvServiceRepository(Table<Service> table) : IServiceRepository
{
    public List<Service> All() => table.All();

    public Service? Find(string id) =>
        table.Where(s => s.Id == id).FirstOrDefault();

    public List<Service> UpTo(decimal max) => table.Where(s => s.Price <= max);
}

// The C of MVC: gets what it needs through its constructor.
public class ServicesController(IServiceRepository repository,
    Templates views)
{
    public Response Index() => List("Our services", repository.All());

    public Response UpTo(int max) =>
        List($"Services up to {Rp(max)}", repository.UpTo(max));

    public Response Details(string id)
    {
        var s = repository.Find(id);
        if (s is null)
        {
            return Html(404, views.Page("missing.html",
                new() { ["title"] = "Not found", ["id"] = id }));
        }
        return Html(200, views.Page("service.html", new()
        {
            ["title"] = s.Id, ["description"] = s.Description,
            ["price"] = Rp(s.Price)
        }));
```

Kode 2.7 (Lanjutan) *Laundry.cs: model, repository, dan controller daftar harga. Tidak ada perutean, penguraian, SQL, atau escaping HTML di sini.*

```
    }

    private Response List(string title, List<Service> list) =>
        Html(200, views.Page("services.html", new()
        {
            ["title"] = title,
            ["rows"] = list.Select(s => new Dictionary<string, object>
            {
                ["id"] = s.Id, ["description"] = s.Description,
                ["price"] = Rp(s.Price)
            })
        }));

    private static Response Html(int status, string body) =>
        new(status, "text/html; charset=utf-8", body);

    private static string Rp(decimal amount) =>
        "Rp " + amount.ToString("N0", CultureInfo.InvariantCulture)
            .Replace(",", ".");
}
```

Bacalah controller itu sebagai daftar isi. `Index` meminta semua layanan kepada *repository* dan memilih halaman daftar. `UpTo` meminta layanan sampai harga tertentu. `Details` meminta satu layanan dan memilih halaman rincian atau, dengan status 404, halaman “missing”. *Repository* tidak berisi string SQL; ia menyatakan syarat dalam C#, `s => s.Id == id` dan `s => s.Price <= max`, dan menyerahkan penerjemahannya kepada pemeta. Tak satu pun di berkas ini mengenal HTTP, format CSV, atau *escaping*. Itulah arti “controller ramping”.

Kode 2.8 adalah kode awal jalan, bagian yang sebagian besar ditulis kerangka kerja sungguhan untuk Anda: pendaftaran, rute, dan gelung yang melayani HTTP. Di dalamnya juga ada demo luring, dijalankan dengan `--demo`, yang mengirim lima permintaan langsung ke router tanpa jaringan, sehingga log setiap fondasi dapat dibaca di satu tempat.

Kode 2.8 `Program.cs`: pendaftaran, rute, demo luring, dan server HTTP minimal.

```
using System.Net;
using System.Text;
using MiniFrame;

// Wiring: what a framework does for you when the application starts.
var container = new Container();
container.Instance(new Table<Service>("service", Csv.Read("data/Service.csv")));
container.Instance(new Templates("templates"));
container.Singleton<IServiceRepository, CsvServiceRepository>();
container.Transient<ServicesController, ServicesController>();

var router = new Router();
router.Add("GET", "/services",
    _ => container.Resolve<ServicesController>().Index());
router.Add("GET", "/services/{id}",
    v => container.Resolve<ServicesController>().Details(v["id"]));
router.Add("GET", "/services/under/{max:int}",
    v => container.Resolve<ServicesController>().UpTo(int.Parse(v["max"])));

Response Handle(string method, string path)
{
    var (handler, values) = router.Match(method, path);
    string shown = string.Join(", ", values.Select(p => $"{p.Key}={p.Value}"));
    Console.WriteLine($"  [router] {method} {path} -> "
        + (handler is null ? "no route" : $"handler {{ {shown} }}"));
    return handler is null
        ? new Response(404, "text/plain; charset=utf-8", $"Nothing at {path}")
        : handler(values);
}

if (args.Length > 0 && args[0] == "--demo")
{
    // Answers a few requests without a network, to show each foundation.
    foreach (string path in new[] { "/services/S0004", "/services/under/15000",
                 "/services/under/cheap", "/services/%3Cb%3E", "/prices" })
    {
        Console.WriteLine($"REQUEST GET {path}");
        var r = Handle("GET", path);
        string Inside(string tag) => r.Body.Contains($"<{tag}>")
            ? r.Body.Split($"<{tag}>")[1].Split($"</{tag}>")[0] : "";
        Console.WriteLine($"RESPONSE {r.Status} {r.ContentType}");
        Console.WriteLine(r.ContentType.StartsWith("text/plain")
            ? $"  {r.Body}\n"
            : $"  <h1>{Inside("h1")}</h1> <p>{Inside("p")}</p> "
              + $"rows: {r.Body.Split("<tr>").Length - 1}\n");
    }
```

Kode 2.8 (Lanjutan) *Program.cs: pendaftaran, rute, demo luring, dan server HTTP minimal.*

```
    return;
}

var listener = new HttpListener();
listener.Prefixes.Add("http://127.0.0.1:4020/");
listener.Start();
Console.WriteLine("MiniFrame on http://127.0.0.1:4020/services (Ctrl+C stops)");
while (true)
{
    var context = listener.GetContext();
    var r = Handle(context.Request.HttpMethod, context.Request.Url!.AbsolutePath);
    byte[] bytes = Encoding.UTF8.GetBytes(r.Body);
    context.Response.StatusCode = r.Status;
    context.Response.ContentType = r.ContentType;
    context.Response.OutputStream.Write(bytes);
    context.Response.Close();
}
```

Perhatikan bahwa *handler* tidak membuat controller dengan `new`; ia memintanya kepada container, `container.Resolve<ServicesController>()`. Begitu jugalah kerangka kerja sungguhan memanggil controller Anda: ia me-*resolve*-nya dari container, itulah sebabnya controller cukup mendeklarasikan apa yang ia butuhkan.

2.4 Menjalankannya

Buka terminal di `Chapter02/miniframe` dan jalankan demonya:

```
dotnet run -- --demo
```

Dua tanda hubung memisahkan argumen untuk `dotnet` dari argumen untuk program kita. Inilah keluaran sesungguhnya; baris-baris yang menjorok adalah apa yang dicetak setiap fondasi selama bekerja.

```
REQUEST GET /services/S0004
  [router] GET /services/S0004 -> handler { id=S0004 }
  [container] built CsvServiceRepository
  [container] built ServicesController
  [mapper] SQL: SELECT * FROM service WHERE id = 'S0004'
RESPONSE 200 text/html; charset=utf-8
  <h1>S0004</h1> <p>Dry cleaning - shirt/blouse (per item): Rp 20.000</p> rows: 0

REQUEST GET /services/under/15000
  [router] GET /services/under/15000 -> handler { max=15000 }
  [container] built ServicesController
  [mapper] SQL: SELECT * FROM service WHERE price <= 15000
RESPONSE 200 text/html; charset=utf-8
  <h1>Services up to Rp 15.000</h1> <p></p> rows: 4

REQUEST GET /services/under/cheap
  [router] GET /services/under/cheap -> no route
RESPONSE 404 text/plain; charset=utf-8
  Nothing at /services/under/cheap

REQUEST GET /services/%3Cb%3E
  [router] GET /services/%3Cb%3E -> handler { id=<b> }
  [container] built ServicesController
  [mapper] SQL: SELECT * FROM service WHERE id = '<b>'
RESPONSE 404 text/html; charset=utf-8
  <h1>Not found</h1> <p>There is no service with the code &lt;b&gt;.</p> rows: 0

REQUEST GET /prices
  [router] GET /prices -> no route
RESPONSE 404 text/plain; charset=utf-8
  Nothing at /prices
```

Setiap blok menceritakan kisah kecil.

- **`/services/S0004`.** Router memilih rute rincian dan mengekstrak `id=S0004`. Container membangun *repository* (pemakaian pertama sebuah *singleton*) lalu controller. Syarat milik *repository* menjadi `WHERE id = 'S0004'`. Judul dan paragraf halaman menampilkan layanannya.
- **`/services/under/15000`.** Kali ini container hanya membangun controller: *repository* adalah *singleton* dan sudah ada. Syarat `s.Price <= max` menjadi `price <= 15000`: pemeta mengganti variabel tangkapan `max` dengan nilainya. Empat layanan berharga paling banyak Rp 15.000: S0001, S0002, S0003, dan S0010.
- **`/services/under/cheap`.** Tidak ada rute yang cocok: batasan rute ketiga menolak `cheap`, dan dua rute lainnya punya jumlah segmen yang salah. Tidak ada yang dibangun, tidak ada yang dikueri, dan jawabannya 404 polos.

- **`/services/%3Cb%3E`.** Router men-*decode* nilainya menjadi `<b>`; *repository* tidak menemukan apa-apa; halaman "missing" mencetak kodenya sebagai `<b>`, di-*escape* oleh mesin templat tanpa controller melakukan apa pun. Perhatikan juga SQL-nya: nilai `<b>` berada di dalam tanda kutip sebagai literal. Subbab 2.6 menjelaskan mengapa ORM sungguhan mengirim nilai semacam itu sebagai *parameter*.
- **`/prices`.** Sama sekali tidak ada rutenya.

Kelas-kelas yang sama juga melayani HTTP sungguhan. `dotnet run` tanpa `--demo` menjalankan server di port 4020; `Chapter02/verify.sh` mengajukan empat pertanyaan kepadanya dengan `curl`. Kode statusnya `200` untuk `/services`, `/services/S0010`, dan `/services/under/7000`, dan `404` untuk `/services/under/x`, dan badan jawaban untuk `/services/under/7000` adalah halaman berikut, *layout* yang membungkus daftar berisi satu baris.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Services up to Rp 7.000 - Profitina Laundry</title>
</head>
<body>
  <header><a href="/services">Profitina Laundry</a></header>
  <main>
<h1>Services up to Rp 7.000</h1>
<table>
  <tr><td><a href="/services/S0003">S0003</a></td>
      <td>Ironing only (per kg)</td><td>Rp 6.000</td></tr>
</table>

  </main>
</body>
</html>
```

2.5 Fitur yang Sama di Tiga Kerangka Kerja Sungguhan

Setelah setiap fondasi dibangun sekali secara manual, kita dapat membaca kerangka kerja sungguhan. Subbab ini menulis tiga alamat yang sama (daftar, rincian, layanan sampai harga tertentu) di Laravel dan di Spring Boot, dan menunjuk versi ASP.NET Core yang dibangun utuh oleh Bab 3. Berkas-berkas Laravel dan Spring ada di `Chapter02/snippets`. Seperti dijelaskan Bab 1, mesin yang membangun buku ini tidak dapat mengunduh Laravel maupun Spring, sehingga berkas-berkas ini **diperiksa tetapi tidak dijalankan**: berkas PHP dengan `php -l`, yang melaporkan galat sintaksis, dan berkas Java dengan `javac` terhadap segelintir tipe pengganti mungil yang meniru beberapa nama Spring dan Jakarta yang dipakainya (`snippets/spring/stubs`), yang memeriksa sintaksis dan tipe. Bab 7 dan 11 menjadikannya aplikasi yang berjalan. Berkas-berkas itu mengikuti dokumentasi terkini Laravel 13 dan Spring Boot 4 [5], [6].

2.5.1 Perutean

Di **Laravel**, rute tinggal di satu berkas terpusat, `routes/web.php`, dan terbaca hampir seperti kalimat. **Kode 2.9** adalah seluruh perutean daftar harga.

Kode 2.9 `routes/web.php` di Laravel: tiga rute, masing-masing menunjuk metode controller; `whereNumber` adalah batasannya.

```php
<?php

use App\Http\Controllers\ServiceController;
use Illuminate\Support\Facades\Route;

Route::get('/services', [ServiceController::class, 'index']);
Route::get('/services/under/{max}', [ServiceController::class, 'under'])
    ->whereNumber('max');
Route::get('/services/{id}', [ServiceController::class, 'show']);
```

`Route::get` memasangkan metode `GET` dan sebuah templat dengan aksi, yang ditulis sebagai larik berisi kelas controller dan nama metodenya. Parameter berada dalam kurung kurawal, seperti di MiniFrame, dan `->whereNumber('max')` adalah batasan yang di MiniFrame ditulis `{max:int}` [5]. Laravel mencocokkan rute sesuai urutan pendaftarannya, seperti MiniFrame; itulah sebabnya `/services/under/{max}` yang lebih khusus ditulis sebelum `/services/{id}`, meskipun di sini jumlah segmennya sudah memisahkan keduanya.

Di **Spring Boot**, rute berupa anotasi pada metode-metode controller, dengan gaya yang oleh ASP.NET Core disebut *attribute routing*: `@RequestMapping("/services")` pada kelas memberi prefiks, dan `@GetMapping("/{id}")` pada metode menambahkan sisanya [7]. Batasan ditulis sebagai ekspresi reguler setelah titik dua, `{max:\\d+}`, "satu angka atau lebih". Spring tidak bergantung pada urutan: ia memilih pola cocok yang paling khusus.

Di **ASP.NET Core**, kedua gaya tersedia. Bab 3 memakai rute *konvensional*, satu pola terpusat `{controller=Home}/{action=Index}/{id?}` yang memetakan `/Outlets/Details/O01` ke metode `Details` milik `OutletsController` dengan `id = "O01"`, dan *attribute route* seperti `[HttpGet("under/{max:int:min(1000)}")]` pada metode-metode controller yang ditandai `[Route("services")]` [1].

2.5.2 Injeksi dependensi

Ketiga kerangka kerja punya container, dan di ketiganya controller menerima dependensinya lewat konstruktor. Bedanya terletak pada cara pendaftaran dibuat.

- **ASP.NET Core** mendaftarkan layanan secara eksplisit di `Program.cs`, dengan masa hidup tercantum di nama metodenya: `builder.Services.AddSingleton<ILaundryCatalog, CsvCatalog>()`, `AddScoped<…>()`, `AddTransient<…>()` [8]. Controller dibuat oleh kerangka kerja untuk setiap permintaan dan menerima apa yang dideklarasikannya.
- *Service container* **Laravel** me-*resolve* sebagian besar kelas tanpa pendaftaran apa pun, dengan merefleksi konstruktornya persis seperti MiniFrame (Laravel menyebutnya *zero configuration resolution*); sebuah antarmuka memerlukan satu baris di *service provider*, `$this->app->singleton(ServiceRepository::class, EloquentServiceRepository::class)` [9].
- **Spring** menemukan kelas-kelas bertanda `@Component`, `@Service`, `@Repository`, atau `@Controller` dengan memindai paket aplikasi, membuat satu objek untuk masing-masing (*singleton* adalah *scope* bawaan Spring), lalu menyuntikkannya lewat konstruktor; jika sebuah kelas hanya punya satu konstruktor, anotasi pun tidak diperlukan padanya [10]. Pada **Kode 2.18** di bawah, konstruktor `ServiceController(ServiceRepository repository)` adalah seluruh injeksinya.

Kata-katanya berbeda; gagasannya adalah `Container` milik MiniFrame: nyatakan apa yang Anda butuhkan, daftarkan sekali apa yang menjawab kebutuhan itu, biarkan container membangun grafnya.

2.5.3 Pemetaan objek-relasional

Ada dua keluarga ORM, yang dinamai Martin Fowler dalam *Patterns of Enterprise Application Architecture* [11]. Pada pola **Active Record**, objek model tahu cara memuat dan menyimpan dirinya sendiri: `Service::find('S0004')`, `$service->save()`. Pada pola **Data Mapper**, model adalah objek biasa, dan objek terpisah (sebuah *context*, *session*, atau *repository*) memindahkannya dari dan ke basis data. Eloquent milik Laravel adalah ORM Active Record; Entity Framework Core dan Spring Data JPA (dengan Hibernate di bawahnya) adalah Data Mapper. `Table<T>` milik MiniFrame adalah Data Mapper kecil.

Kode 2.10 adalah model Eloquent untuk tabel layanan. Isinya nyaris kosong, karena Eloquent memetakan berdasarkan konvensi: kelas `Service` akan dipetakan ke tabel `services` dengan kunci bilangan bulat yang bertambah otomatis, dan keempat baris di dalamnya hanya menyatakan di mana basis data Profitina Laundry menyimpang dari konvensi itu [12].

Kode 2.10 `app/Models/Service.php`: model Eloquent. Properti-propertinya hanya menyatakan apa yang berbeda dari konvensi Laravel.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

// One row of the table "service" becomes one Service object.
class Service extends Model
{
    protected $table = 'service';
    protected $keyType = 'string';     // ids look like "S0004"
    public $incrementing = false;
    public $timestamps = false;        // the table has no created_at/updated_at
}
```

Versi Spring terdiri atas dua bagian. **Kode 2.11** adalah *entity*, kelas biasa yang anotasinya memberi tahu JPA tabel dan kuncinya; **Kode 2.12** adalah *repository*, yang hanya berupa antarmuka.

Kode 2.11 `Service.java`: *entity* JPA yang dipetakan ke tabel `service`.

```java
package com.profitina.laundry;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import java.math.BigDecimal;

// One row of the table "service" becomes one Service object.
@Entity
@Table(name = "service")
public class Service {
    @Id
    private String id;
    private String description;
    private BigDecimal price;

    protected Service() {}                    // JPA needs an empty constructor

    public String getId() { return id; }
    public String getDescription() { return description; }
    public BigDecimal getPrice() { return price; }
}
```

Kode 2.12 `ServiceRepository.java`: *repository* Spring Data. Kuerinya diturunkan dari nama metodenya.

```
package com.profitina.laundry;

import java.math.BigDecimal;
import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;

// No class to write: Spring Data reads the method name and writes the query.
public interface ServiceRepository extends JpaRepository<Service, String> {
    List<Service> findByPriceLessThanEqual(BigDecimal max);
}
```

Tidak ada yang menulis kelas yang mengimplementasikan `ServiceRepository`. Ketika aplikasi mulai, Spring Data membuatkannya: `findAll` dan `findById` berasal dari `JpaRepository`, dan untuk `findByPriceLessThanEqual` ia *mengurai nama metodenya*, "cari berdasarkan `price`, kurang dari atau sama dengan", lalu menurunkan kueri `… where s.price <= ?1` [6]. Ini penerjemahan yang sama dengan pohon ekspresi MiniFrame, hanya berangkat dari sebuah nama alih-alih lambda. Entity Framework Core berangkat dari lambda, persis seperti MiniFrame: `db.Services.Where(s => s.Price <= max)` menjadi SQL [4].

2.5.4 Templat

Ketiga kerangka kerja punya tiga bahasa templat, dan masing-masing memiliki tiga ciri **Kode 2.4**: lubang yang di-*escape*, gelung, dan *layout*.

Blade, mesin milik Laravel, menulis lubang yang di-*escape* sebagai `{{ $value }}` dan lubang mentah sebagai `{!! $value !!}`; direktif seperti `@foreach` dan `@if` diawali `@`; sebuah halaman menyebut *layout*-nya dengan `@extends` dan mengisi slot `@yield` milik *layout* dengan `@section` [13]. **Kode 2.13** dan **Kode 2.14** adalah *layout* dan halaman daftarnya.

Kode 2.13 `resources/views/layouts/app.blade.php`: *layout* Blade; `@yield` menandai slot-slotnya.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>@yield('title') - Profitina Laundry</title>
</head>
<body>
  <header><a href="/services">Profitina Laundry</a></header>
  <main>
    @yield('content')
  </main>
</body>
</html>
```

Kode 2.14 `resources/views/services/index.blade.php`: halaman daftar, yang mengisi dua slot milik *layout*.

```
@extends('layouts.app')

@section('title', $title)

@section('content')
  <h1>{{ $title }}</h1>
  <table>
    @foreach ($services as $s)
      <tr><td><a href="/services/{{ $s->id }}">{{ $s->id }}</a></td>
          <td>{{ $s->description }}</td><td>{{ $s->price }}</td></tr>
    @endforeach
  </table>
@endsection
```

Controller yang memasok view ini adalah **Kode 2.15**. Setiap metodenya mengembalikan `view('services.index', [...])`, yaitu nama templat dan modelnya, persis seperti `views.Page("services.html", model)` milik MiniFrame.

Kode 2.15 `app/Http/Controllers/ServiceController.php`: controller Laravel. `findOrFail` menjawab 404 dengan sendirinya jika kodenya tidak ada.

```
<?php

namespace App\Http\Controllers;

use App\Models\Service;
use Illuminate\View\View;

class ServiceController extends Controller
{
    public function index(): View
    {
        return view('services.index', [
            'title' => 'Our services',
            'services' => Service::all(),
        ]);
    }

    public function under(int $max): View
    {
        return view('services.index', [
            'title' => 'Services up to Rp ' . number_format($max, 0, ',', '.'),
            'services' => Service::where('price', '<=', $max)->get(),
        ]);
    }

    public function show(string $id): View
    {
        return view('services.show', ['service' => Service::findOrFail($id)]);
    }
}
```

Thymeleaf, mesin yang dipakai bersama Spring Boot di buku ini, menempuh jalan berbeda: templatnya berupa berkas HTML yang sah, dan atribut-atributnya, yang semuanya diawali `th:`, memberi tahu mesin apa yang harus diganti [14]. `th:text` mengisi sebuah elemen dengan nilai yang di-*escape* (`th:utext` tidak meng-*escape*), `th:each` mengulang sebuah elemen, dan `th:replace` menyisipkan *fragment* dari berkas lain, yang menjadi cara Thymeleaf membangun *layout*. Teks pengganti di dalam elemen (`Our services`, `S0001`) terlihat ketika seorang perancang membuka berkasnya langsung di peramban, dan diganti ketika Spring merendernya. **Kode 2.16** memuat *fragment* bersama dan **Kode 2.17** adalah halaman daftarnya.

Kode 2.16 `templates/layout.html`: dua *fragment* Thymeleaf, bagian *head* (dengan parameter) dan bagian kepala halaman.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head th:fragment="head(title)">
  <meta charset="utf-8">
  <title th:text="${title} + ' - Profitina Laundry'">Profitina Laundry</title>
</head>
<body>
  <header th:fragment="header">
    <a th:href="@{/services}">Profitina Laundry</a>
  </header>
</body>
</html>
```

Kode 2.17 `templates/services/index.html`: halaman daftar dalam Thymeleaf; `th:each` adalah gelungnya, `th:text` lubang yang di-*escape*.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head th:replace="~{layout :: head(${title})}"></head>
<body>
  <header th:replace="~{layout :: header}"></header>
  <main>
    <h1 th:text="${title}">Our services</h1>
    <table>
      <tr th:each="s : ${services}">
        <td><a th:href="@{/services/{id}(id=${s.id})}"
               th:text="${s.id}">S0001</a></td>
        <td th:text="${s.description}">Wash, dry &amp;amp; fold (per kg)</td>
        <td th:text="${s.price}">8000.00</td>
      </tr>
    </table>
  </main>
</body>
</html>
```

Kode 2.18 adalah controller Spring. Metode-metodenya menaruh model ke dalam objek `Model` dan mengembalikan nama templatnya.

Kode 2.18 `ServiceController.java`: controller Spring MVC, dengan *repository*-nya disuntikkan lewat konstruktor.

```
package com.profitina.laundry;

import java.math.BigDecimal;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.server.ResponseStatusException;

@Controller
@RequestMapping("/services")
public class ServiceController {
    private final ServiceRepository repository;

    public ServiceController(ServiceRepository repository) {   // injected
        this.repository = repository;
    }

    @GetMapping
    public String index(Model model) {
        model.addAttribute("title", "Our services");
        model.addAttribute("services", repository.findAll());
        return "services/index";
    }

    @GetMapping("/under/{max:\\d+}")
    public String under(@PathVariable int max, Model model) {
        model.addAttribute("title", "Services up to Rp " + max);
        model.addAttribute("services",
            repository.findByPriceLessThanEqual(BigDecimal.valueOf(max)));
        return "services/index";
    }

    @GetMapping("/{id}")
    public String show(@PathVariable String id, Model model) {
        Service service = repository.findById(id).orElseThrow(
            () -> new ResponseStatusException(HttpStatus.NOT_FOUND));
        model.addAttribute("service", service);
        return "services/show";
    }
}
```

Razor, mesin milik ASP.NET Core, adalah pokok bahasan Bab 3. Razor menandai C# dengan satu `@`: `@Model.Name` adalah lubang yang di-*escape*, `@foreach (var s in Model) { … }` sebuah gelung, `@Html.Raw(x)` pengecualian yang disengaja, dan `_Layout.cshtml` bersama menyimpan bingkainya, tempat `@RenderBody()` menyisipkan setiap halaman [15].

2.5.5 Satu fitur, tiga kerangka kerja

Tabel 2.2 merangkum subbab ini ke dalam satu tabel, dengan MiniFrame sebagai kolom pertama. Bacalah per baris: setiap baris adalah satu gagasan, yang kini Anda kenal dari dalam, dieja dengan empat cara.

Tabel 2.2 Keempat fondasi di MiniFrame dan di tiga kerangka kerja buku ini.

Fondasi	MiniFrame	ASP.NET Core	Laravel	Spring Boot
Tabel rute	`router.Add("GET", ...)`	`MapControllerRoute`, `[HttpGet]`	`Route::get(...)` di `routes/web.php`	`@GetMapping`
Parameter	`{id}`	`{id}`	`{id}`	`{id}`
Batasan	`{max:int}`	`{max:int}`	`->whereNumber('max')`	`{max:\\d+}`
Pendaftaran	`container.Singleton<I, C>()`	`builder.Services.AddSingleton<I, C>()`	`$this->app->singleton(I::class, C::class)`	`@Repository`, `@Service` (dipindai)
Injeksi	konstruktor	konstruktor	konstruktor	konstruktor
Baris ke objek	`Table<Service>`	*entity* EF Core	model Eloquent	`@Entity` JPA
Kueri	`table.Where(s => s.Price <= max)`	`db.Services.Where(s => s.Price <= max)`	`Service::where('price', '<=', $max)`	`findByPriceLessThanEqual(max)`
Lubang di-*escape*	`{{ x }}`	`@x`	`{{ $x }}`	`th:text="${x}"`
Lubang mentah	`{{{ x }}}`	`@Html.Raw(x)`	`{!! $x !!}`	`th:utext="${x}"`
Gelung	`{{#each rows}}`	`@foreach`	`@foreach`	`th:each`
Layout	`layout.html`, `{{{ body }}}`	`_Layout.cshtml`, `@RenderBody()`	`@extends`, `@yield`	*fragment*, `th:replace`

2.6 Menjadikannya Lebih Baik: Yang Ditinggalkan MiniFrame

MiniFrame benar untuk apa yang dikerjakannya, tetapi kerangka kerja sungguhan mengerjakan jauh lebih banyak, dan setiap bagian yang hilang adalah pelajaran tentang mengapa kerangka kerja berukuran besar.

- **Parameter, bukan literal, di SQL.** MiniFrame mencetak nilai ke dalam teks SQL (`WHERE id = '<b>'`), sambil menggandakan setiap tanda kutip. ORM sungguhan mengirim SQL dengan penanda tempat (`WHERE id = @p0`) dan nilainya secara terpisah, sehingga tidak ada nilai, seaneh apa pun, yang dapat mengubah makna pernyataannya. Kueri berparameter adalah pertahanan utama terhadap *SQL injection* [16].
- **Prioritas rute dan kecepatan.** MiniFrame mencoba rute satu per satu sesuai urutan. Router sungguhan mengompilasi tabel rute menjadi pohon dan mengurutkan templat, sehingga hasilnya tidak bergantung pada urutan pendaftaran dan pencocokan tetap cepat dengan ribuan rute [1].
- **Masa hidup *scoped* dan pelepasan objek.** MiniFrame mengenal *singleton* dan *transient*. Container sungguhan menambahkan masa hidup *scoped* dan melepaskan objek (misalnya menutup koneksi basis data) di akhir setiap permintaan [8].
- **Konkurensi.** Server MiniFrame melayani satu permintaan pada satu waktu, dan kamus *singleton*-nya tidak aman diakses bersamaan. Kerangka kerja sungguhan dibangun untuk ribuan permintaan sekaligus.

- **Templat terkompilasi.** MiniFrame membaca dan mengurai setiap berkas templat pada setiap permintaan. Razor mengompilasi templat menjadi kelas C# ketika proyek di-*build*; Blade dan Thymeleaf menyimpan bentuk terkompilasinya di *cache*.
- **Migrasi.** MiniFrame tidak dapat membuat atau mengubah tabel. Setiap ORM sungguhan punya migrasi (Bab 5, 9, dan 11).
- **Semua sisanya** dalam tabel di Subbab 1.2.4: *model binding*, validasi, keamanan, konfigurasi, halaman galat, pencatatan.

Menambahkan semua ini ke MiniFrame adalah cara yang sangat baik untuk menghabiskan setahun dan cara yang buruk untuk menjalankan laundry. Itulah intinya: mesinnya dapat dipahami, dan juga sangat banyak pekerjaan untuk mengerjakannya dengan baik, itulah sebabnya kita memakai kerangka kerja yang dibangun dan diuji oleh ribuan orang.

2.7 Memakainya Sungguhan

2.7.1 Membaca kerangka kerja apa pun lewat empat fondasi

Keempat fondasi adalah panduan membaca untuk kerangka kerja web mana pun yang Anda jumpai, termasuk yang tidak diajarkan buku ini. Ketika berhadapan dengan dokumentasi baru, ajukan empat pertanyaan. *Di mana tabel rutenya, dan bagaimana parameter serta batasan ditulis? Di mana layanan didaftarkan, dengan masa hidup apa, dan bagaimana sebuah kelas menerimanya? Bagaimana tabel dipetakan ke kelas, dan bagaimana kueri ditulis? Apa nama lubang yang di-*escape*, lubang mentah, gelung, dan layout-nya?* Jawab keempatnya, dan Anda dapat membaca sebagian besar contoh kode kerangka kerja itu pada hari pertama. **Tabel 2.2** adalah lembar jawaban untuk ketiga kerangka kerja kita.

2.7.2 Profitina dalam praktik

Aplikasi FastAPI milik Profitina menjawab keempat pertanyaan itu dengan caranya sendiri. Rutenya adalah fungsi Python yang diberi dekorator berisi path, `@router.get("/api/me/dashboard")`, dengan parameter dan tipe yang diambil dari tanda tangan fungsinya. Injeksi dependensinya adalah penanda `Depends(...)` pada sebuah parameter: *handler* yang membangun dasbor menyatakan bahwa ia memerlukan pengguna yang sedang masuk dan sebuah sesi basis data, dan FastAPI menyediakan keduanya untuk setiap permintaan, sesinya dengan masa hidup per permintaan. Akses datanya melalui lapisan ORM yang menulis SQL berparameter untuk PostgreSQL di produksi dan SQLite di sebagian rangkaian pengujian, dan sebuah penjaga di rangkaian pengujian menjaga agar kedua dialek SQL itu tetap jujur. Halaman HTML-nya memakai mesin templat dengan *escaping* sebagai bawaan. Tidak ada yang eksotis di sini; ini Subbab 2.2 dengan nama-nama Python.

2.7.3 Merencanakan fitur baru dengan empat pertanyaan

Keempat pertanyaan itu juga berguna ke arah sebaliknya, untuk *merencanakan*. Ketika Bu Wati meminta Dimas membuat halaman yang mendaftar ketiga gerai beserta alamatnya, Dimas tidak memulai dengan mengetik kode. Ia menulis empat baris pendek di kertas, satu untuk setiap fondasi. *Rute*: `GET /outlets` untuk daftar dan `GET /outlets/{id}` untuk satu gerai, dengan id berbentuk seperti `O01`. *Dependensi*: controller memerlukan sesuatu yang mengenal gerai-gerai; itu cukup berupa metode baru pada katalog yang sudah ada, bukan kelas baru, jadi tidak perlu pendaftaran baru. *Data*: tabel `Outlet` punya kolom `id`, `name`, `address`, `phone`, dan `opened`, sehingga modelnya adalah *record* dengan lima properti bernama sama, dan kuerinya hanya “semua gerai”

atau "gerai dengan id ini". *Templat*: halaman daftar yang mengulang setiap gerai dan halaman rincian, keduanya di dalam *layout* yang sudah ada, dengan setiap nilai di lubang yang di-*escape*.

Dengan keempat baris itu, pekerjaan di kerangka kerja mana pun tinggal soal ejaan. Di ASP.NET Core rencana itu menjadi `OutletsController` pada Bab 3, yang dicapai melalui rute konvensional; di Laravel rencana itu menjadi dua baris di `routes/web.php`, sebuah model Eloquent `Outlet`, dan dua view Blade; di Spring menjadi sebuah *entity*, antarmuka *repository* tanpa metode sendiri, sebuah controller, dan dua templat Thymeleaf. Rencananya tidak berubah di antara ketiganya; hanya nama berkasnya yang berubah. Merencanakan per fondasi lebih dulu juga memunculkan pertanyaan sejak awal. Haruskah gerai yang sudah tutup tetap didaftar? Itu pertanyaan data dan tempatnya di kueri, bukan di templat. Haruskah nomor telepon menjadi tautan `tel:` yang dapat diklik? Itu pertanyaan templat dan tidak mengubah apa pun yang lain. Menjaga setiap keputusan di fondasinya sendiri adalah aturan Bu Wati dari Bab 1, "satu perubahan di satu tempat", yang diterapkan pada perancangan.

2.8 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Mendaftarkan rute dengan urutan yang salah pada router yang peka urutan. Di MiniFrame dan di Laravel, rute umum yang didaftarkan sebelum rute khusus akan menelan permintaan milik rute khusus itu (Soal 1). Daftarkan rute khusus lebih dulu, atau tambahkan batasan agar rute umum tidak dapat cocok.

Membuat dependensi dengan `new` di dalam kelas yang dibangun container. Cara itu berjalan, dan diam-diam menggagalkan container: kelas itu tidak dapat lagi diberi implementasi lain atau tiruan untuk pengujian. Deklarasikan dependensinya di konstruktor.

Memilih masa hidup yang salah. *Singleton* yang menyimpan data per permintaan (pengguna yang sedang masuk, keranjang belanja) mencampur data seorang pengunjung ke halaman pengunjung lain. *Transient* yang membuka sumber daya mahal pada setiap pemakaian memboroskannya. Tanyakan pada setiap pendaftaran: *bolehkah dua permintaan berbagi objek ini?*

Menyusun SQL dengan menyambung nilai. Bahkan dengan ORM, kebanyakan punya pintu darurat untuk SQL mentah (`FromSqlRaw`, `DB::raw`, `@Query(nativeQuery = true)`). Jangan pernah menempelkan nilai dari permintaan ke string semacam itu; pakailah mekanisme parameter yang disediakan ORM [16].

Mematikan *escaping* untuk "memperbaiki" masalah tampilan. Ketika `&` muncul di halaman, penyebabnya biasanya teks yang di-*escape* dua kali, bukan *escaping* itu sendiri. Temukan di mana *escaping* kedua terjadi; jangan meraih lubang mentah.

Mengira sintaksis sebagai gagasannya. Pemula sering mempelajari `@GetMapping`, `Route::get`, dan `[HttpGet]` sebagai tiga fakta yang tidak berhubungan. Pelajari gagasannya sekali, dan setiap sintaksis menjadi sekadar rincian.

2.9 Ringkasan Bab

- Di balik sintaksisnya yang berbeda, ASP.NET Core, Laravel, dan Spring Boot berbagi Model–View–Controller dan empat fondasi: perutean, injeksi dependensi, pemetaan objek-relasional, dan templat.

- Router mencocokkan metode dan path sebuah permintaan dengan templat yang tersusun dari segmen literal dan parameter, menerapkan batasan, dan meneruskan nilai hasil ekstraksi ke satu aksi. Router sungguhan mengurutkan templat berdasarkan prioritas alih-alih mencobanya berurutan.
- Container DI membangun objek dari pendaftaran, me-*resolve* parameter konstruktor secara rekursif. Masa hidup *singleton*, *scoped*, dan *transient* menyatakan berapa lama sebuah objek hidup dan siapa yang berbagi. Bergantung pada antarmuka membuat implementasi dapat diganti.
- ORM memetakan baris ke objek dengan mencocokkan nama, dan mengubah kueri yang ditulis dalam bahasa pemrograman menjadi SQL, melalui pohon ekspresi (EF Core, MiniFrame), pembangun kueri berantai (Eloquent), atau nama metode (Spring Data). ORM sungguhan selalu mengirim nilai sebagai parameter.
- Mesin templat mengisi lubang yang di-*escape*, mengulang blok untuk daftar, dan menyisipkan halaman ke *layout* bersama. *Escaping* adalah bawaan; keluaran mentah adalah pengecualian yang harus dapat dipertanggungjawabkan.
- MiniFrame mengimplementasikan keempatnya dalam 226 baris. Apa yang ditinggalkannya (parameter di SQL, prioritas, masa hidup *scoped*, konkurensi, templat terkompilasi, migrasi) memperlihatkan mengapa kerangka kerja sungguhan berukuran besar.

2.10 Soal Latihan

1. Tambahkan rute `GET /services/cheapest` ke MiniFrame yang menjawab dengan teks polos `cheapest`. Mula-mula daftarkan rute itu *sesudah* rute `/services/{id}` dan jalankan demo dengan alamat ini; lalu daftarkan *sebelum* rute tersebut. Jelaskan kedua hasilnya. Bagaimana ASP.NET Core akan berperilaku?

2. SQL apa yang dicetak pemeta MiniFrame untuk syarat `s => s.Price > 10000 && s.Price < 40000`, dan layanan mana yang dipilih syarat itu?

3. Hapus pendaftaran `IServiceRepository` dari `Program.cs` dan minta sebuah `ServicesController`. Apa yang terjadi, dan di baris mana pada **Kode 2.2**?

4. Di `templates/services.html`, ubah `{{ description }}` menjadi `{{{ description }}}` dan buka `/services`. Apa yang berubah pada HTML baris pertama? Kapan perbedaan itu menjadi berbahaya?

5. Demo mencetak `[container] built ServicesController` untuk setiap permintaan tetapi `[container] built CsvServiceRepository` hanya sekali. Jelaskan dari **Kode 2.2** dan **Kode 2.8**. Apa yang akan salah jika *repository* didaftarkan sebagai *transient* dan membaca berkas besar di konstruktornya?

6. Tulis kueri "layanan berharga antara Rp 10.000 dan Rp 40.000" dalam Eloquent dan sebagai nama metode Spring Data, dengan memakai dokumentasi masing-masing [12], [6].

2.11 Penyelesaian Soal Latihan

1. Jika didaftarkan sesudah `/services/{id}`, rute baru itu tidak pernah tercapai: router mencoba `/services/{id}` lebih dulu, rute itu cocok (dua segmen, dan `{id}` menerima apa saja), dan demo mencetak `handler { id=cheapest }`, SQL `WHERE id = 'cheapest'`, serta halaman 404 "There is no service with the code cheapest." Jika didaftarkan sebelumnya, router menjumpai templat literal lebih dulu dan demo mencetak `handler {  }` dan `RESPONSE 200 text/plain` dengan teks

`cheapest`. Kedua hasil berasal dari menjalankan program yang diubah itu. ASP.NET Core tidak bergantung pada urutan: segmen literal punya prioritas lebih tinggi daripada parameter, sehingga `/services/cheapest` akan mencapai rute literal pada kedua kasus [1].

2. Pemeta mencetak `SELECT * FROM service WHERE price > 10000 AND price < 40000`: simpul `&&` menjadi `AND`, dan setiap perbandingan menjadi kolom, operator, dan literal. Sebuah percobaan memilih S0002 (12.000), S0004 (20.000), S0008 (35.000), S0009 (25.000), S0010 (15.000), dan S0012 (30.000). S0001 dan S0003 terlalu murah, sedangkan S0005, S0006, S0007, dan S0011 terlalu mahal; S0007 berharga tepat 40.000, yang dikecualikan oleh `<`.

3. `Resolve` melempar `InvalidOperationException: Nothing registered for IServiceRepository.` Hal ini terjadi di dalam pemanggilan rekursif untuk parameter konstruktor pertama controller: `map.TryGetValue` gagal untuk `IServiceRepository` dan `throw` di awal `Resolve` dijalankan. Container sungguhan memberi pesan yang sejenis, menyebut tipe yang tidak dapat di-*resolve* dan kelas yang membutuhkannya; membaca pesan itu adalah cara tercepat menemukan pendaftaran yang hilang.

4. Deskripsi baris pertama berubah dari `Wash, dry & fold (per kg)` menjadi `Wash, dry & fold (per kg)`. Peramban kebetulan menampilkan keduanya sama, karena `&` tunggal yang diikuti spasi ditoleransi. Perbedaannya menjadi berbahaya begitu sebuah deskripsi mengandung *markup*: jika pemilik (atau siapa pun yang punya akses ke data) menyimpan deskripsi berisi `<script>`, lubang mentah akan mengirimkannya ke setiap pengunjung sebagai kode. Lubang yang di-*escape* adalah bawaan yang aman; lubang mentah hanya untuk HTML yang dihasilkan program sendiri, seperti *body* halaman di dalam *layout*.

5. Controller didaftarkan dengan `Transient`, sehingga `Resolve` tidak pernah menyimpannya di `singles` dan membangun yang baru untuk setiap permintaan. *Repository* didaftarkan dengan `Singleton`; `Resolve` yang pertama menyimpannya, dan permintaan berikutnya menemukannya di `singles` lalu mengembalikannya tanpa membangun. Jika *repository* bersifat *transient* dan membaca berkas besar di konstruktornya, setiap permintaan akan membaca berkas itu lagi, sehingga setiap halaman menjadi lambat; dan karena setiap permintaan mendapat salinannya sendiri, perubahan yang dibuat lewat satu salinan (misalnya pembaruan harga yang disimpan di memori) tidak akan terlihat oleh yang lain.

6. Dalam Eloquent, `Service::whereBetween('price', [10000, 40000])->get();` `whereBetween` mencakup kedua ujungnya, seperti `BETWEEN` di SQL [12]. Dalam Spring Data, metode `List<Service> findByPriceBetween(BigDecimal min, BigDecimal max);` di `ServiceRepository`; kata kunci `Between` pada namanya menurunkan `where s.price between ?1 and ?2` [6]. Keduanya tidak dijalankan di mesin *build*; Bab 7 dan 11 menjalankan kueri semacam itu terhadap basis data sungguhan.

Daftar Pustaka

[1] Microsoft, “Routing in ASP.NET Core,” *ASP.NET Core Documentation*, for ASP.NET Core 10.0, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/fundamentals/routing

[2] M. Fowler, “Inversion of control containers and the dependency injection pattern,” martinfowler.com, Jan. 2004. [Online]. Available: https://martinfowler.com/articles/injection.html

[3] Microsoft, “Expression trees,” *C# Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/dotnet/csharp/advanced-topics/expression-trees/

[4] Microsoft, "Entity Framework Core overview," *EF Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/ef/core/

[5] Laravel, "Routing," *Laravel 13.x Documentation*, 2026. [Online]. Available: https://laravel.com/docs/13.x/routing

[6] Spring, "JPA query methods," *Spring Data JPA Reference Documentation*, accessed Sep. 2026. [Online]. Available: https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html

[7] Spring, "Mapping requests," *Spring Framework Reference Documentation: Web MVC*, accessed Sep. 2026. [Online]. Available: https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-controller/ann-requestmapping.html

[8] Microsoft, "Dependency injection in ASP.NET Core," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/fundamentals/dependency-injection

[9] Laravel, "Service container," *Laravel 13.x Documentation*, 2026. [Online]. Available: https://laravel.com/docs/13.x/container

[10] Spring, "Dependency injection," *Spring Framework Reference Documentation: The IoC Container*, accessed Sep. 2026. [Online]. Available: https://docs.spring.io/spring-framework/reference/core/beans/dependencies/factory-collaborators.html

[11] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley, 2002.

[12] Laravel, "Eloquent: getting started," *Laravel 13.x Documentation*, 2026. [Online]. Available: https://laravel.com/docs/13.x/eloquent

[13] Laravel, "Blade templates," *Laravel 13.x Documentation*, 2026. [Online]. Available: https://laravel.com/docs/13.x/blade

[14] The Thymeleaf Team, "Tutorial: Using Thymeleaf," version 3.1, accessed Sep. 2026. [Online]. Available: https://www.thymeleaf.org/doc/tutorials/3.1/usingthymeleaf.html

[15] Microsoft, "Razor syntax reference for ASP.NET Core," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/mvc/views/razor

[16] OWASP Foundation, "SQL injection prevention cheat sheet," *OWASP Cheat Sheet Series*, accessed Sep. 2026. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Bab 3

ASP.NET Core MVC I: Proyek, Pipeline, Perutean, Controller, dan View Razor

Bab ini membangun situs web publik Profitina Laundry dengan ASP.NET Core MVC: halaman beranda, daftar layanan beserta harganya, satu halaman untuk setiap layanan, layanan sampai harga tertentu, formulir penawaran harga yang menerapkan aturan toko yang sesungguhnya, satu halaman untuk setiap gerai, halaman "tentang kami", halaman galat yang ramah, dan *endpoint* JSON kecil untuk program lain. Sepanjang jalan, bab ini membongkar proyek ASP.NET Core yang baru, berkas demi berkas. Pada akhir bab ini Anda akan mampu membuat proyek MVC dengan perintah `dotnet` dan menjelaskan setiap berkas di dalamnya; menjelaskan *pipeline* permintaan dan menulis *middleware* sendiri; mendaftarkan layanan di container injeksi dependensi dan menerimanya di controller maupun view; merutekan permintaan dengan rute konvensional dan dengan *attribute route*, termasuk batasannya; menulis aksi controller yang mengembalikan view, JSON, pengalihan (*redirect*), dan kode status; mengikat formulir ke objek C# dan melaporkan kesalahan masukan; serta menulis view Razor dengan *layout*, *partial view*, *section*, dan *tag helper*. Semua yang ada di bab ini dibangun dan dijalankan di mesin yang menghasilkan buku ini; setiap keluaran yang ditampilkan adalah keluaran sesungguhnya, dan tangkapan layarnya diambil dari aplikasi yang sedang berjalan.

3.1 Situs Web Bu Wati

Bab 1 meninggalkan Bu Wati dengan daftar harga yang dapat dibaca program lain sebagai JSON, dan Bab 2 membuka mesin yang dipakai kerangka kerja. Kini ia menginginkan apa yang benar-benar akan dilihat pelanggannya. Ia menuliskan daftar keinginannya di balik selembar nota pesanan:

- **halaman beranda** yang menyebut toko apa ini dan menampilkan tiga layanan yang paling sering diminta;
- **daftar layanan** beserta harganya, dan **satu halaman untuk setiap layanan**;
- cara cepat melihat **layanan sampai harga tertentu**, bagi pelanggan yang anggarannya terbatas;
- **penawaran harga**: pilih layanan, ketik berat atau jumlah barang, centang "antar-jemput", dan lihat berapa biayanya, *dengan aturan toko*: minimal 3 kg untuk layanan yang dihitung per berat, barang dihitung utuh, antar-jemput Rp 10.000 tetapi gratis mulai Rp 100.000;
- **gerai-gerai**, masing-masing dengan alamat dan nomor teleponnya;
- halaman **tentang kami** dengan jam buka;
- tidak ada halaman galat yang jelek: alamat yang salah ketik harus mendapat halaman yang sopan dengan gaya toko;
- dan, untuk Dimas yang memimpikan aplikasi seluler, data layanan dalam bentuk **JSON**.

Bu Wati juga punya dua aturan sendiri, keduanya dipelajari dengan susah payah dari situs lama. Aturan harga harus tinggal di satu tempat, sehingga pada hari minimalnya berubah menjadi 2,5 kg hanya satu baris yang berubah. Dan situs harus tampil sama di setiap halaman tanpa ada yang menyalin bagian kepalanya. Bab 1 menyebut aturan pertama "jaga controller tetap ramping" dan

Bab 2 memperlihatkan bahwa aturan kedua adalah gunanya *layout*. Bab ini membangun semuanya dengan ASP.NET Core 10, tanpa apa pun selain *shared framework* yang datang bersama SDK. Datanya sama dengan sebelumnya, ekspor CSV (`data/Service.csv` dan `data/Outlet.csv`); Bab 5 memindahkannya ke basis data.

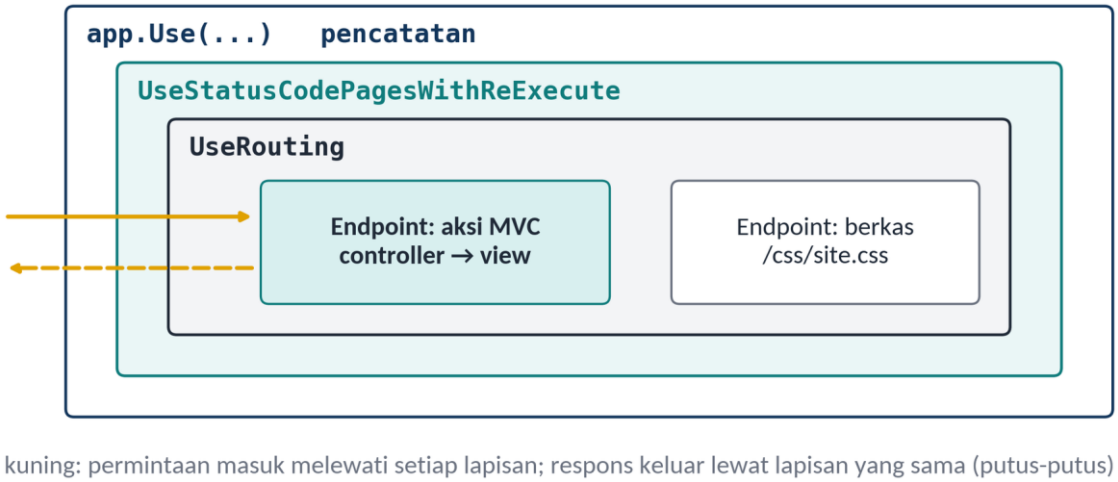
3.2 Gagasannya dalam Kata-Kata Sederhana

3.2.1 *Host* dan *pipeline*

Aplikasi ASP.NET Core adalah program *console* yang membangun sebuah **host**, yaitu objek yang memiliki server web (Kestrel), konfigurasi, pencatatan, dan container injeksi dependensi, lalu menjalankannya. Setiap permintaan yang diterima Kestrel diubah menjadi objek `HttpContext`, yang memuat permintaan dan respons yang sedang dibangun, lalu dilewatkan melalui **pipeline permintaan**.

> **DEFINISI. Pipeline permintaan** (*request pipeline*) adalah rantai berurutan komponen **middleware**. Setiap *middleware* menerima `HttpContext` dan rujukan ke *middleware berikutnya*; ia boleh mengerjakan sesuatu sebelum memanggil yang berikutnya, memanggilnya (atau tidak), dan mengerjakan sesuatu setelah pemanggilan itu kembali. Unsur terakhir *pipeline* adalah **endpoint**: kode yang dipilih untuk menghasilkan respons, misalnya aksi controller atau berkas statis.

Gambar 3.1 memperlihatkan *pipeline* aplikasi bab ini sebagai lapisan-lapisan bersarang. Permintaan masuk melalui setiap lapisan dari luar ke dalam, mencapai sebuah *endpoint*, lalu respons keluar lagi melalui lapisan yang sama dalam urutan terbalik. Lapisan terluar mencatat setiap permintaan beserta status dan durasinya, yang hanya dapat ia ketahui pada perjalanan *keluar*. Lapisan berikutnya, halaman kode status, memperhatikan respons seperti 404 pada perjalanan keluar dan menggantinya dengan halaman yang ramah. Lapisan perutean memilih *endpoint*. Karena *middleware* membungkus *middleware*, **urutan itu penting**: sebuah lapisan hanya dapat melihat dan mengubah apa yang melewatinya [1].

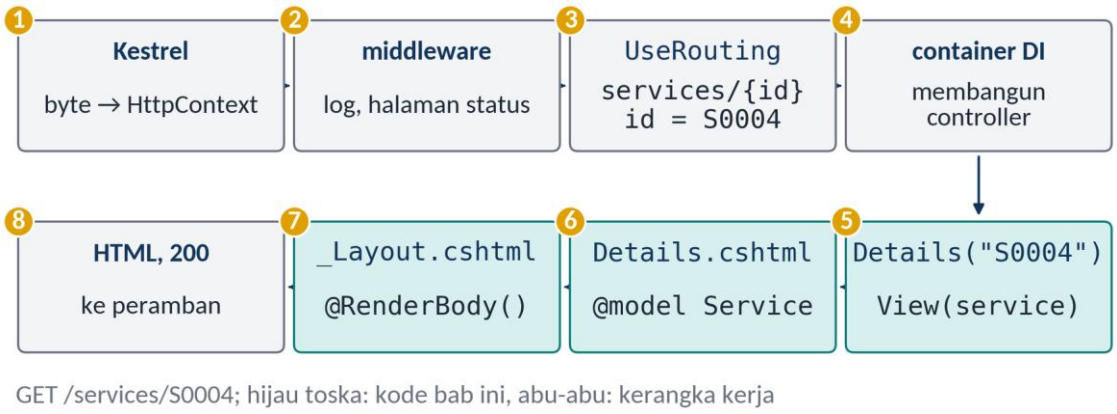


Gambar 3.1 *Pipeline* permintaan LaundryMvc: setiap permintaan melewati *middleware* pencatatan, kode status, dan perutean menuju *endpoint*, lalu respons kembali melalui lapisan yang sama.

3.2.2 Satu permintaan melalui MVC

Gambar 3.2 mengikuti satu permintaan, `GET /services/S0004`, melalui semua bagian yang ditulis bab ini. Kestrel mengubah byte menjadi `HttpContext` (1); *middleware* pada **Gambar 3.1** meloloskannya (2); perutean mencocokkan templat `services/{id}` dan mencatat `id = "S0004"`

(3); container injeksi dependensi membangun `ServicesController`, sambil menyerahkan katalog dan kalkulator harga (4); kerangka kerja memanggil aksi `Details("S0004")`, yang mengembalikan `View(service)` (5); Razor merender `Details.cshtml` dengan layanan itu sebagai modelnya (6) di dalam `_Layout.cshtml` (7); dan HTML-nya kembali dengan status 200 (8). Hanya langkah 5, 6, dan 7 yang merupakan kode bab ini [2]; sisanya adalah kerangka kerja, yang mengerjakan apa yang dikerjakan MiniFrame di Bab 2, dengan kekuatan industri.



GET /services/S0004; hijau toska: kode bab ini, abu-abu: kerangka kerja

Gambar 3.2 Satu permintaan, `GET /services/S0004`, melalui LaundryMvc: dari byte ke `HttpContext`, melalui *middleware* dan perutean, ke controller yang dibangun container, sebuah view, dan *layout*.

3.3 Membuat Proyeknya

3.3.1 Templat MVC

Bab 1 membuat proyek dengan `dotnet new mvc -o FirstMvc`. Kita memulai dengan cara yang sama, menamai proyeknya `LaundryMvc`:

```
dotnet new mvc -o LaundryMvc
cd LaundryMvc
```

Templat itu membuat berkas-berkas pada **Tabel 3.1**. Dua folder, `bin` dan `obj`, muncul setelah *build* pertama dan menyimpan hasil kompilasi; keduanya tidak pernah disunting dan tidak pernah dimasukkan ke kendali versi.

Tabel 3.1 Berkas-berkas yang dibuat `dotnet new mvc`, dan guna masing-masing.

Berkas atau folder	Gunanya
`LaundryMvc.csproj`	Proyeknya: SDK web, target `net10.0`, pemeriksaan *nullable*, `using` implisit
`Program.cs`	Membangun *host*: mendaftarkan layanan, lalu membangun *pipeline*, lalu berjalan
`appsettings.json`, `appsettings.Development.json`	Konfigurasi, seperti tingkat log; yang kedua menimpa yang pertama di Development
`Properties/launchSettings.json`	Cara `dotnet run` dan VS Code menjalankan aplikasi: alamat, lingkungan
`Controllers/HomeController.cs`	Controller pertama dengan aksi `Index`, `Privacy`, dan `Error`
`Models/ErrorViewModel.cs`	Model untuk halaman galat templat
`Views/Home/Index.cshtml`, `Privacy.cshtml`	View milik `HomeController`
`Views/Shared/_Layout.cshtml`	*Layout*: bingkai di sekeliling setiap halaman
`Views/Shared/Error.cshtml`, `_ValidationScriptsPartial.cshtml`	View bersama
`Views/_ViewImports.cshtml`, `_ViewStart.cshtml`	Pengaturan untuk setiap view di folder itu: *namespace*, *tag helper*, *layout*
`wwwroot/`	Berkas statis yang dilayani apa adanya: `css/`, `js/`, `favicon.ico`, dan `lib/` berisi Bootstrap dan jQuery (9,5 MB)

Nama-nama folder itu bukan hiasan; itulah **konvensi** yang disebut Bab 1. Kelas di `Controllers` yang namanya berakhiran `Controller` adalah controller. View untuk aksi `Details` milik `ServicesController` dicari lebih dulu di `Views/Services/Details.cshtml` lalu di `Views/Shared/Details.cshtml`. Berkas yang diawali garis bawah bukan halaman tersendiri, melainkan potongan yang dipakai halaman lain. Mempelajari konvensi ini adalah sebagian besar dari mempelajari MVC.

3.3.2 Yang kita ubah dari templat

LaundryMvc mempertahankan struktur templat tetapi melakukan empat perubahan, masing-masing dengan alasannya.

- **Tanpa Bootstrap, tanpa jQuery.** Templat menyalin 9,5 MB pustaka pihak ketiga ke `wwwroot/lib`. Situs Bu Wati cukup memerlukan satu lembar gaya kecil, `wwwroot/css/site.css`, kurang dari empat puluh baris dengan warna-warna rumah, sehingga `lib/` dihapus dan *layout* hanya menautkan CSS kita. Tanpa jQuery, skrip validasi sisi klien milik templat tidak berlaku; server tetap memeriksa setiap masukan (Subbab 3.6.4), yang memang harus dilakukannya dalam keadaan apa pun.
- **Tanpa pengalihan HTTPS selama pengembangan.** *Pipeline* templat diawali `app.UseHttpsRedirection()`, yang mengalihkan setiap permintaan `http://` ke `https://`. Di mesin pengembangan tanpa sertifikat tepercaya, hal itu hanya menghasilkan peringatan, sehingga LaundryMvc melayani HTTP biasa di `http://localhost:4030`; Bab 6 men-*deploy*-nya di balik HTTPS.
- **Port tetap.** `Properties/launchSettings.json` dikurangi menjadi satu profil, `http`, dengan alamat `http://localhost:4030` dan lingkungan `Development`, seperti pada **Kode 3.1**.

- **Controller, model, dan view kita sendiri**, menggantikan `Privacy` dan `ErrorViewModel`, seperti diuraikan sisa bab ini.

Kode 3.1 `Properties/launchSettings.json`: `dotnet run` menjalankan aplikasi di `http://localhost:4030` dalam lingkungan Development.

```
{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "applicationUrl": "http://localhost:4030",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

Kode 3.2 adalah berkas konfigurasinya. Berkas ini mengatur tingkat log pesan-pesan kerangka kerja sendiri (kategori yang diawali `Microsoft.AspNetCore`) ke `Warning`, sehingga konsol menampilkan baris-baris awal jalan dan log permintaan milik kita, tetapi tidak pesan rinci kerangka kerja untuk setiap permintaan. `AllowedHosts` membatasi nama *host* yang dijawab aplikasi; `*` berarti apa saja, dan itu tidak masalah di balik *proxy* yang memeriksanya.

Kode 3.2 `appsettings.json`: tingkat log dan *host* yang diizinkan.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Berkas proyeknya persis berkas *minimal API* di Bab 1 (Subbab 1.6.1) dengan nama lain: `Sdk="Microsoft.NET.Sdk.Web"`, `net10.0`, `Nullable`, dan `ImplicitUsings`. `Nullable` membuat kompilator membedakan `string` (tidak pernah *null*) dari `string?` (boleh *null*) dan memberi peringatan ketika *null* bisa lolos, yang akan Anda lihat dari tanda `?` di sepanjang kode bab ini.

3.4 `Program.cs`: Layanan, Lalu *Pipeline*

Kode 3.3 adalah seluruh kode awal jalan LaundryMvc. Kode ini terdiri atas dua bagian, dan urutan kedua bagian itu adalah urutan setiap aplikasi ASP.NET Core: pertama daftarkan apa yang *dimiliki* aplikasi (layanan), lalu uraikan apa yang *dilakukan*nya terhadap setiap permintaan (*pipeline*).

Kode 3.3 `Program.cs`: layanan didaftarkan pada *builder*, lalu *pipeline* dibangun pada aplikasi.

```
using System.Diagnostics;
using LaundryMvc.Data;
using LaundryMvc.Models;

var builder = WebApplication.CreateBuilder(args);

// 1. Services: everything the application needs, registered once.
builder.Services.AddControllersWithViews();
builder.Services.AddSingleton<ILaundryCatalog>(_ => new CsvCatalog(
    Path.Combine(builder.Environment.ContentRootPath, "data")));
builder.Services.AddSingleton<PriceCalculator>();

var app = builder.Build();

// 2. Pipeline: what happens to every request, in this order.
app.Use(async (context, next) =>
{
    var clock = Stopwatch.StartNew();
    await next(context);                  // everything later runs in here
    app.Logger.LogInformation("{Method} {Path}{Query} -> {Status} in {Ms} ms",
        context.Request.Method, context.Request.Path,
        context.Request.QueryString, context.Response.StatusCode,
        clock.ElapsedMilliseconds);
});

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/error/500");
}
app.UseStatusCodePagesWithReExecute("/error/{0}");

app.UseRouting();

app.MapStaticAssets();
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}")
    .WithStaticAssets();

app.Run();
```

Builder. `WebApplication.CreateBuilder(args)` menyiapkan *host*: ia membaca `appsettings.json` dan pengaturan lingkungannya, menyiapkan pencatatan ke konsol, dan membuat *service collection*, yaitu daftar pendaftaran untuk container. `builder.Environment.ContentRootPath` adalah folder tempat aplikasi dijalankan, yaitu folder proyek ketika Anda memakai `dotnet run`.

Layanan. `AddControllersWithViews()` mendaftarkan semua yang diperlukan MVC: penemuan controller, *model binding*, kompilasi dan perenderan view Razor, *tag helper*, token *antiforgery*. Dua baris berikutnya mendaftarkan layanan kita sendiri, dan keduanya adalah `Container` milik MiniFrame dalam bentuk sungguhannya [3]. `AddSingleton<ILaundryCatalog>(_ => new CsvCatalog(...))` berarti: "kapan pun ada yang meminta `ILaundryCatalog`, berikan satu `CsvCatalog` itu, yang dibangun sekali oleh fungsi ini". Fungsi diperlukan karena konstruktornya menerima nama folder, sebuah string yang tidak dapat ditebak container. `AddSingleton<PriceCalculator>()` mendaftarkan aturan harga sebagai kelas yang dapat dibangun container sendiri, karena konstruktornya tidak menerima apa-apa. Keduanya *singleton* karena keduanya aman dipakai bersama: katalog dibaca sekali dan tidak pernah berubah, dan kalkulator sama sekali tidak menyimpan keadaan.

Pipeline. `builder.Build()` membuat aplikasinya, dan setiap panggilan `app.Use…` menambahkan satu *middleware*, sesuai urutan kemunculannya dalam kode.

- *Middleware* pertama ditulis langsung dengan `app.Use(async (context, next) => …)`. Ia menyalakan *stopwatch*, memanggil `await next(context)`, yang menjalankan *seluruh sisa pipeline*, dan setelah pemanggilan itu kembali, ketika kode status sudah diketahui, ia mencatat satu baris: metode, path, *query string*, status, dan milidetik. Segala yang dikerjakan *middleware* setelah `await next` terjadi pada perjalanan keluar.
- `UseExceptionHandler("/error/500")` hanya ditambahkan di luar Development. Jika komponen yang lebih belakang melempar *exception*, ia menangkapnya, mencatatnya, dan menjalankan ulang permintaan sebagai `/error/500`, sehingga pengunjung melihat halaman galat kita. Di Development, ASP.NET Core justru menampilkan *developer exception page* yang rinci beserta *stack trace*-nya [4], dan itulah yang diinginkan pemrogram serta yang tidak boleh sekali pun dilihat pelanggan.
- `UseStatusCodePagesWithReExecute("/error/{0}")` mengawasi perjalanan keluar. Jika sebuah respons berstatus antara 400 dan 599 dan *tidak berbadan*, ia menjalankan ulang *pipeline* dengan path `/error/404` (`{0}` menjadi kode statusnya), sehingga 404 yang kosong menjadi halaman ramah kita, tetap dengan status 404 [5].
- `UseRouting()` mencocokkan permintaan dengan semua *endpoint* dan menyimpan yang terpilih di `HttpContext`. *Middleware* yang diletakkan sesudahnya dapat menanyakan *endpoint* mana yang terpilih; Bab 4 memakai hal itu.
- `MapStaticAssets()` menambahkan *endpoint* untuk berkas-berkas di `wwwroot`, dengan kompresi dan header *cache*, berdasarkan manifes yang dihasilkan saat *build*. `.WithStaticAssets()` pada rute controller memungkinkan view menautkan berkas-berkas itu dengan sidik (*fingerprint*) di URL-nya, yang akan kita lihat di *layout* [6].
- `MapControllerRoute` menambahkan **rute konvensional**, yang dibahas di Subbab 3.6.1, dan `app.Run()` menjalankan Kestrel dan menunggu sampai aplikasi berhenti.

Di mana *endpoint* milik controller sendiri dijalankan? Sesudah *middleware* terakhir: ASP.NET Core menambahkan *middleware* pelaksana *endpoint* di ujung *pipeline* secara otomatis. Jadi urutan kita adalah: catat, (tangani *exception*), halaman status, perutean, *endpoint*. **Gambar 3.1** adalah daftar ini yang digambar sebagai lapisan.

3.5 Model: Data dan Aturan

"Model" dalam MVC berarti segala sesuatu yang berkaitan dengan bisnis dan tidak berkaitan dengan HTTP: data, aturan, dan kode yang mengambil data. LaundryMvc menyimpan kelas data dan aturan di `Models/` dan akses data di `Data/`.

3.5.1 Layanan, gerai, dan uang

Kode 3.4 adalah layanan. Ini sebuah **record** C#, kelas yang identitasnya adalah nilai-nilainya, dideklarasikan dengan *primary constructor*: ketiga parameternya menjadi tiga properti hanya-baca, `Id`, `Description`, dan `Price`.

Kode 3.4 `Models/Service.cs`: sebuah layanan dengan kode, deskripsi, dan harganya, serta tiga properti yang diturunkan dari deskripsinya.

```
namespace LaundryMvc.Models;

// One row of data/Service.csv: something the laundry sells, and its price.
public record Service(string Id, string Description, decimal Price)
{
    // "Wash, dry & fold (per kg)" -> "Wash, dry & fold"
    public string Name => Description.Contains(" (")
        ? Description[..Description.LastIndexOf(" (")] : Description;

    public bool PerKg => Description.EndsWith("(per kg)");

    public string Unit => PerKg ? "kg"
        : Description.EndsWith("(per pair)") ? "pair" : "item";
}
```

Ketiga properti terhitung itu mengubah deskripsi dari basis data menjadi apa yang diperlukan halaman. `Name` memotong satuan dalam kurung, sehingga `"Wash, dry & fold (per kg)"` menjadi `"Wash, dry & fold"`; `PerKg` menyatakan apakah layanan dihitung per berat; `Unit` memberi `kg`, `pair`, atau `item`. Ketiganya dihitung setiap kali dibaca, sehingga tidak mungkin bertentangan dengan deskripsinya. **Kode 3.5** adalah gerai, sebuah *record* dengan lima properti; `DateOnly` menyimpan tanggal tanpa waktu.

Kode 3.5 `Models/Outlet.cs`: sebuah gerai Profitina Laundry.

```
namespace LaundryMvc.Models;

// One row of data/Outlet.csv: a shop where customers bring their laundry.
public record Outlet(string Id, string Name, string Address, string Phone,
    DateOnly Opened);
```

Harga di Indonesia ditulis dengan titik di antara ribuan, `Rp 36.000`. .NET dapat melakukannya dengan kultur `id-ID`, tetapi data kultur bergantung pada pemasangan sistem operasi, sehingga LaundryMvc memformat rupiah dengan kultur invarian lalu menukar koma dengan titik, di satu tempat, **Kode 3.6**.

Kode 3.6 `Models/Money.cs`: jumlah rupiah dengan gaya Indonesia, diformat di satu tempat.

```
using System.Globalization;

namespace LaundryMvc.Models;

public static class Money
{
    // 36000 -> "Rp 36.000" (Indonesian style: a dot between thousands)
    public static string Rp(decimal amount) =>
        "Rp " + amount.ToString("N0", CultureInfo.InvariantCulture)
            .Replace(",", ".");
}
```

3.5.2 Aturan harga

Kode 3.7 adalah tempat tinggal aturan Bu Wati, dan hanya di situ. `Calculate` menerima sebuah layanan, jumlah, dan pilihan antar-jemput, lalu mengembalikan `Quote`, sebuah *record* berisi semua yang perlu ditampilkan halaman: apa yang ditagihkan, subtotal, biaya antar-jemput, total, dan catatan yang menjelaskan setiap aturan yang diterapkan.

Kode 3.7 `Models/PriceCalculator.cs`: aturan harga toko, dengan penjelasan untuk setiap aturan yang diterapkan.

```
namespace LaundryMvc.Models;

public record Quote(Service Service, decimal Quantity, decimal Billed,
    decimal Subtotal, decimal PickupFee, decimal Total, List<string> Notes);

// The shop's pricing rules live here, not in a controller or a view.
public class PriceCalculator
{
    public const decimal MinimumKg = 3m;
    public const decimal PickupFee = 10_000m;
    public const decimal FreePickupFrom = 100_000m;

    public Quote Calculate(Service service, decimal quantity, bool pickup)
    {
        var notes = new List<string>();
        decimal billed = quantity;
        if (service.PerKg && quantity < MinimumKg)
        {
            billed = MinimumKg;
            notes.Add($"Minimum {MinimumKg} kg, so {quantity} kg is billed "
                + $"as {MinimumKg} kg.");
        }
        else if (!service.PerKg && quantity != Math.Ceiling(quantity))
        {
            billed = Math.Ceiling(quantity);
            notes.Add($"Items are counted whole: {quantity} becomes {billed}.");
        }
        decimal subtotal = billed * service.Price;
        decimal fee = 0m;
        if (pickup)
        {
            fee = subtotal >= FreePickupFrom ? 0m : PickupFee;
            notes.Add(fee == 0m
                ? $"Pickup and delivery are free from {Money.Rp(FreePickupFrom)}."
                : $"Pickup and delivery cost {Money.Rp(PickupFee)}.");
        }
        return new Quote(service, quantity, billed, subtotal, fee,
            subtotal + fee, notes);
    }
}
```

Kerjakan satu penawaran secara manual sebelum memercayai kodenya. Seorang pelanggan membawa 2,5 kg untuk “cuci, kering, lipat” (Rp 8.000 per kg) dan ingin antar-jemput. Layanannya per kg dan 2,5 lebih kecil daripada minimal 3, jadi `billed` menjadi 3 dan sebuah catatan menyatakannya. Subtotalnya $3 \times 8000 = 24000$. Antar-jemput diminta; 24.000 di bawah ambang 100.000, jadi biayanya 10.000 dan catatan kedua menyatakannya. Totalnya $24000 + 10000 = 34000$. Subbab 3.8 memperlihatkan bahwa situs yang berjalan memberi hasil persis ini.

Mengapa kelas tersendiri, bukan beberapa baris di dalam aksi controller? Karena aturan itulah bisnisnya. Aturan yang sama akan diperlukan oleh layar konter di Bab 5, oleh *endpoint* JSON untuk aplikasi Dimas, dan oleh pengujian otomatis; tak satu pun dapat memanggil aksi controller dengan mudah, dan semuanya dapat memanggil `PriceCalculator.Calculate`. Konstanta di bagian atas adalah satu-satunya tempat minimal, biaya, dan ambang itu ditulis, dan itulah aturan pertama Bu Wati dalam C#.

3.5.3 *View model* dan model formulir

View model adalah kelas yang dibentuk untuk satu view, berisi persis apa yang ditampilkan view itu. Halaman beranda menampilkan tiga layanan populer dan dua hitungan, jadi **Kode 3.8** adalah *record* berisi tiga hal itu dan tidak lebih.

Kode 3.8 `Models/HomeViewModel.cs`: persis apa yang ditampilkan halaman beranda.

```
namespace LaundryMvc.Models;

// Exactly what the home page shows, and nothing more.
public record HomeViewModel(IReadOnlyList<Service> Popular,
    int ServiceCount, int OutletCount);
```

Formulir penawaran memerlukan kelas yang dapat diisi *model binding* dari permintaan, **Kode 3.9**. Properti-propertinya punya *setter*, karena *binder* membuat objeknya lalu mengisinya satu per satu; `ServiceId` dan `Quantity` bertipe *nullable* agar "tidak diberikan" dapat dibedakan dari "diberikan nol". Properti `Result` diisi oleh controller, tidak pernah dari permintaan; atribut `[BindNever]` memberi tahu *binder* untuk tidak menyentuhnya, bahkan jika seseorang menambahkan `?Result=…` ke alamat.

Kode 3.9 `Models/QuoteForm.cs`: masukan formulir penawaran harga, diisi oleh *model binding*.

```
using Microsoft.AspNetCore.Mvc.ModelBinding;

namespace LaundryMvc.Models;

// Filled by model binding from the query string of /services/quote.
public class QuoteForm
{
    public string? ServiceId { get; set; }
    public decimal? Quantity { get; set; }
    public bool Pickup { get; set; }

    [BindNever]                          // set by the controller, never by a URL
    public Quote? Result { get; set; }
}
```

3.5.4 Katalog: akses data di balik antarmuka

Controller tidak boleh tahu dari mana datanya berasal. Controller bergantung pada antarmuka **Kode 3.10**; kini implementasinya membaca berkas CSV, di Bab 5 sebuah basis data.

Kode 3.10 `Data/ILaundryCatalog.cs`: apa yang boleh ditanyakan controller tentang layanan dan gerai.

```
using LaundryMvc.Models;

namespace LaundryMvc.Data;

// What the controllers may ask for. Chapter 5 swaps the CSV files
// behind it for a real database without touching a controller.
public interface ILaundryCatalog
{
    IReadOnlyList<Service> Services();
    Service? FindService(string id);
    IReadOnlyList<Outlet> Outlets();
    Outlet? FindOutlet(string id);
}
```

Kode 3.11 membaca kedua berkas sekali, di konstruktornya, ke dalam daftar. Karena didaftarkan sebagai *singleton*, hal itu terjadi sekali setiap kali aplikasi dijalankan. Metode-metode `Find` membandingkan kode tanpa membedakan huruf besar-kecil, sehingga `/services/s0004` menemukan `S0004`. Pembaca CSV kecilnya sama dengan di Bab 1, ditulis sebagai *iterator* (`yield return`) yang menghasilkan satu baris setiap kali.

Kode 3.11 `Data/CsvCatalog.cs`: katalog yang diimplementasikan di atas kedua berkas CSV, dibaca sekali saat awal jalan.

```
using System.Globalization;
using System.Text;
using LaundryMvc.Models;

namespace LaundryMvc.Data;

// Reads data/Service.csv and data/Outlet.csv once, at start-up.
public class CsvCatalog : ILaundryCatalog
{
    private readonly List<Service> services;
    private readonly List<Outlet> outlets;

    public CsvCatalog(string folder)
    {
        services = Read(Path.Combine(folder, "Service.csv"))
            .Select(f => new Service(f[0], f[1],
                decimal.Parse(f[2], CultureInfo.InvariantCulture)))
            .ToList();
        outlets = Read(Path.Combine(folder, "Outlet.csv"))
            .Select(f => new Outlet(f[0], f[1], f[2], f[3],
                DateOnly.Parse(f[4], CultureInfo.InvariantCulture)))
            .ToList();
    }

    public IReadOnlyList<Service> Services() => services;

    public Service? FindService(string id) => services.FirstOrDefault(
        s => s.Id.Equals(id, StringComparison.OrdinalIgnoreCase));

    public IReadOnlyList<Outlet> Outlets() => outlets;

    public Outlet? FindOutlet(string id) => outlets.FirstOrDefault(
        o => o.Id.Equals(id, StringComparison.OrdinalIgnoreCase));

    // Splits each line on commas, except commas inside "double quotes".
    private static IEnumerable<string[]> Read(string path)
    {
        foreach (var line in File.ReadLines(path).Skip(1))  // skip header
        {
            var fields = new List<string>();
            var field = new StringBuilder();
            bool quoted = false;
            foreach (char c in line)
            {
                if (c == '"') quoted = !quoted;
                else if (c == ',' && !quoted)
```

Kode 3.11 (Lanjutan) *Data/CsvCatalog.cs: katalog yang diimplementasikan di atas kedua berkas CSV, dibaca sekali saat awal jalan.*

```
                {
                    fields.Add(field.ToString());
                    field.Clear();
                }
                else field.Append(c);
            }
            fields.Add(field.ToString());
            yield return fields.ToArray();
        }
    }
}
```

3.6 Controller dan Perutean

Controller adalah kelas yang metode-metode publiknya, disebut **aksi** (*action*), menangani permintaan. Di ASP.NET Core MVC, controller biasanya diturunkan dari `Controller`, yang memberinya metode bantu seperti `View()`, `NotFound()`, `Ok()`, dan `RedirectToAction()`, serta properti seperti `Request`, `Response`, `ModelState`, dan `ViewData`. Sebuah aksi mengembalikan `IActionResult`, yaitu objek yang *menggambarkan* respons (view yang harus dirender, sebuah status, JSON, sebuah pengalihan) dan yang kemudian dilaksanakan kerangka kerja. LaundryMvc punya tiga controller, dan bersama-sama ketiganya memperlihatkan kedua cara perutean.

3.6.1 Perutean konvensional: `HomeController` dan `OutletsController`

Rute yang didaftarkan di `Program.cs`,

```
{controller=Home}/{action=Index}/{id?}
```

adalah **rute konvensional**: satu templat, didefinisikan secara terpusat, yang cocok untuk setiap controller yang mengikuti konvensi penamaan [7]. Segmen pertamanya menyebut controller (tanpa kata `Controller`), segmen kedua aksinya, dan segmen ketiga `id` yang opsional. `=Home` dan `=Index` adalah nilai bawaan, yang dipakai jika segmennya tidak ada, dan `?` membuat `id` opsional. **Tabel 3.2** memperlihatkan apa yang dihasilkannya dari beberapa alamat; semuanya adalah jawaban nyata LaundryMvc (Subbab 3.8).

Tabel 3.2 Cara rute konvensional `{controller=Home}/{action=Index}/{id?}` memetakan alamat ke aksi. Pencocokan tidak membedakan huruf besar dan kecil.

Alamat	Controller	Aksi	`id`	Status
`/`	`Home` (bawaan)	`Index` (bawaan)	tidak ada	200
`/Home`	`Home`	`Index` (bawaan)	tidak ada	200
`/home/index`	`Home`	`Index`	tidak ada	200
`/Home/About`	`Home`	`About`	tidak ada	200
`/Outlets`	`Outlets`	`Index` (bawaan)	tidak ada	200
`/Outlets/Details/O02`	`Outlets`	`Details`	`O02`	200
`/Outlets/Details/O09`	`Outlets`	`Details`	`O09`	404 (tidak ada gerai itu)
`/nothing/here`	tidak ada `NothingController`			404

Kode 3.12 adalah controller beranda. Ia memakai **primary constructor**, `HomeController(ILaundryCatalog catalog)`, yang tersedia untuk kelas sejak C# 12 [8]: parameter `catalog` berlaku di seluruh kelas, dan container menyediakannya ketika kerangka kerja membuat controller untuk sebuah permintaan. Tidak perlu menulis *field* maupun penugasan.

Kode 3.12 `Controllers/HomeController.cs`: halaman beranda, halaman tentang kami, dan halaman yang ditampilkan untuk kode status galat.

```
using LaundryMvc.Data;
using LaundryMvc.Models;
using Microsoft.AspNetCore.Diagnostics;
using Microsoft.AspNetCore.Mvc;

namespace LaundryMvc.Controllers;

// Reached through the conventional route {controller=Home}/{action=Index}.
public class HomeController(ILaundryCatalog catalog) : Controller
{
    public IActionResult Index()
    {
        string[] popular = ["S0001", "S0002", "S0008"];
        var model = new HomeViewModel(
            popular.Select(id => catalog.FindService(id)!).ToList(),
            catalog.Services().Count, catalog.Outlets().Count);
        return View(model);
    }

    public IActionResult About() => View();

    // An attribute route: only /error/404, /error/500, ... reach this action.
    [Route("/error/{code:int}")]
    public IActionResult Error(int code)
    {
        // The status-code middleware re-ran the request as /error/404;
        // this feature remembers the address the visitor really asked for.
        var original = HttpContext.Features.Get<IStatusCodeReExecuteFeature>();
        ViewData["Path"] = original?.OriginalPath ?? Request.Path.Value;
        Response.StatusCode = code;
        return View(code);
    }
}
```

`Index` membangun *view model* dari tiga kode populer dan dua hitungan, lalu mengembalikan `View(model)`: "render view yang bernama sama dengan aksi ini, `Views/Home/Index.cshtml`, dengan model ini". `About` tidak punya data dan mengembalikan `View()`. Aksi ketiga, `Error`, berbeda. Ia membawa **attribute route**, `[Route("/error/{code:int}")]`, sehingga hanya dapat dicapai di `/error/404`, `/error/500`, dan seterusnya, tidak pernah melalui rute konvensional. Inilah alamat yang dijalankan ulang oleh *middleware* kode status. Ketika itu terjadi, path permintaannya *adalah* `/error/404`; alamat yang sebenarnya diketik pengunjung disimpan dalam sebuah *feature* pada konteksnya, `IStatusCodeReExecuteFeature`, dan aksi itu menyalin `OriginalPath` miliknya ke `ViewData` untuk view. Aksi itu juga mengisi `Response.StatusCode = code`, agar halaman ramah itu tetap menyatakan 404 kepada peramban dan mesin pencari.

Kode 3.13 adalah controller gerai, yang paling sederhana: sebuah daftar, dan satu gerai atau `NotFound()`.

Kode 3.13 `Controllers/OutletsController.cs`: gerai-gerai, dicapai melalui rute konvensional.

```
using LaundryMvc.Data;
using Microsoft.AspNetCore.Mvc;

namespace LaundryMvc.Controllers;

// Conventional routing: /Outlets and /Outlets/Details/O01.
public class OutletsController(ILaundryCatalog catalog) : Controller
{
    public IActionResult Index() => View(catalog.Outlets());

    public IActionResult Details(string id)
    {
        var outlet = catalog.FindOutlet(id);
        if (outlet is null)
        {
            return NotFound();
        }
        return View(outlet);
    }
}
```

`NotFound()` menghasilkan respons 404 tanpa badan. Itulah persis jenis respons yang ditunggu *middleware* kode status: pada perjalanan keluar ia melihat "404, kosong" dan menjalankan ulang *pipeline* dengan `/error/404`. Controller tidak perlu tahu bahwa halaman ramah itu ada.

3.6.2 *Attribute routing*: `ServicesController`

Rute konvensional menghasilkan alamat seperti `/Services/Details/S0004`, yang benar tetapi tidak indah, dan tidak dapat menyatakan `/services/under/20000`. Untuk alamat semacam itu ASP.NET Core menawarkan **attribute routing**: rute ditulis pada controller dan aksi-aksinya [7]. **Kode 3.14** adalah controller layanan, jantung situs ini.

Kode 3.14 `Controllers/ServicesController.cs`: *attribute route*, sebuah batasan, aksi formulir, dan aksi JSON.

```
using LaundryMvc.Data;
using LaundryMvc.Models;
using Microsoft.AspNetCore.Mvc;

namespace LaundryMvc.Controllers;

// Attribute routing: every URL of this controller starts with /services.
[Route("services")]
public class ServicesController(ILaundryCatalog catalog,
    PriceCalculator calculator) : Controller
{
    [HttpGet("")]                                   // /services
    public IActionResult Index() => View(catalog.Services());

    [HttpGet("{id}")]                               // /services/S0004
    public IActionResult Details(string id)
    {
        var service = catalog.FindService(id);
        return service is null ? NotFound() : View(service);
    }

    [HttpGet("under/{max:int:min(1000)}")]          // /services/under/20000
    public IActionResult Under(int max)
    {
        ViewData["Title"] = $"Services up to {Money.Rp(max)}";
        var cheap = catalog.Services().Where(s => s.Price <= max).ToList();
        return View("Index", cheap);
    }

    [HttpGet("quote")]                    // /services/quote?serviceId=...
    public IActionResult Quote(QuoteForm form)
    {
        if (form.Quantity is null)
        {
            return View(form);                      // first visit: empty form
        }
        var service = catalog.FindService(form.ServiceId ?? "");
        if (service is null)
        {
            ModelState.AddModelError(nameof(form.ServiceId),
                "Choose a service from the list.");
        }
        if (form.Quantity <= 0)
        {
            ModelState.AddModelError(nameof(form.Quantity),
                "Enter a quantity above zero.");
```

Kode 3.14 (Lanjutan) *Controllers/ServicesController.cs: attribute route, sebuah batasan, aksi formulir, dan aksi JSON.*

```
        }
        if (service is not null && ModelState.IsValid)
        {
            form.Result = calculator.Calculate(service, form.Quantity.Value,
                form.Pickup);
        }
        return View(form);
    }

    [HttpGet("/api/services/{id}")]                // JSON for other programs
    public IActionResult Api(string id)
    {
        var service = catalog.FindService(id);
        return service is null
            ? NotFound(new { error = $"No service with code {id}." })
            : Ok(service);
    }
}
```

`[Route("services")]` pada kelas memberi setiap aksi prefiks `services`. Setiap `[HttpGet("...")]` menambahkan sisa templatnya dan membatasi aksi itu pada metode `GET`. Begitu sebuah controller punya *attribute route*, rute konvensional tidak lagi mencapainya. **Tabel 3.3** mendaftar templat-templat yang didefinisikan controller ini.

Tabel 3.3 *Attribute route* milik `ServicesController`. Templat yang diawali `/` mengabaikan prefiks controller.

Templat	Aksi	Contoh	Hasil
`services`	`Index`	`/services`	daftarnya
`services/{id}`	`Details`	`/services/S0004`	satu layanan, atau 404
`services/under/{max:int:min(1000)}`	`Under`	`/services/under/20000`	daftar yang disaring
`services/quote`	`Quote`	`/services/quote?serviceId=S0001&quantity=2.5`	formulir, beserta hasilnya
`/api/services/{id}`	`Api`	`/api/services/S0004`	JSON, atau 404 dengan galat JSON

Tiga rincian patut diperhatikan.

Batasan. `{max:int:min(1000)}` menggabungkan dua batasan: segmennya harus bilangan bulat, dan paling kecil 1000. Karena itu `/services/under/abc` dan `/services/under/500` tidak cocok dengan rute *mana pun*, dan jawabannya 404 tanpa aksinya pernah dijalankan. Batasan bukanlah validasi masukan pengguna; ia pernyataan tentang alamat mana yang ada. Aksinya menerima `max` yang sudah diubah menjadi `int`.

Prioritas. `services/{id}` dan `services/quote` sama-sama dapat cocok dengan `/services/quote`. ASP.NET Core tidak mencoba rute sesuai urutan kemunculannya dalam kode, seperti MiniFrame di Bab 2. Ia mengurutkannya: segmen literal (`quote`) lebih khusus daripada parameter (`{id}`), sehingga `/services/quote` selalu mencapai `Quote` [9]. Pindahkan metode-metodenya di dalam berkas dan tidak ada yang berubah.

Rute di luar prefiks. Templat `Api` diawali garis miring, `/api/services/{id}`, sehingga bersifat mutlak: prefiks kelas diabaikan. Gaya aksi yang sama melayani dua khalayak, `/services/S0004` untuk manusia dan `/api/services/S0004` untuk program.

3.6.3 Apa yang dikembalikan aksi

Empat jenis hasil di **Kode 3.14** mencakup sebagian besar yang pernah dilakukan sebuah aksi.

- `View(model)` merender sebuah view. `View("Index", cheap)` di `Under` merender view dengan nama *lain*, yaitu view daftar milik `Index`, dan memakainya ulang untuk daftar yang disaring; judulnya diteruskan lewat `ViewData["Title"]`.
- `NotFound()` mengirim 404 tanpa badan, yang diubah *middleware* kode status menjadi halaman ramah. `NotFound(new { error = … })` mengirim 404 *dengan* badan JSON; karena badannya tidak kosong, *middleware* membiarkannya, dan aplikasi Dimas menerima galat yang dapat dibaca mesin, bukan halaman HTML.
- `Ok(service)` mengirim 200 dengan objek yang diserialisasi menjadi JSON, dengan nama *camelCase* yang sama seperti di Bab 1.
- Jenis keempat, pengalihan, muncul di Bab 4 (`RedirectToAction`).

3.6.4 *Model binding*: formulir penawaran

Aksi `Quote(QuoteForm form)` tidak pernah membaca *query string* sendiri. Sebelum memanggilnya, **model binding** MVC membuat `QuoteForm` dan mengisi setiap propertinya dari permintaan: untuk `ServiceId` ia mencari nilai bernama `serviceId` (tanpa membedakan huruf besar-kecil) di nilai rute, *query string*, dan isian formulir; ia menemukan `?serviceId=S0001`; ia mengubah teksnya ke tipe properti [10]. Untuk `Quantity=2.5` ia mengubahnya menjadi `decimal?`; nilai *query string* diubah dengan kultur invarian, sehingga pemisah desimalnya selalu titik. Untuk kotak centang `Pickup` ia menemukan `true`.

Ketika sebuah nilai tidak dapat diubah, *model binding* tidak melempar *exception*. Ia mencatat galat di `ModelState`, kamus galat per properti, dan membiarkan properti itu kosong. `?quantity=abc` menghasilkan galat "The value 'abc' is not valid for Quantity." tanpa satu baris pun kode kita. Aksi menambahkan galatnya sendiri dengan `ModelState.AddModelError` (layanan yang tidak dikenal, jumlah nol atau kurang), dan hanya menghitung jika `ModelState.IsValid`. Formulir yang bergalat cukup ditampilkan lagi, dengan pesan-pesannya di atas, dan apa pun yang diketik pelanggan masih ada di isiannya.

Mengapa penawaran memakai `GET` dan bukan `POST`? Karena menghitung harga tidak mengubah apa pun di server: itu pertanyaan, bukan pesanan, dan HTTP mendefinisikan `GET` sebagai metode yang *aman* tepat untuk permintaan semacam itu [11]. Formulir `GET` menaruh pertanyaannya di alamat, sehingga sebuah penawaran dapat ditandai, dibagikan, dan dimuat ulang dengan aman. Membuat pesanan *akan* mengubah data, dan Bab 5 melakukannya dengan `POST` dan token *antiforgery*.

3.7 View Razor

View Razor adalah berkas HTML berekstensi `.cshtml` yang di dalamnya C# ditandai dengan satu `@` [12]. `@Model.Name` menuliskan nilai sebuah ekspresi, **di-*encode* sebagai HTML**; `@{ … }` adalah blok pernyataan C#; `@if`, `@foreach`, dan `@switch` adalah pernyataan C# yang badannya kembali berupa HTML. Razor mengompilasi setiap view menjadi kelas C# ketika proyek di-*build*, sehingga salah ketik di view menjadi galat kompilasi, bukan kejutan di depan pelanggan.

3.7.1 Pengaturan untuk semua view: `_ViewImports` dan `_ViewStart`

Kode 3.15 dibaca sebelum setiap view di folder `Views`. Baris-baris `@using`-nya membuat *namespace* kita tersedia, sehingga view dapat menulis `Service` alih-alih `LaundryMvc.Models.Service`, dan `@addTagHelper` menyalakan *tag helper* bawaan (Subbab 3.7.4).

Kode 3.15 `Views/_ViewImports.cshtml`: *namespace* dan *tag helper* untuk setiap view.

```
@using LaundryMvc
@using LaundryMvc.Models
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Kode 3.16 dijalankan sebelum setiap view dan menetapkan *layout*-nya; sebuah view dapat menimpanya, atau mengisi `Layout = null` agar tidak memakai *layout*.

Kode 3.16 `Views/_ViewStart.cshtml`: setiap view memakai *layout* `_Layout`.

```
@{
    Layout = "_Layout";
}
```

3.7.2 *Layout*

Kode 3.17 adalah bingkai di sekeliling setiap halaman, aturan kedua Bu Wati [13]. Bingkai ini ditulis sekali; setiap halaman muncul di tempat `@RenderBody()`.

Kode 3.17 `Views/Shared/_Layout.cshtml`: bingkai bersama, dengan menu yang menandai halaman yang sedang dibuka.

```
@{
    // "here" marks the menu link of the page being shown.
    string? Here(string controller, string action) =>
        ViewContext.RouteData.Values["controller"] as string == controller
        && ViewContext.RouteData.Values["action"] as string == action
            ? "here" : null;
}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Profitina Laundry</title>
    <link rel="icon" href="data:," />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
</head>
<body>
    <header>
        <a class="brand" asp-controller="Home" asp-action="Index">
            Profitina Laundry</a>
        <nav>
            <a class="@Here("Home", "Index")"
               asp-controller="Home" asp-action="Index">Home</a>
            <a class="@Here("Services", "Index")"
               asp-controller="Services" asp-action="Index">Services</a>
            <a class="@Here("Services", "Quote")"
               asp-controller="Services" asp-action="Quote">Price quote</a>
            <a class="@Here("Outlets", "Index")"
               asp-controller="Outlets" asp-action="Index">Outlets</a>
            <a class="@Here("Home", "About")"
               asp-controller="Home" asp-action="About">About</a>
        </nav>
    </header>
    <main>
        @RenderBody()
    </main>
    <footer>
        @await RenderSectionAsync("Footer", required: false)
        <p>&copy; @DateTime.Now.Year Profitina Laundry, Surabaya</p>
    </footer>
</body>
</html>
```

Blok `@{ … }` di bagian atas mendefinisikan fungsi lokal kecil, `Here`, yang mengembalikan kelas CSS `"here"` jika controller dan aksi permintaan saat ini (dari `ViewContext.RouteData`) adalah yang disebutkan, dan `null` jika bukan. Razor punya aturan yang berguna untuk atribut: jika nilai sebuah atribut `null`, atribut itu dihilangkan sama sekali, sehingga tautan halaman lain tidak membawa `class` apa pun. Judulnya berasal dari `ViewData["Title"]`, yang diisi setiap view. `@await RenderSectionAsync("Footer", required: false)` menandai **section** opsional: sebuah view boleh mengirim HTML ke tempat ini di *layout*, dan halaman Tentang Kami melakukannya. `asp-append-version="true"` pada lembar gaya menambahkan sidik isi berkas ke URL-nya, sehingga peramban mengambil CSS baru begitu berkasnya berubah.

3.7.3 Halaman, *partial view*, dan *section*

Kode 3.18 adalah halaman beranda. `@model HomeViewModel` mendeklarasikan tipe `Model`, sehingga `Model.Popular` diperiksa oleh kompilator. Ketiga kartunya digambar oleh **partial view**, yaitu potongan halaman yang disimpan di berkasnya sendiri (**Kode 3.19**), yang dipanggil sekali untuk setiap layanan dengan `<partial name="_ServiceCard" model="service" />` [14].

Kode 3.18 `Views/Home/Index.cshtml`: halaman beranda, dengan *partial view* untuk setiap kartu.

```
@model HomeViewModel
@{
    ViewData["Title"] = "Home";
}
<h1>Clean clothes, honest prices</h1>
<p>We wash, iron and dry-clean in @Model.OutletCount outlets across
   Surabaya. Choose from @Model.ServiceCount services, or
   <a asp-controller="Services" asp-action="Quote">work out a price</a>
   before you come.</p>
<h2>Most asked for</h2>
<div class="cards">
    @foreach (var service in Model.Popular)
    {
        <partial name="_ServiceCard" model="service" />
    }
</div>
```

Kode 3.19 `Views/Shared/_ServiceCard.cshtml`: satu kartu, ditulis sekali dan dipakai untuk setiap layanan populer.

```
@model Service
<div class="card">
    <h3>@Model.Name</h3>
    <p class="price">@Money.Rp(Model.Price) <span>per @Model.Unit</span></p>
    <a asp-controller="Services" asp-action="Details"
       asp-route-id="@Model.Id">More about @Model.Id</a>
</div>
```

Halaman Tentang Kami, **Kode 3.20**, memperlihatkan **section**: `@section Footer { … }` mengirim isinya ke tempat di *layout* yang berisi `RenderSectionAsync("Footer")`, di bawah setiap baris kaki halaman lain pada halaman itu.

Kode 3.20 `Views/Home/About.cshtml`: halaman yang menambahkan jam buka ke kaki halaman *layout* melalui sebuah *section*.

```
@{
    ViewData["Title"] = "About us";
}
<h1>About us</h1>
<p>Profitina Laundry opened its first outlet in Rungkut in 2019. Every
   order is weighed or counted in front of you, gets a rack number, and
   is ready on the day we promise.</p>
@section Footer {
    <p class="hours">All outlets are open every day, 07.00 to 21.00.</p>
}
```

Daftar layanan, **Kode 3.21**, melayani dua aksi. `ViewData["Title"] ??= "Our services"` mengisi judul hanya jika aksinya belum mengisinya, sehingga view yang sama berjudul “Our services” untuk `/services` dan “Services up to Rp 20.000” untuk `/services/under/20000`.

Kode 3.21 `Views/Services/Index.cshtml`: tabel layanan, dipakai bersama oleh aksi `Index` dan `Under`.

```
@model IReadOnlyList<Service>
@{
    ViewData["Title"] ??= "Our services";
}
<h1>@ViewData["Title"]</h1>
<p>@Model.Count services. Prices are per kilogram, per item or per pair.</p>
<table class="list">
    <thead>
        <tr><th>Code</th><th>Service</th><th class="num">Price</th><th></th></tr>
    </thead>
    <tbody>
    @foreach (var s in Model)
    {
        <tr>
            <td>@s.Id</td>
            <td>@s.Name</td>
            <td class="num">@Money.Rp(s.Price) / @s.Unit</td>
            <td><a asp-action="Details" asp-route-id="@s.Id">Details</a></td>
        </tr>
    }
    </tbody>
</table>
<p>On a budget? See the services
   <a asp-action="Under" asp-route-max="20000">up to Rp 20.000</a>.</p>
```

Kode 3.22 `Views/Services/Details.cshtml`: satu layanan, dengan tautan ke penawaran untuknya.

```
@model Service
@{
    ViewData["Title"] = Model.Name;
}
<h1>@Model.Name</h1>
<dl class="facts">
    <dt>Code</dt><dd>@Model.Id</dd>
    <dt>Price</dt><dd>@Money.Rp(Model.Price) per @Model.Unit</dd>
    <dt>Charged by</dt><dd>@(Model.PerKg ? "weight" : "count")</dd>
</dl>
<p><a asp-action="Quote" asp-route-serviceId="@Model.Id">Get a quote
   for this service</a> or <a asp-action="Index">see all services</a>.</p>
```

Kode 3.22 menampilkan satu layanan. Ekspresi `@(Model.PerKg ? "weight" : "count")` memerlukan tanda kurung, karena tanpa itu Razor berhenti membaca C# di spasi pertama. View gerai, **Kode 3.23** dan **Kode 3.24**, mengikuti pola yang sama; `ToString("d MMMM yyyy")` memformat tanggal pembukaan menjadi, misalnya, “1 February 2022”.

Kode 3.23 `Views/Outlets/Index.cshtml`: daftar gerai.

```
@model IReadOnlyList<Outlet>
@{
    ViewData["Title"] = "Outlets";
}
<h1>Our outlets</h1>
<ul class="outlets">
    @foreach (var o in Model)
    {
        <li><a asp-action="Details" asp-route-id="@o.Id">@o.Name</a><br />
            @o.Address</li>
    }
</ul>
```

Kode 3.24 `Views/Outlets/Details.cshtml`: satu gerai.

```
@model Outlet
@{
    ViewData["Title"] = Model.Name;
}
<h1>@Model.Name</h1>
<dl class="facts">
    <dt>Address</dt><dd>@Model.Address</dd>
    <dt>Phone</dt><dd>@Model.Phone</dd>
    <dt>Open since</dt><dd>@Model.Opened.ToString("d MMMM yyyy")</dd>
</dl>
<p><a asp-action="Index">All outlets</a></p>
```

View galat, **Kode 3.25**, menerima kode status sebagai modelnya dan path aslinya melalui `ViewData`; path itu ditulis dengan `@`, sehingga alamat yang penuh `<script>` pun ditampilkan tanpa bahaya.

Kode 3.25 `Views/Shared/Error.cshtml`: halaman ramah untuk 404 dan status galat lainnya.

```
@model int
@{
    ViewData["Title"] = Model == 404 ? "Not found" : "Something went wrong";
}
<h1>@ViewData["Title"]</h1>
@if (Model == 404)
{
    <p>We could not find the page
        <code>@ViewData["Path"]</code>.</p>
}
else
{
    <p>The server could not finish your request (status @Model).</p>
}
<p><a asp-controller="Home" asp-action="Index">Back to the home page</a></p>
```

3.7.4 *Tag helper* dan formulir penawaran

Tag helper adalah kelas C# yang ikut merender elemen HTML yang tampak biasa, dikenali dari atribut yang diawali `asp-` [15]. Merekalah sebabnya view di bab ini hampir tidak berisi URL yang ditulis tangan.

- `<a asp-controller="Services" asp-action="Details" asp-route-id="@s.Id">` meminta *sistem perutean* memberikan URL aksi itu dengan `id` itu. Karena `ServicesController` memakai *attribute route*, jawabannya `/services/S0004`; untuk `OutletsController`, yang memakai rute konvensional, tautan sejenis menjadi `/Outlets/Details/O02`. Jika sebuah rute berubah, setiap tautan ikut berubah. Jika controller tidak disebut, yang dimaksud adalah controller saat ini, seperti di view-view layanan.
- `<form asp-action="Quote" method="get">` menghasilkan atribut `action` dari rutenya, `/services/quote`.
- `<label asp-for="Quantity">`, `<input asp-for="Quantity" />`, dan `<select asp-for="ServiceId">` mengikat elemen ke properti model: mereka menulis `name` dan `id` yang akan dicari *model binder*, mengembalikan nilai saat ini ke isiannya, dan menambahkan kelas CSS `input-validation-error` jika `ModelState` menyimpan galat untuk properti itu. Untuk properti `bool`, `<input asp-for="Pickup" />` menjadi kotak centang ditambah isian tersembunyi `Pickup=false` di akhir formulir, agar kotak yang tidak dicentang tetap mengirim nilai.
- `<div asp-validation-summary="All">` mendaftar setiap galat di `ModelState`.

Kode 3.26 adalah view penawaran, tempat semuanya bekerja bersama.

Kode 3.26 `Views/Services/Quote.cshtml`: formulir penawaran, dibangun dengan *tag helper*, beserta tabel hasilnya.

```
@model QuoteForm
@inject LaundryMvc.Data.ILaundryCatalog Catalog
@{
    ViewData["Title"] = "Price quote";
    var choices = Catalog.Services().Select(s => new SelectListItem(
        $"{s.Id} {s.Name} ({Money.Rp(s.Price)}/{s.Unit})", s.Id));
}
<h1>Price quote</h1>
<form asp-action="Quote" method="get" class="quote">
    <div asp-validation-summary="All" class="errors"></div>
    <label asp-for="ServiceId">Service</label>
    <select asp-for="ServiceId" asp-items="choices">
        <option value="">-- choose a service --</option>
    </select>
    <label asp-for="Quantity">Quantity (kg, items or pairs)</label>
    <input asp-for="Quantity" />
    <label class="check"><input asp-for="Pickup" /> Pickup and delivery</label>
    <button type="submit">Calculate</button>
</form>
@if (Model.Result is { } q)
{
    <table class="list quote-result">
        <tr><th>Service</th><td>@q.Service.Name</td></tr>
        <tr><th>Billed</th>
            <td>@q.Billed @q.Service.Unit x @Money.Rp(q.Service.Price)</td></tr>
        <tr><th>Subtotal</th><td class="num">@Money.Rp(q.Subtotal)</td></tr>
        <tr><th>Pickup and delivery</th>
            <td class="num">@Money.Rp(q.PickupFee)</td></tr>
        <tr class="total"><th>Total</th><td class="num">@Money.Rp(q.Total)</td></tr>
    </table>
    <ul class="notes">
        @foreach (var note in q.Notes)
        {
            <li>@note</li>
        }
    </ul>
}
```

Dua fitur lagi muncul di sini. `@inject LaundryMvc.Data.ILaundryCatalog Catalog` meminta sebuah layanan kepada container DI *dari dalam view*: daftar pilihan untuk kotak *drop-down* adalah urusan tampilan, jadi view mengambilnya sendiri alih-alih membebani controller. Dan tabel hasil hanya ditampilkan jika `Model.Result is { } q`, sebuah *pattern* C# yang bernilai benar jika `Result` tidak *null* dan sekaligus menamainya `q`.

3.8 Menjalankannya

3.8.1 Di terminal atau di VS Code

Dari folder `Chapter03/aspnet/LaundryMvc`:

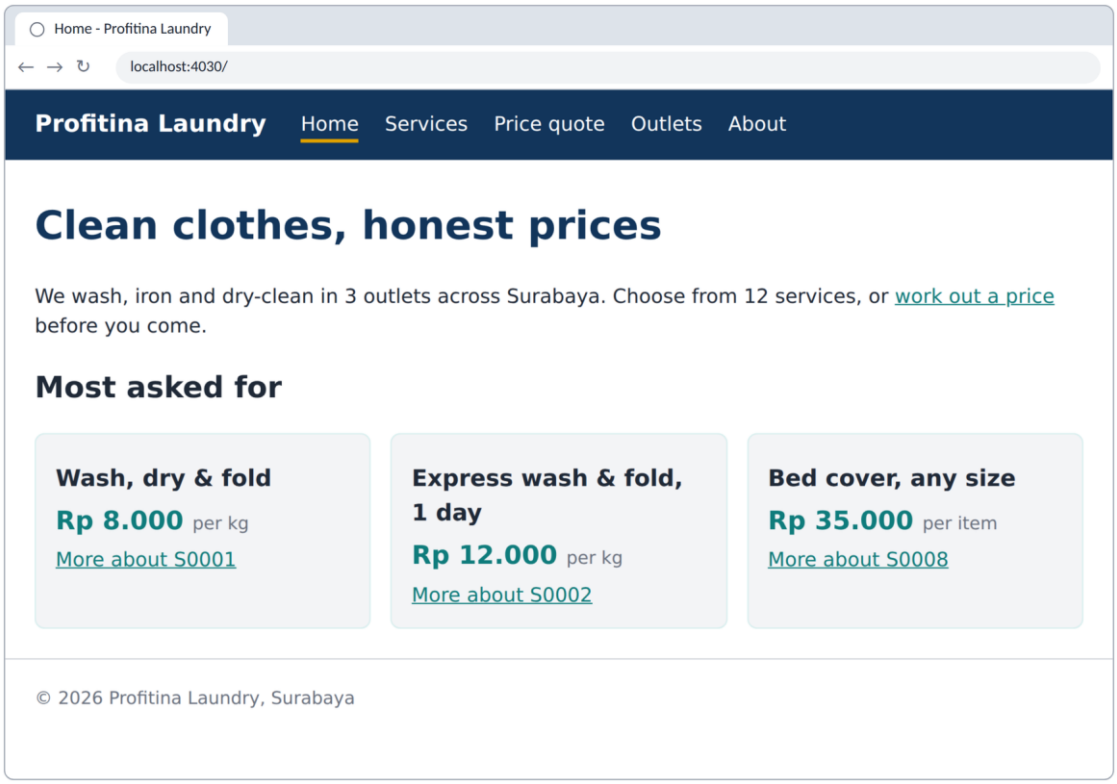
```
dotnet run
```

```
Using launch settings from …/LaundryMvc/Properties/launchSettings.json...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://127.0.0.1:4030
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: …/Chapter03/aspnet/LaundryMvc
```

(Mesin *build* menjalankan aplikasi dengan `--urls http://127.0.0.1:4030`, itulah sebabnya ia mendengarkan di `127.0.0.1` alih-alih `localhost`; port dan semua yang lain sama.) Lingkungannya adalah **Development**, yang ditetapkan oleh profil peluncuran. Di VS Code, buka folder `LaundryMvc` dengan **File → Open Folder…**; C# Dev Kit mengenali proyeknya, dan **Run → Start Debugging** (F5) menjalankannya dengan *debugger* terpasang, sehingga *breakpoint* di `PriceCalculator.Calculate` menghentikan program ketika sebuah penawaran dihitung.

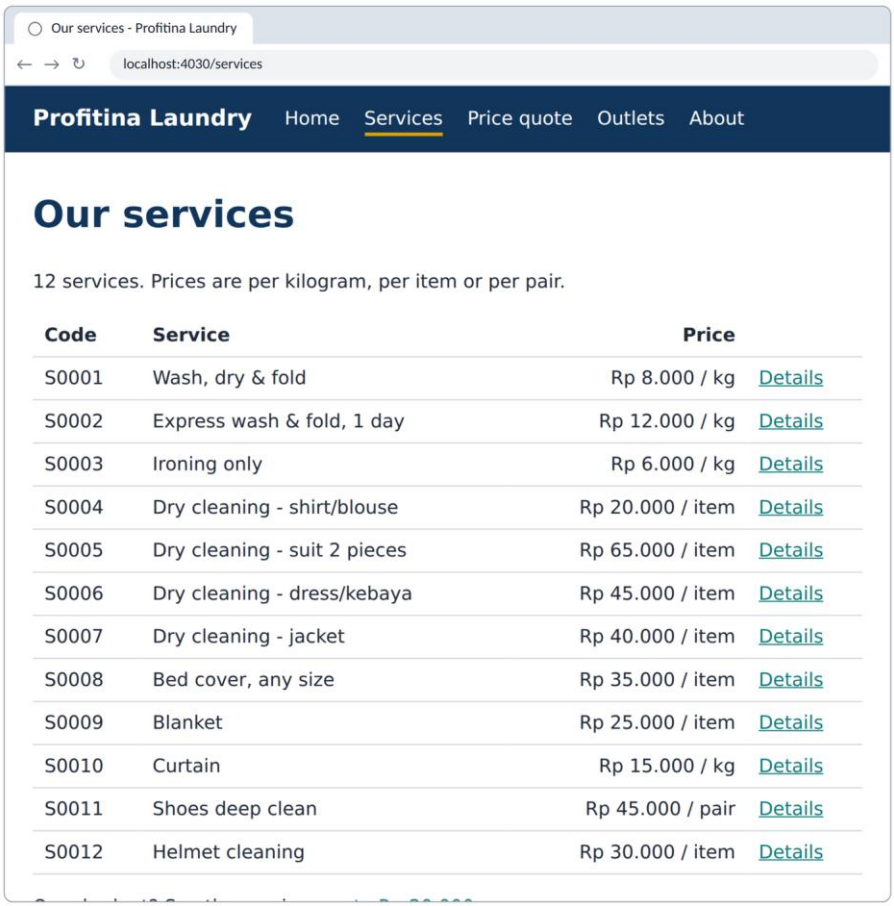
3.8.2 Apa yang ditampilkan peramban

Gambar 3.3 adalah halaman beranda. Menu menandai "Home" dengan garis bawah kuning yang dipilih fungsi `Here` milik *layout*; ketiga kartunya adalah *partial view* yang dipanggil tiga kali; hitungannya berasal dari *view model*.



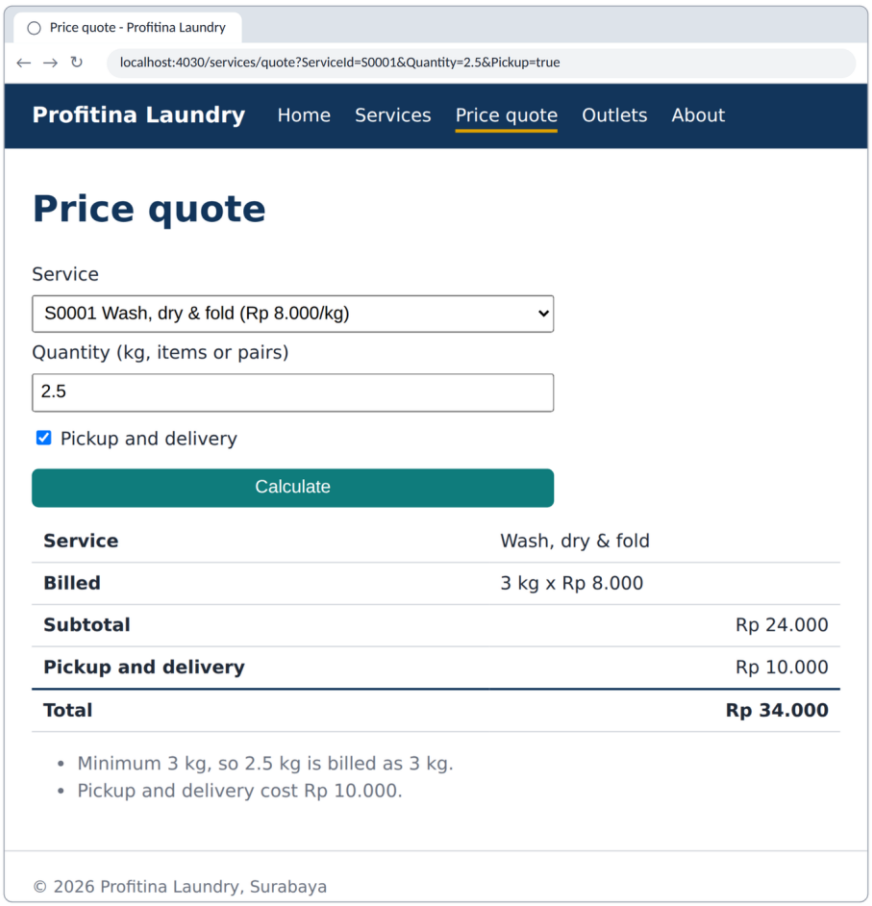
Gambar 3.3 Halaman beranda LaundryMvc: kepala halaman dan menu dari *layout*, teks view dengan hitungan dari *view model*, dan tiga kartu yang digambar *partial view* `_ServiceCard`.

Gambar 3.4 adalah `/services`: view daftar dengan kedua belas layanan, setiap harga diformat oleh `Money.Rp` dan setiap tautan "Details" dihasilkan *tag helper* dari *attribute route*.



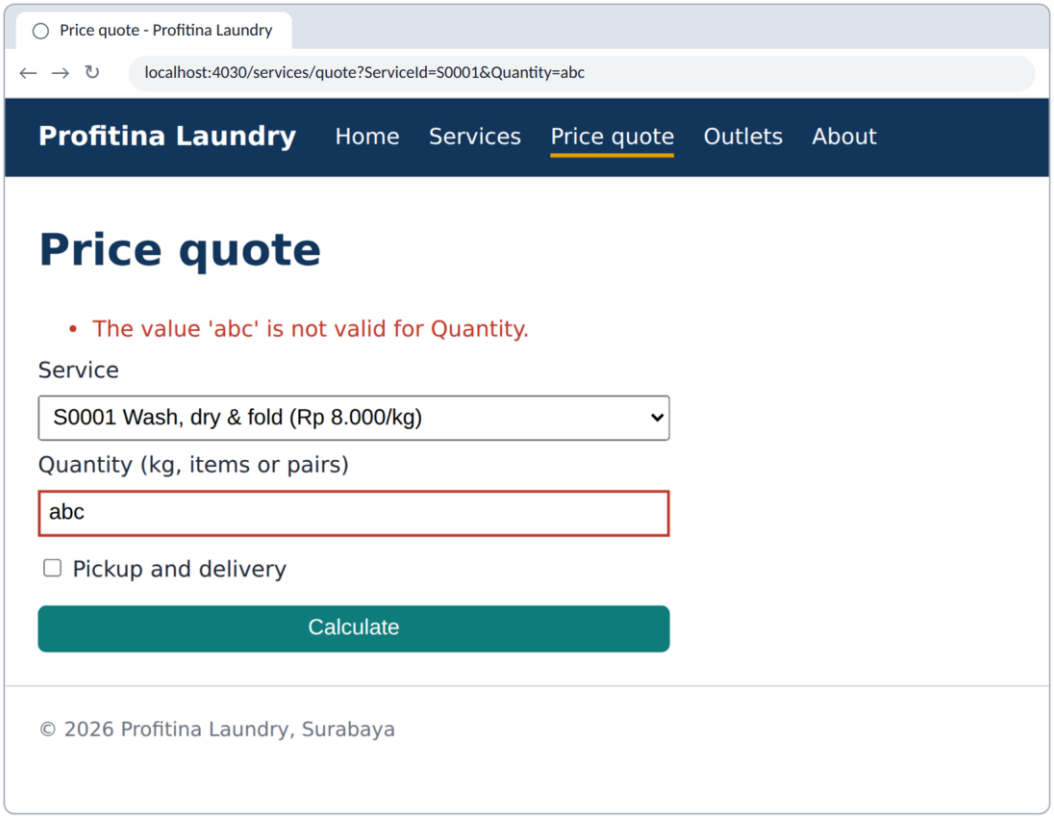
Gambar 3.4 Daftar layanan di `/services`, dirender oleh `Views/Services/Index.cshtml`.

Gambar 3.5 adalah penawaran untuk 2,5 kg "cuci, kering, lipat" dengan antar-jemput, contoh yang dikerjakan secara manual di Subbab 3.5.2. Situs itu sepakat dengan hitungan manual: 3 kg ditagihkan, subtotal Rp 24.000, antar-jemput Rp 10.000, total Rp 34.000, dan kedua catatan menjelaskan alasannya. Formulirnya masih memuat pilihan-pilihannya, karena *tag helper* menuliskan kembali nilai terikat ke isiannya.



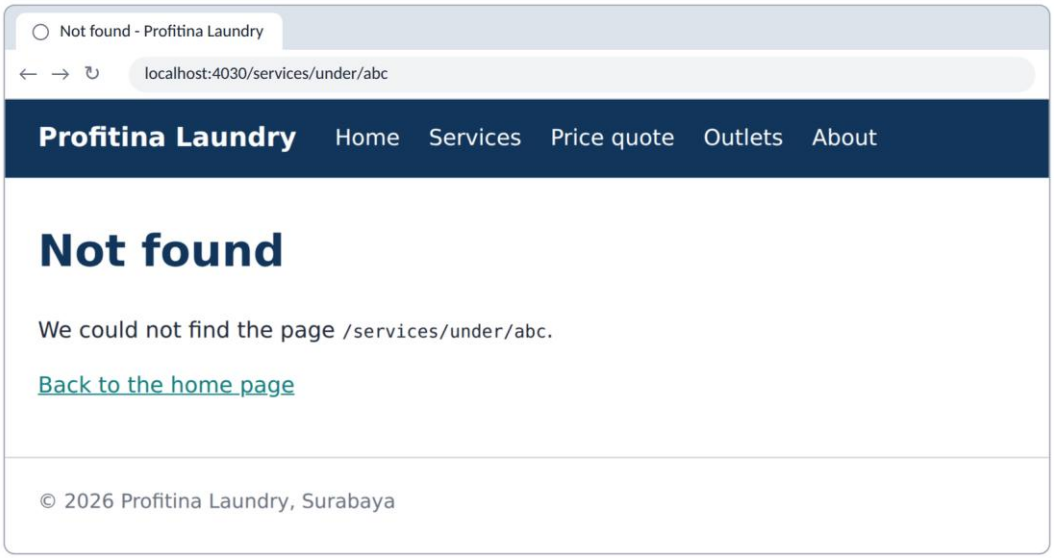
Gambar 3.5 Sebuah penawaran harga: *model binding* mengisi `QuoteForm` dari alamat, `PriceCalculator` menerapkan berat minimal dan biaya antar-jemput, dan view menampilkan hasilnya beserta penjelasannya.

Gambar 3.6 adalah formulir yang sama dengan `Quantity=abc`. Aksinya tidak pernah melihat angka yang rusak: *model binding* mencatat galatnya di `ModelState`, ringkasan validasi mendaftarnya, dan isian masukannya diberi bingkai merah dari kelas `input-validation-error`.



Gambar 3.6 Formulir penawaran setelah `Quantity=abc`: galat konversi dari *model binding* muncul di ringkasan validasi, dan isiannya ditandai.

Gambar 3.7 adalah apa yang didapat alamat yang salah ketik: `/services/under/abc` tidak cocok dengan rute mana pun karena batasan `int`, 404 yang kosong diganti view galat pada perjalanan keluar, dan halamannya tetap memakai *layout* toko.



Gambar 3.7 Halaman 404 yang ramah untuk `/services/under/abc`, dihasilkan *middleware* kode status dan `HomeController.Error`.

3.8.3 Setiap alamat, diperiksa

Skrip `Chapter03/verify.sh` mem-*build* aplikasi, menjalankannya, dan menanyakan setiap alamat yang disebut bab ini dengan `curl`. **Tabel 3.4** adalah bagian pertama keluaran sesungguhnya, yaitu kode status setiap alamat; tabel ini menegaskan kedua tabel perutean di Subbab 3.6.

Tabel 3.4 Kode status yang dikembalikan LaundryMvc yang sedang berjalan untuk setiap alamat bab ini, dari `verify.sh`.

Status	Alamat	Alasan
200	`/`, `/Home`, `/home/index`	rute konvensional dengan nilai bawaan; pencocokan tidak membedakan huruf besar-kecil
200	`/services`, `/services/S0004`, `/services/s0004`	*attribute route*; `FindService` tidak membedakan huruf besar-kecil
200	`/services/quote`	literal `quote` mengalahkan `{id}`
200	`/services/under/20000`	`int` dan `min(1000)` terpenuhi
404	`/services/under/abc`, `/services/under/500`	batasan gagal, tidak ada rute yang cocok
200	`/Outlets`, `/Outlets/Details/O02`, `/Home/About`	rute konvensional
404	`/Outlets/Details/O09`	aksinya mengembalikan `NotFound()`
404	`/nothing/here`	tidak ada controller bernama `Nothing`
200	`/api/services/S0004`	*attribute route* mutlak, JSON

Endpoint JSON menjawab seperti ini, termasuk properti terhitungnya:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Server: Kestrel
Transfer-Encoding: chunked

{"id":"S0004","description":"Dry cleaning - shirt/blouse (per
item)","price":20000.00,"name":"Dry cleaning - shirt/blouse","perKg":false,"unit":"item"}

HTTP/1.1 404 Not Found
Content-Type: application/json; charset=utf-8
Server: Kestrel
Transfer-Encoding: chunked

{"error":"No service with code S9999."}
```

Jawaban kedua memperlihatkan beda kedua panggilan `NotFound`: sebuah badan diberikan, sehingga *middleware* kode status tidak menggantinya dengan HTML. Untuk formulir penawaran, skrip hanya mengambil sel tabel, catatan, dan pesan galat dari setiap halaman. Enam pertanyaan, enam jawaban nyata:

```
== quote serviceId=S0001&quantity=2.5&pickup=true
Service Wash, dry &amp;amp; fold
Billed
3 kg x Rp 8.000
Subtotal Rp 24.000
Pickup and delivery
Rp 10.000
Total Rp 34.000
Minimum 3 kg, so 2.5 kg is billed as 3 kg.
Pickup and delivery cost Rp 10.000.
== quote serviceId=S0005&quantity=1.5&pickup=true
Service Dry cleaning - suit 2 pieces
Billed
2 item x Rp 65.000
Subtotal Rp 130.000
Pickup and delivery
Rp 0
Total Rp 130.000
Items are counted whole: 1.5 becomes 2.
Pickup and delivery are free from Rp 100.000.
== quote serviceId=S0002&quantity=9
Service Express wash &amp;amp; fold, 1 day
Billed
9 kg x Rp 12.000
Subtotal Rp 108.000
Pickup and delivery
Rp 0
Total Rp 108.000
== quote serviceId=S0099&quantity=4
Choose a service from the list.
== quote serviceId=S0001&quantity=abc
The value &amp;#x27;abc&amp;#x27; is not valid for Quantity.
== quote serviceId=S0001&quantity=0
Enter a quantity above zero.
```

Periksa masing-masing terhadap aturannya. Setelan jas dihitung per barang, sehingga 1,5 menjadi 2 dan $2 \times 65000 = 130000$, di atas ambang, sehingga antar-jemput gratis. Sembilan kilogram layanan kilat adalah $9 \times 12000 = 108000$; antar-jemput tidak diminta, jadi biayanya nol dan tidak ada catatan antar-jemput. Kode yang tidak dikenal, bukan-angka, dan nol masing-masing menghasilkan tepat satu pesan. `&` dan `'` adalah *encoding* HTML milik Razor untuk `&` dan `'`, terlihat di sini karena skrip menampilkan HTML mentahnya; peramban menampilkannya sebagai karakter aslinya.

Satu baris keluaran ini kurang tepat: “2 item” seharusnya “2 items”. Ini bug kecil yang nyata, jenis bug yang hanya terungkap ketika programnya dijalankan, dan Soal 1 memperbaikinya.

3.8.4 Log

Selama `verify.sh` mengajukan pertanyaannya, konsol aplikasi mencetak satu baris per permintaan, yang ditulis *middleware* pertama kita. Cuplikannya:

```
info: LaundryMvc[0]
      GET /services/under/20000 -> 200 in 5 ms
info: LaundryMvc[0]
      GET /services/under/abc -> 404 in 36 ms
info: LaundryMvc[0]
      GET /Outlets/Details/O09 -> 404 in 4 ms
info: LaundryMvc[0]
      GET /nothing/here -> 404 in 1 ms
info: LaundryMvc[0]
      GET /services/quote?serviceId=S0001&quantity=2.5&pickup=true -> 200 in 12 ms
```

Perhatikan `/nothing/here`: log menyatakan 404, dengan path yang diketik pengunjung, meskipun badannya berasal dari `/error/404`. *Middleware* pencatatan berada *di luar middleware* kode status, sehingga ia melihat permintaan aslinya masuk dan status akhirnya keluar, dan tidak pernah melihat permintaan yang dijalankan ulang. Soal 4 memindahkannya ke dalam dan memperlihatkan bedanya. Permintaan pertama setelah awal jalan memakan waktu lebih lama (baris pertama log lengkap menunjukkan sekitar 200 ms) karena view dan katalog dimuat pada pemakaian pertama; waktu sesudahnya beberapa milidetik saja.

3.9 Menjadikannya Lebih Baik

3.9.1 Versi controller gemuk, dan mengapa kita tidak menulisnya

Ada gunanya melihat versi `Quote` yang sering ditulis pemula pertama kali. Aturan harganya diketik langsung ke dalam aksi:

```
// Tempting, and wrong: the rules live inside the controller action.
decimal billed = form.Quantity.Value;
if (service.Description.EndsWith("(per kg)") && billed < 3) billed = 3;
decimal total = billed * service.Price;
if (form.Pickup && total < 100000) total += 10000;
```

Versi ini berjalan, dan tampak lebih pendek. Tetapi konstanta 3, 100000, dan 10000 kini berada di controller, tempat halaman lain, *endpoint* JSON, dan pengujian mana pun tidak dapat menjangkaunya; penjelasannya (“ditagihkan sebagai 3 kg”) harus dibangun di view dengan mengulang uji yang sama; dan pada hari layar konter di Bab 5 memerlukan perhitungan yang sama, aturannya akan disalin, dan salinan-salinan itu akan saling menjauh, persis seperti situs lama Bu Wati menampilkan dua harga setrika. `PriceCalculator` hanya berharga satu kelas kecil dan menghapus semua masalah itu.

3.9.2 Galat yang ramah yang tetap jujur statusnya

Halaman galat yang ramah tidak boleh berpura-pura semuanya baik-baik saja. `HomeController.Error` mengisi `Response.StatusCode = code`, sehingga halaman 404 dikirim dengan status 404: peramban, mesin pencari, dan perkakas pemantau melihat kenyataannya, dan manusia melihat halaman dengan gaya toko. *Endpoint* JSON sengaja menjawab galat dengan badan JSON agar program tidak diberi HTML yang tidak dapat dibacanya. Perbedaan antara pengembangan dan produksi juga diuji: aksi yang melempar *exception* mengembalikan, di Development, *developer exception page* dengan *stack trace*-nya, dan, ketika dijalankan dengan `ASPNETCORE_ENVIRONMENT=Production` dan `--no-launch-profile`, view galat kita dengan “Something went wrong” dan status 500, sementara *exception* lengkapnya masuk ke log server (Soal 6).

3.9.3 Yang masih kurang

Situs ini membaca datanya dari berkas CSV dan tidak dapat mengubahnya. Bab 5 menghubungkan Entity Framework Core dan SQLite, menambahkan layar konter yang mencatat pesanan, dan menjadikan katalog sebagai implementasi `ILaundryCatalog` yang sama di atas basis data. Formulir penawaran memeriksa masukannya secara manual; Bab 6 memperkenalkan atribut validasi yang menguraikan aturan pada model, dan menambahkan login untuk karyawan. Bab 4 lebih dulu melatih materi bab ini pada sebuah proyek kecil yang lengkap.

3.10 Memakainya Sungguhan

3.10.1 Usaha Anda sendiri di atas LaundryMvc

LaundryMvc adalah kerangka yang baik untuk usaha kecil mana pun yang punya daftar harga, cabang, dan satu-dua aturan. Ganti `data/Service.csv` dan `data/Outlet.csv` dengan milik Anda (pertahankan kolomnya, atau ubah *record* dan katalognya bersama-sama), ubah kode-kode populer di `HomeController.Index`, dan tulis ulang `PriceCalculator` dengan aturan Anda sendiri: biaya minimal seorang penjahit, harga per halaman sebuah percetakan dengan potongan di atas seratus halaman, biaya pendaftaran klinik yang ditambahkan sekali. Karena aturannya tinggal di satu kelas, halaman-halamannya tidak berubah. Jika Anda mengubah tampilan, ubah `site.css` dan `_Layout.cshtml` saja.

3.10.2 Sekilas tentang *deployment*

Untuk uji cepat di komputer lain dalam jaringan yang sama, `dotnet run --urls http://0.0.0.0:4030` membuat aplikasi mendengarkan di semua antarmuka jaringan, dan ponsel di Wi-Fi yang sama dapat membuka `http://<alamat komputer>:4030`. Cara ini baik untuk mencoba dan salah untuk produksi, yang memerlukan hasil *publish* (`dotnet publish -c Release`), HTTPS, pengelola proses, dan *reverse proxy*; Bab 6 mengerjakannya dengan benar.

3.10.3 Profitina dalam praktik

Aplikasi web Profitina sendiri mengikuti bentuk yang sama dengan bab ini, dalam kerangka kerja yang berbeda. *Pipeline* permintaannya memiliki *middleware* yang berjalan untuk setiap permintaan (misalnya pembatasan laju login yang memperlambat percobaan masuk yang gagal berulang kali, yang diwajibkan oleh pekerjaan kepatuhan UU PDP-nya), lalu perutean, lalu *handler*. *Handler*-nya ramping: *handler* dasbor mengumpulkan hasil dari beberapa fungsi layanan lalu mengembalikannya, dan aturan bisnisnya, misalnya kapan langganan diperpanjang atau bagaimana sebuah unggahan komunitas disaring dari tautan yang tidak aman, tinggal di modulnya sendiri tempat pengujian dapat menjangkaunya. Halaman publiknya dirender dari templat dengan *escaping* otomatis, hal yang penting di situs tempat pengguna mengunggah teks yang dibaca pengguna lain. Kata-katanya milik Python; arsitekturnya adalah yang baru saja Anda bangun.

3.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Controller yang tidak dapat ditemukan kerangka kerja. Kelasnya harus publik, tidak abstrak, dan namanya harus berakhiran `Controller` (atau membawa `[Controller]`). `ServiceController` dan `ServicesController` adalah dua controller berbeda dengan URL bawaan yang berbeda; tetapkan salah satunya.

View yang tidak dapat ditemukan. Pesan galatnya mendaftar setiap tempat yang dicari, misalnya `/Views/Services/Detail.cshtml` dan `/Views/Shared/Detail.cshtml`. Bandingkan nama aksi dengan nama berkas huruf demi huruf; `Detail` bukan `Details`.

Mencampur rute konvensional dan *attribute route* pada satu controller. Begitu sebuah controller punya *attribute route*, rute konvensional tidak lagi mencapai aksi-aksinya. Pasang *attribute route* pada setiap aksi controller itu, seperti `ServicesController`.

***Middleware* dengan urutan yang salah.** Halaman kode status yang diletakkan sesudah perutean dan *endpoint* tidak pernah melihat responsnya; *middleware* pencatatan yang diletakkan di dalam *middleware* kode status mencatat setiap galat dua kali. Tulis *pipeline* dari luar ke dalam: pencatatan dan penanganan galat lebih dulu, lalu halaman status, lalu perutean, lalu *endpoint*.

Membaca *query string* secara manual. `Request.Query["quantity"]` berjalan, tetapi melompati konversi, `ModelState`, dan perjalanan bolak-balik *tag helper*. Deklarasikan parameter atau kelas model dan biarkan *model binding* mengerjakannya.

Aturan bisnis di controller atau view. Jika angka seperti berat minimal atau biaya muncul di controller atau berkas `.cshtml`, pindahkan ke model. Tolok ukurnya: dapatkah sebuah *endpoint* JSON dan sebuah *unit test* memakai aturan itu tanpa permintaan HTTP? Jika tidak, aturan itu berada di tempat yang salah.

`@Html.Raw` untuk "memperbaiki" masalah tampilan. Ia mematikan *encoding* Razor. Pakailah hanya untuk HTML yang dihasilkan dan dipercaya program Anda sendiri, jangan pernah untuk apa pun yang berasal dari pengguna atau dari isian basis data yang dapat disunting pengguna.

3.12 Ringkasan Bab

- Aplikasi ASP.NET Core membangun *host*, mendaftarkan layanan di container DI, membangun *pipeline* permintaan dari *middleware*, dan menjalankan Kestrel. `Program.cs` adalah persis langkah-langkah itu, dengan urutan itu.
- *Middleware* membungkus *middleware*; masing-masing dapat bertindak sebelum dan sesudah sisa *pipeline*, sehingga urutan itu penting. LaundryMvc mencatat, menangani *exception* di luar Development, menjalankan ulang status galat sebagai halaman ramah, merutekan, dan melaksanakan *endpoint*.
- Model menyimpan data dan aturan: *record* untuk layanan dan gerai, `PriceCalculator` untuk aturan toko, *view model* yang dibentuk untuk view, model formulir untuk *binding*, dan akses data di balik antarmuka `ILaundryCatalog` yang didaftarkan sebagai *singleton*.
- Controller dicapai melalui rute konvensional (`{controller=Home}/{action=Index}/{id?}`) atau melalui *attribute route* dengan prefiks, batasan, dan prioritas. Aksi mengembalikan hasil: view, 404 dengan atau tanpa badan, JSON, pengalihan.
- *Model binding* mengisi parameter aksi dan model formulir dari rute, *query string*, dan formulir; galat konversi serta pemeriksaan milik aksi terkumpul di `ModelState`.
- View Razor memadukan HTML dan C# dengan `@`, meng-*encode* setiap nilai secara bawaan, berbagi satu *layout* dengan *section*, memakai ulang *partial view*, dan memakai *tag helper* untuk menghasilkan URL dari rute dan mengikat isian formulir ke model.
- Situs yang berjalan sepakat dengan hitungan manual, setiap rute menjawab seperti yang diramalkan, dan satu-satunya kejutan, "2 item", menjadi soal latihan.

3.13 Soal Latihan

1. Buat penawaran menyebut "2 items" dan "1 item", tetapi tetap "3 kg" dan "2.5 kg". Ubah sesedikit mungkin berkas.

2. Tambahkan batasan `length(5)` pada rute `Details`, agar `/services/S04` tidak pernah mencapai aksinya. Status apa yang didapat `/services/S04`, dan apa yang ditampilkan log?

3. Tambahkan aksi `Cheapest` di `/services/cheapest` yang menampilkan layanan termurah dengan view `Details`. Apakah berpengaruh jika Anda menulisnya di atas atau di bawah `Details` di dalam berkas? Bandingkan dengan Soal 1 di Bab 2.

4. Pindahkan *middleware* pencatatan di `Program.cs` agar berada *sesudah* `UseStatusCodePagesWithReExecute`. Minta `/services/S04` (setelah Soal 2) dan `/nothing/here`. Bagaimana log berubah, dan mengapa?

5. Tanpa menjalankan program, tuliskan `href` yang dihasilkan masing-masing *tag helper* berikut, dan jelaskan perbedaannya: `<a asp-controller="Services" asp-action="Details" asp-route-id="S0004">` dan `<a asp-controller="Outlets" asp-action="Details" asp-route-id="O03">`.

6. Tambahkan aksi di `/services/boom` yang melempar `InvalidOperationException("Boom")`. Apa yang dikembalikan permintaan ke aksi itu ketika aplikasi dijalankan dengan `dotnet run`, dan apa ketika dijalankan dengan `ASPNETCORE_ENVIRONMENT=Production dotnet run --no-launch-profile`? Mengapa perintah kedua memerlukan `--no-launch-profile`? Hapus aksi itu sesudahnya.

7. Bu Wati memutuskan antar-jemput gratis mulai Rp 150.000, bukan Rp 100.000. Berkas mana yang berubah? Menjadi apa penawaran untuk 1,5 setelan jas (keluaran di atas)?

3.14 Penyelesaian Soal Latihan

1. Hanya model dan satu baris di satu view yang berubah. Tambahkan metode ke `Service`:

```
    // "1 item", "2 items", but "2.5 kg" (units such as kg have no plural)
    public string UnitFor(decimal quantity) =>
        PerKg || quantity == 1 ? Unit : Unit + "s";
```

dan di `Quote.cshtml` tulis `@q.Service.UnitFor(q.Billed)` alih-alih `@q.Service.Unit`. Menjalankan persis perubahan ini menjawab `2 items x Rp 65.000` untuk 1,5 setelan jas dan `1 item x Rp 65.000` untuk satu setelan; kilogram tidak berubah. Menaruh aturannya di `Service` alih-alih di view berarti setiap halaman yang menampilkan jumlah dapat memakainya.

2. Tulis `[HttpGet("{id:length(5)}")]`. `/services/S04` kini tidak cocok dengan rute mana pun, statusnya 404, dan halaman ramah ditampilkan; `/services/S0004` tetap berjalan. Log menampilkan `GET /services/S04 -> 404 in 0 ms`: aksinya tidak pernah dijalankan, karena router menolak alamat itu. Sebelum perubahan, aksinya berjalan, mencari di katalog dengan sia-sia, lalu mengembalikan `NotFound()`; pengunjung melihat halaman yang sama, tetapi pekerjaannya dilakukan percuma.

3.

```
    [HttpGet("cheapest")]
    public IActionResult Cheapest() =>
        View("Details", catalog.Services().MinBy(s => s.Price));
```

`/services/cheapest` menampilkan “Ironing only” (Rp 6.000 per kg), di mana pun metode itu berada di dalam berkas, seperti ditegaskan sebuah percobaan. ASP.NET Core menempatkan literal `cheapest` di atas parameter `{id}` [9]. Di MiniFrame (Bab 2, Soal 1) urutan pendaftaranlah yang menentukan, dan urutan yang salah mengirim “cheapest” ke `Details` sebagai kode layanan.

4. Dengan *middleware* pencatatan di dalam *middleware* kode status, setiap galat muncul *dua kali* di log. Baris-baris nyatanya:

```
      GET /services/S04 -> 404 in 0 ms
      GET /error/404 -> 404 in 13 ms
      GET /nothing/here -> 404 in 0 ms
      GET /error/404 -> 404 in 3 ms
```

Eksekusi ulang menjalankan lagi bagian *pipeline sesudah middleware* kode status, dengan path `/error/404`; pencatat kita kini berada di bagian itu, sehingga ia mencatat permintaan aslinya lalu permintaan yang dijalankan ulang. Di luar, ia melihat setiap permintaan sekali, dengan path yang diketik pengunjung. Untuk log permintaan, di luar adalah tempat yang benar; untuk mengukur lamanya halaman galat dirender, di dalam bisa berguna.

5. Yang pertama memberi `href="/services/S0004"`, yang kedua `href="/Outlets/Details/O03"`. *Tag helper* meminta sistem perutean memberikan URL yang akan mencapai aksi yang disebut. `ServicesController.Details` hanya dapat dicapai melalui *attribute route*-nya `services/{id}`; `OutletsController.Details` hanya melalui rute konvensional `{controller}/{action}/{id?}`, yang mengisi segmennya dengan nama controller dan aksi seperti tertulis di kelas. Itu jugalah sebabnya tautan menu di *layout* berbunyi `/Outlets` dan `/Home/About` tetapi `/services`.

6. Dengan `dotnet run`, profil peluncuran mengisi `ASPNETCORE_ENVIRONMENT=Development`, dan responsnya berstatus 500 dengan *developer exception page*; jika diminta `curl`, yang tidak meminta HTML, isinya teks polos yang diawali `System.InvalidOperationException: Boom`, diikuti *stack trace*-nya. Dengan perintah Production, responsnya berstatus 500 dengan halaman kita "Something went wrong" dan "status 500", yang dihasilkan `UseExceptionHandler("/error/500")`, dan log memuat baris `fail:` dari `ExceptionHandlerMiddleware` beserta *exception*-nya. Opsi `--no-launch-profile` diperlukan karena variabel lingkungan dari profil peluncuran menimpa variabel dari terminal; tanpa opsi itu, pengaturan Development dari perintah pertama akan menang. Kedua perilaku diamati di mesin *build*.

7. Tepat satu baris yang berubah: `FreePickupFrom = 150_000m` di `PriceCalculator.cs`. Catatan di halaman penawaran dibangun dari konstanta itu, sehingga berubah dengan sendirinya menjadi "Pickup and delivery are free from Rp 150.000." Untuk 1,5 setelan jas subtotalnya tetap Rp 130.000, yang kini di bawah ambang, sehingga biaya antar-jemput menjadi Rp 10.000, totalnya Rp 140.000, dan catatannya berbunyi "Pickup and delivery cost Rp 10.000." Tidak ada yang berubah di controller atau view mana pun; aturan pertama Bu Wati terjaga.

Daftar Pustaka

[1] Microsoft, "ASP.NET Core middleware," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/fundamentals/middleware/

[2] Microsoft, "Overview of ASP.NET Core MVC," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/mvc/overview

[3] Microsoft, "Dependency injection in ASP.NET Core," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/fundamentals/dependency-injection

[4] Microsoft, "Use multiple environments in ASP.NET Core," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/fundamentals/environments

[5] Microsoft, "Handle errors in ASP.NET Core," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/fundamentals/error-handling

[6] Microsoft, "Static files in ASP.NET Core," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/fundamentals/static-files

[7] Microsoft, "Routing to controller actions in ASP.NET Core," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/mvc/controllers/routing

[8] Microsoft, "Declare primary constructors for classes and structs," *C# Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/dotnet/csharp/whats-new/tutorials/primary-constructors

[9] Microsoft, "Routing in ASP.NET Core," *ASP.NET Core Documentation*, for ASP.NET Core 10.0, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/fundamentals/routing

[10] Microsoft, "Model binding in ASP.NET Core," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/mvc/models/model-binding

[11] R. Fielding, M. Nottingham, and J. Reschke, "HTTP Semantics," RFC 9110, Internet Engineering Task Force, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9110

[12] Microsoft, "Razor syntax reference for ASP.NET Core," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/mvc/views/razor

[13] Microsoft, "Layout in ASP.NET Core," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/mvc/views/layout

[14] Microsoft, "Partial views in ASP.NET Core," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/mvc/views/partial

[15] Microsoft, "Tag Helpers in ASP.NET Core," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/mvc/views/tag-helpers/intro

Bab 4

Evaluasi dan Pembahasan I: Situs Etalase Profitina Laundry, Diselesaikan Tuntas

Bab ini tidak menambahkan bagian baru ASP.NET Core. Sebaliknya, bab ini mengambil apa yang dibangun Bab 1 sampai 3 (*pipeline* permintaan, perutean, controller, injeksi dependensi, view Razor, dan konfigurasi) lalu memakainya pada satu proyek nyata yang kecil tetapi lengkap: situs web etalase yang ditunjukkan Profitina Laundry kepada calon investor, mitra, dan calon karyawan, dengan satu halaman tempat pelanggan dapat melacak pesanannya. Proyek itu dipecah menjadi lima soal. Dua soal pertama membangun situs dan halaman pelacakannya; tiga lainnya menelisik mesin di bawahnya: meramalkan tabel rute secara manual, menambahkan header respons dari *middleware*, dan mengalihkan seluruh situs ke mode pemeliharaan tanpa merusak lembar gayanya. Setiap soal diselesaikan tuntas, dalam tujuh bagian yang sama: soalnya, memahaminya secara manual, merancangnya di atas kertas, penyelesaiannya, menjalankannya, penyelesaian keliru yang menggoda beserta keluaran nyatanya, dan sebuah variasi untuk menguji diri. Pada akhir bab ini Anda akan memiliki rutinitas untuk menghadapi soal kerangka kerja apa pun, sebuah aplikasi jadi yang dapat Anda sesuaikan dengan usaha Anda sendiri, dan pengalaman langsung dengan lima kesalahan yang pernah dibuat setiap pengembang ASP.NET Core sekali dalam hidupnya.

4.1 Posisi Kita Sekarang

Sebelum menyelesaikan apa pun, mari kita gelar perkakasnya. Setiap soal di bab ini meraih segelintir perkakas dari tiga bab pertama, dan menyebut nama perkakasnya sudah separuh pekerjaan.

Dari **Bab 1** Anda punya meja kerja (.NET 10 SDK, VS Code, `dotnet new`, `dotnet run`, `curl`) dan gambaran besarnya: kerangka kerja memanggil kode Anda; setiap permintaan ditunggu, dirutekan, diekstrak, diambil datanya, diputuskan, dirender, dijawab, dan dipertahankan; dan Model–View–Controller membagi bagian Anda dari pekerjaan itu.

Dari **Bab 2** Anda mengenal keempat fondasi dari dalam: router yang mencocokkan templat segmen demi segmen dan menerapkan batasan; container yang membangun objek dari pendaftaran; pemeta yang mengubah baris menjadi objek; mesin templat yang meng-*escape* secara bawaan dan membungkus halaman dengan *layout*.

Dari **Bab 3** Anda memiliki ASP.NET Core MVC itu sendiri: `Program.cs` dengan layanan lebih dulu dan *pipeline* sesudahnya; *middleware* yang bertindak sebelum dan sesudah `await next`; rute konvensional dan *attribute route* dengan batasan dan prioritas; aksi yang mengembalikan view, 404, dan JSON; *model binding* dan `ModelState`; view Razor dengan *layout*, *partial view*, *section*, dan *tag helper*; serta halaman kode status dan *exception*.

Tabel 4.1 memperlihatkan perkakas mana yang paling dilatih setiap soal. Pakailah sebagai peta: jika *middleware* masih terasa samar, mulailah dari Soal 4.4 dan 4.5; jika perutean yang samar, mulailah dari 4.3.

Tabel 4.1 Kelima soal bab ini dan perkakas dari Bab 1 sampai 3 yang paling dilatih masing-masing.

Soal	Topik	Perkakas utama dari Bab 1–3
4.1	Situs etalase	*attribute route* dengan prefiks, *layout*, DI untuk konten dan data, *view model*
4.2	Di mana cucian saya?	batasan rute `regex`, formulir `GET`, `RedirectToAction`, halaman 404, privasi
4.3	Tabel rute secara manual	pencocokan rute, prioritas, huruf besar-kecil, rute konvensional lawan *attribute route*
4.4	Header waktu respons	*middleware*, saat respons dimulai, `OnStarting`
4.5	Mode pemeliharaan	konfigurasi dan variabel lingkungan, urutan *middleware*, `UseRouting` dan *endpoint*

4.2 Cara Menghadapi Soal Kerangka Kerja

Pemula yang diberi soal kerangka kerja biasanya membuka penyunting dan mulai membuat controller. Pengembang berpengalaman lebih dulu menuliskan *alamat-alamat*. Bedanya bukan bakat, melainkan rutinitas, dan **Gambar 4.1** memperlihatkan rutinitas yang diikuti bab ini untuk setiap soal. Rutinitas ini adalah cara memecahkan masalah yang diuraikan George Pólya untuk matematika (pahami, rencanakan, laksanakan, tinjau kembali) [1], yang disesuaikan untuk aplikasi web.



Gambar 4.1 Enam langkah yang dipakai untuk setiap soal di bab ini; panah putus-putus adalah gelung antara menjalankan program dan memperbaikinya.

Langkah 1: Baca dan rumuskan ulang. Baca soalnya dua kali dan tuliskan dalam satu atau dua kalimat apa yang diminta pengunjung dan apa yang harus dijawab situs. Garis bawahi setiap alamat dan setiap aturan; itulah pengujian pertama Anda.

Langkah 2: Tulis tabel alamat. Untuk setiap halaman, tuliskan metode, alamat, aksi yang akan menanganinya, dan status yang harus dikembalikannya, termasuk alamat-alamat yang harus *gagal* (kode yang salah, tipe yang salah, metode yang salah). Tabel ini adalah spesifikasi perutean, dan pada langkah 5 ia menjadi daftar hal yang diperiksa.

Langkah 3: Sketsa halaman dan data. Gambarlah setiap halaman sebagai kotak berisi apa yang ditampilkannya, dan tulis di sampingnya asal setiap bagian: teks tetap (konten), angka yang dihitung dari data (model), atau permintaan (*binding*). Segala yang dihitung tempatnya di model atau di sebuah layanan, bukan di view.

Langkah 4: Tulis controller yang ramping, view, dan aturannya. Controller merutekan, bertanya, dan memilih; aturan masuk ke kelas yang disediakan container; view menampilkan. Tulis satu aksi dan view-nya setiap kali, dan jalankan setelah masing-masing.

Langkah 5: Jalankan dan uji. Jalankan aplikasinya dan tanyakan setiap alamat dari tabel langkah 2, di peramban dan dengan `curl -i`, yang menampilkan status dan header. Ketika sebuah hasil berbeda dari tabel, jangan menebak: baca log dan temukan tempat pertama kenyataan berbeda dari rencana. Itulah gelung putus-putus pada **Gambar 4.1**.

Langkah 6: Tinjau kembali. Ajukan tiga pertanyaan. Alamat mana yang belum saya coba? Apa cara keliru yang paling menggoda untuk menyelesaikan soal ini, dan apa tepatnya yang terjadi jika saya mencobanya? Dapatkah saya menyelesaikan versi yang sedikit berbeda? Itulah dua bagian terakhir setiap soal di bawah.

> **WAWASAN KUNCI: MENGAPA TABEL ALAMAT DIBUAT SEBELUM KODE.** Dari luar, aplikasi web tidak lain adalah sekumpulan alamat beserta jawabannya. Jika Anda tidak dapat menulis tabelnya, Anda belum tahu apa yang harus dibangun; jika dapat, perutean hampir menulis dirinya sendiri, dan memeriksa situs yang sudah jadi tinggal menyusuri daftarnya.

Kode kelima soal adalah satu aplikasi, `Chapter04/aspnet/LaundryShowcase`, yang tumbuh dari soal ke soal; kelima penyelesaian keliru yang menggoda adalah proyek-proyek mungil terpisah di `Chapter04/aspnet/Tempting`, agar setiap kesalahan dapat dilihat tersendiri. Semuanya di-*build* dengan .NET 10 SDK dan dijalankan; `Chapter04/verify.sh` menjalankan penyelesaian dan setiap penyelesaian keliru, menanyakan setiap alamat bab ini dengan `curl`, dan mencetak jawabannya, dan setiap keluaran di bawah berasal dari skrip itu. Seperti aplikasi di Bab 3, LaundryShowcase hanya memakai *shared framework* ASP.NET Core.

4.3 Soal 4.1: Situs Etalase

4.3.1 Soalnya

> **SOAL 4.1.** Profitina Laundry menginginkan situs web yang memperkenalkan usahanya kepada orang-orang yang mungkin berinvestasi, bekerja sama, atau bekerja di sana. Situs itu harus punya halaman beranda, profil usaha, uraian tentang perusahaan, uraian layanannya, dan halaman yang menjelaskan apa yang ditawarkan dan diminta usaha itu dari investor, mitra, dan calon karyawan. Alamatnya harus: `/laundry` untuk beranda, `/laundry/profile`, `/laundry/company`, `/laundry/services`, dan `/laundry/collaboration`. Akar situs `/` harus mengarah ke beranda. Semua halaman berbagi satu tampilan. Kata-kata di situs harus dapat disunting tanpa kompilasi ulang, dan setiap angka yang ditampilkan harus berasal dari data toko, bukan diketik manual.

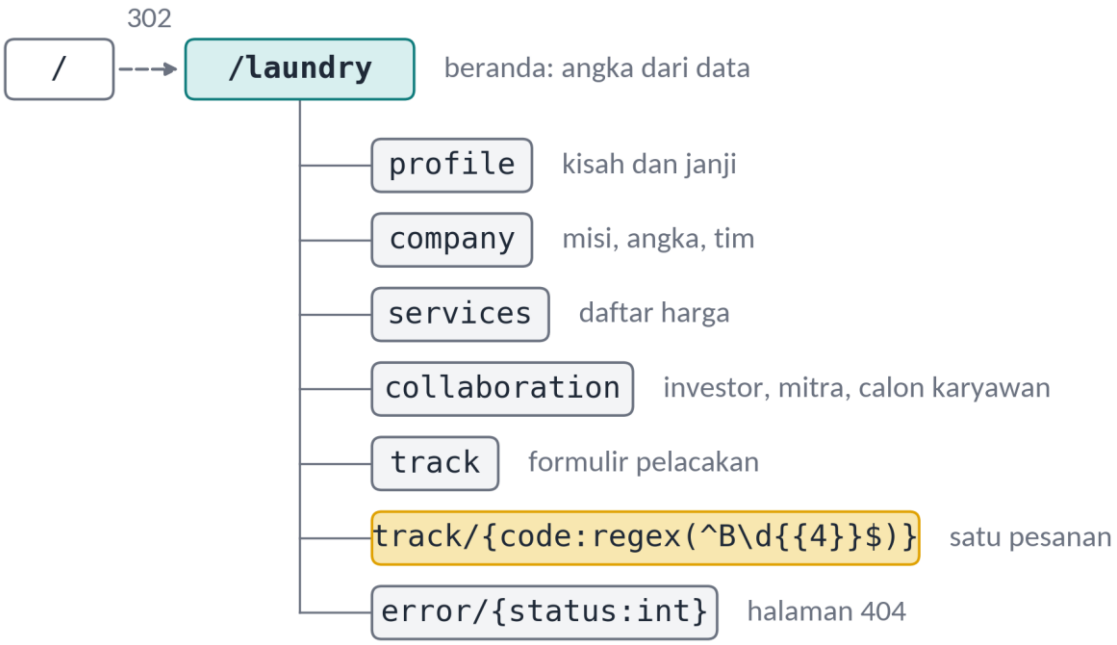
4.3.2 Memahaminya secara manual

Dengan kata lain: lima halaman di bawah satu prefiks bersama, satu bingkai bersama, teks dari berkas, angka dari data. Langkah 2 mengubah soalnya menjadi **Tabel 4.2**; tiga baris terakhir adalah alamat yang *tidak boleh* berfungsi, yang jarang disebut dalam soal tetapi harus selalu ditambahkan pengembang.

Tabel 4.2 Tabel alamat Soal 4.1, termasuk tiga alamat yang menguji batas-batasnya.

Metode	Alamat	Aksi	Yang diharapkan
GET	`/`	pengalihan	302 ke `/laundry`
GET	`/laundry`	`Home`	200, slogan dan hitungan terkini
GET	`/laundry/profile`	`Profile`	200, kisah dan janji
GET	`/laundry/company`	`Company`	200, misi, angka, tim
GET	`/laundry/services`	`Services`	200, daftar harga
GET	`/laundry/collaboration`	`Collaboration`	200, tiga khalayak
GET	`/profile`	tidak ada	404: halaman berada di bawah `/laundry`
GET	`/Showcase/Profile`	tidak ada	404: tidak ada rute konvensional
GET	`/laundry/Profile`	`Profile`	200: pencocokan tidak membedakan huruf besar-kecil

Gambar 4.2 menggambar tabel yang sama sebagai peta situs, bersama alamat pelacakan yang ditambahkan Soal 4.2.



Gambar 4.2 Peta situs LaundryShowcase: setiap alamat adalah *attribute route* dari satu controller di bawah prefiks `/laundry`; rute pelacakan yang berbatasan milik Soal 4.2.

4.3.3 Merancangnya di atas kertas

Perutean. Semua halaman berbagi prefiks `laundry`, dan itulah yang dinyatakan `[Route("laundry")]` pada sebuah controller. Satu controller, `ShowcaseController`, dengan satu aksi per halaman, sudah cukup; lima controller yang masing-masing berisi satu aksi akan menyebarkan satu gagasan ke lima berkas. Akar situs bukan halaman etalase melainkan pintu menuju ke sana, jadi cukup berupa pengalihan satu baris dengan *minimal API* di `Program.cs`. Aplikasi mendaftarkan `MapControllers()` (hanya *attribute route*) dan tidak ada rute konvensional, sehingga `/Showcase/Profile` tidak mungkin ada.

Konten. "Dapat disunting tanpa kompilasi ulang" berarti kata-katanya tinggal di berkas yang dibaca aplikasi saat awal jalan: `content/site.json`, yang dideserialisasi menjadi *record* C#

dengan `System.Text.Json`, bagian dari .NET. Objek konten didaftarkan di container sebagai *singleton*, sehingga setiap controller dan view dapat menerimanya.

Data. "Angka dari data" berarti membaca ekspor toko (`Service.csv`, `Outlet.csv`, `Bill.csv`) sekali dan menghitung faktanya: jumlah pesanan, tanggal pesanan pertama dan terakhir, rata-rata pesanan, dan pesanan per gerai. Kelas `LaundryData` mengerjakan keduanya, juga didaftarkan sebagai *singleton*.

View. Satu *layout* dengan menu (halaman aktif ditandai, seperti di Bab 3), dan satu view per aksi. *Layout* memerlukan nama situs di setiap halaman, jadi alih-alih meneruskannya dari setiap aksi, `_ViewImports.cshtml` menyuntikkan objek konten ke *setiap* view dengan `@inject SiteContent Site`.

4.3.4 Penyelesaiannya

Kode 4.1 adalah berkas kontennya. Teksnya dibuat pendek agar setiap baris muat; situs sungguhan akan punya paragraf yang lebih panjang, dan programnya tidak berubah.

Kode 4.1 `content/site.json`: setiap kata di etalase yang tidak dihitung dari data.

```
{
  "name": "Profitina Laundry",
  "tagline": "Clean clothes, honest prices, ready on the promised day.",
  "city": "Surabaya",
  "story": [
    "We began in January 2019 as one counter in Rungkut.",
    "Every order is weighed or counted in front of the customer.",
    "The notebook became a database; the promise stayed the same."
  ],
  "values": [
    {
      "title": "Weighed in front of you",
      "text": "The scale faces the customer, and the price follows from it."
    },
    {
      "title": "A rack for every order",
      "text": "Finished laundry waits on a numbered rack, found in seconds."
    },
    {
      "title": "The promised day",
      "text": "Express is ready the next day, regular in two to three days."
    }
  ],
  "mission": "Give every household a few hours back each week, at a fair price.",
  "team": [
    { "role": "Counter and transactions", "people": 5 },
    { "role": "Washing and drying", "people": 5 },
    { "role": "Ironing and folding", "people": 3 },
    { "role": "Owner and supervision", "people": 1 }
  ],
  "collaboration": [
    {
      "audience": "Investors",
      "offer": "Two more outlets in east Surabaya, same system, same prices.",
      "ask": "Capital for machines and the first six months of rent.",
      "contact": "invest@profitina-laundry.example"
    },
    {
      "audience": "Partners",
      "offer": "Boarding houses and clinics get a weekly pickup, one invoice.",
      "ask": "A fixed pickup day and one contact person.",
      "contact": "partners@profitina-laundry.example"
    },
    {
      "audience": "Job candidates",
      "offer": "Paid training at the counter or in the washing room.",
```

Kode 4.1 (Lanjutan) *content/site.json: setiap kata di etalase yang tidak dihitung dari data.*

```
      "ask": "Care, honesty and a good sense of time.",
      "contact": "jobs@profitina-laundry.example"
    }
  ]
}
```

Kode 4.2 mendeklarasikan *record* yang bentuknya mencerminkan JSON itu. Dengan `JsonSerializerOptions.Web`, deserializer-nya [2] mencocokkan nama *camelCase* di JSON (`tagline`) dengan properti C# (`Tagline`) dan membaca angka ke `int`. Perhatikan *record* `Opening`: versi pertama menamainya `Offer`, dan gagal dikompilasi karena ia juga punya properti bernama `Offer`, sedangkan sebuah anggota tidak boleh bernama sama dengan tipe yang melingkupinya.

Kode 4.2 `Models/SiteContent.cs`: bentuk `site.json` sebagai *record* C#.

```
namespace LaundryShowcase.Models;

// The words of the site, read from content/site.json at start-up.
public record SiteContent(string Name, string Tagline, string City,
    List<string> Story, List<Value> Values, string Mission,
    List<TeamRole> Team, List<Opening> Collaboration);

public record Value(string Title, string Text);
public record TeamRole(string Role, int People);
public record Opening(string Audience, string Offer, string Ask, string Contact);
```

Kode 4.3 memuat *record* data. `Service` dan `Outlet` sama dengan di Bab 3; `Order` adalah satu baris `Bill.csv`, tabel yang di basis data disebut *bill*. Komentar pada `Order` mencatat keputusan yang disengaja, yang dijelaskan Soal 4.2: pelanggan sama sekali bukan bagian dari *record* ini. `ShopFacts` adalah *view model* untuk angka-angkanya.

Kode 4.3 `Models/Records.cs`: layanan, gerai, pesanan tanpa pelanggan, dan fakta yang dihitung darinya.

```
namespace LaundryShowcase.Models;

public record Service(string Id, string Description, decimal Price)
{
    public string Name => Description.Contains(" (")
        ? Description[..Description.LastIndexOf(" (")] : Description;

    public string Unit => Description.EndsWith("(per kg)") ? "kg"
        : Description.EndsWith("(per pair)") ? "pair" : "item";
}

public record Outlet(string Id, string Name, string Address, string Phone,
    DateOnly Opened);

// One row of data/Bill.csv. The customer is deliberately left out:
// the tracking page must not tell a stranger whose laundry this is.
public record Order(string Id, DateTime Arrived, DateTime? Finished,
    DateTime? Departed, string Status, string Rack, int Units,
    decimal Payment, string ServiceId, string OutletId);

// Numbers the profile and company pages compute from the data.
public record ShopFacts(int Orders, int Outlets, int Services,
    DateOnly FirstOrder, DateOnly LastOrder, decimal AverageTicket,
    Dictionary<string, int> OrdersPerOutlet);
```

`Money.Rp` adalah fungsi bantu yang sama dengan di Bab 3 dan tidak diulang di sini. **Kode 4.4** membaca ketiga berkas sekali dan menghitung faktanya. Pesanan disimpan dalam kamus yang dikunci dengan kode dan pembanding yang tidak membedakan huruf besar-kecil, sehingga mencari `b0072` menemukan `B0072` dalam satu langkah. `Facts()` memakai LINQ: `Min` dan `Max` tanggal kedatangan, `Average` pembayaran yang dibulatkan ke rupiah utuh, dan `GroupBy` atas kode gerai untuk menghitung pesanan per gerai.

Kode 4.4 `Data/LaundryData.cs`: data toko, dimuat sekali, dan fakta yang ditampilkan halaman.

```
using System.Globalization;
using System.Text;
using LaundryShowcase.Models;

namespace LaundryShowcase.Data;

// Loads the three CSV files once and answers questions about them.
public class LaundryData
{
    private static readonly CultureInfo Inv = CultureInfo.InvariantCulture;

    public IReadOnlyList<Service> Services { get; }
    public IReadOnlyList<Outlet> Outlets { get; }
    private readonly Dictionary<string, Order> orders;

    public LaundryData(string folder)
    {
        Services = Csv(Path.Combine(folder, "Service.csv"))
            .Select(f => new Service(f[0], f[1], decimal.Parse(f[2], Inv)))
            .ToList();
        Outlets = Csv(Path.Combine(folder, "Outlet.csv"))
            .Select(f => new Outlet(f[0], f[1], f[2], f[3],
                DateOnly.Parse(f[4], Inv)))
            .ToList();
        orders = Csv(Path.Combine(folder, "Bill.csv"))
            .Select(f => new Order(f[0], DateTime.Parse(f[1], Inv), When(f[2]),
                When(f[3]), f[4], f[5], int.Parse(f[6], Inv),
                decimal.Parse(f[7], Inv), f[10], f[11]))
            .ToDictionary(o => o.Id, StringComparer.OrdinalIgnoreCase);
    }

    public Order? FindOrder(string code) =>
        orders.TryGetValue(code, out var order) ? order : null;

    public Service? FindService(string id) =>
        Services.FirstOrDefault(s => s.Id == id);

    public Outlet? FindOutlet(string id) =>
        Outlets.FirstOrDefault(o => o.Id == id);
```

Kode 4.4 (Lanjutan) *Data/LaundryData.cs: data toko, dimuat sekali, dan fakta yang ditampilkan halaman.*

```
    public ShopFacts Facts()
    {
        var all = orders.Values;
        return new ShopFacts(all.Count, Outlets.Count, Services.Count,
            DateOnly.FromDateTime(all.Min(o => o.Arrived)),
            DateOnly.FromDateTime(all.Max(o => o.Arrived)),
            Math.Round(all.Average(o => o.Payment)),
            all.GroupBy(o => o.OutletId)
               .ToDictionary(g => FindOutlet(g.Key)?.Name ?? g.Key,
                             g => g.Count()));
    }

    private static DateTime? When(string text) =>
        text == "" ? null : DateTime.Parse(text, Inv);

    // Splits each line on commas, except commas inside "double quotes".
    private static IEnumerable<string[]> Csv(string path)
    {
        foreach (var line in File.ReadLines(path).Skip(1))
        {
            var fields = new List<string>();
            var field = new StringBuilder();
            bool quoted = false;
            foreach (char c in line)
            {
                if (c == '"') quoted = !quoted;
                else if (c == ',' && !quoted)
                {
                    fields.Add(field.ToString());
                    field.Clear();
                }
                else field.Append(c);
            }
            fields.Add(field.ToString());
            yield return fields.ToArray();
        }
    }
}
```

Kode 4.5 adalah `Program.cs` yang lengkap, termasuk bagian-bagian yang ditambahkan Soal 4.3 sampai 4.5; masing-masing dijelaskan di soalnya sendiri. Untuk Soal 4.1, baris-baris pentingnya adalah kedua pendaftaran, pengalihan dari `/`, dan `MapControllers()`. Konten dibaca dan dideserialisasi sekali saat awal jalan lalu didaftarkan sebagai *instance*, bentuk kerangka kerja sungguhan dari `container.Instance(...)` milik MiniFrame.

Kode 4.5 `Program.cs` milik LaundryShowcase: konten dan data didaftarkan sebagai *singleton*, *pipeline middleware*, sebuah pengalihan, controller, dan daftar rute khusus pengembangan.

```
using System.Text.Json;
using LaundryShowcase.Data;
using LaundryShowcase.Middleware;
using LaundryShowcase.Models;

var builder = WebApplication.CreateBuilder(args);
var root = builder.Environment.ContentRootPath;

// Services
builder.Services.AddControllersWithViews();
builder.Services.AddSingleton(JsonSerializer.Deserialize<SiteContent>(
    File.ReadAllText(Path.Combine(root, "content", "site.json")),
    JsonSerializerOptions.Web)!);
builder.Services.AddSingleton(new LaundryData(Path.Combine(root, "data")));

var app = builder.Build();

// Pipeline
app.UseMiddleware<ResponseTimeMiddleware>();
app.UseStatusCodePagesWithReExecute("/laundry/error/{0}");
app.UseRouting();

// Maintenance mode: after UseRouting, so the chosen endpoint is known and
// the style sheet can still be served while every page is closed.
app.Use(async (context, next) =>
{
    bool closed = app.Configuration.GetValue<bool>("Showcase:Maintenance");
    var endpoint = context.GetEndpoint() as RouteEndpoint;  // set by routing
    bool isAsset = endpoint?.RoutePattern.RawText?.StartsWith("css/") == true;
    if (closed && !isAsset)
    {
        context.Response.StatusCode = StatusCodes.Status503ServiceUnavailable;
        context.Response.Headers.RetryAfter = "3600";
        context.Response.ContentType = "text/html; charset=utf-8";
        await context.Response.WriteAsync("""
            <!DOCTYPE html><html lang="en"><head><meta charset="utf-8">
            <title>Back soon - Profitina Laundry</title>
            <link rel="stylesheet" href="/css/site.css"></head>
            <body><main><h1>Back soon</h1>
            <p>The website is being updated. The outlets are open as usual.</p>
            </main></body></html>
            """);
        return;
    }
    await next(context);
});
```

Kode 4.5 (Lanjutan) *Program.cs milik LaundryShowcase: konten dan data didaftarkan sebagai singleton, pipeline middleware, sebuah pengalihan, controller, dan daftar rute khusus pengembangan.*

```
app.MapStaticAssets();
app.MapGet("/", () => Results.Redirect("/laundry"));
app.MapControllers();

if (app.Environment.IsDevelopment())
{
    // Lists every route the application knows (Problem 4.3).
    app.MapGet("/_routes", (EndpointDataSource source) => string.Join("\n",
        source.Endpoints.OfType<RouteEndpoint>()
            .Where(e => !e.RoutePattern.RawText!.StartsWith("css/"))
            .OrderBy(e => e.RoutePattern.RawText)
            .Select(e => $"{e.RoutePattern.RawText,-40} " + e.DisplayName?
                .Replace("LaundryShowcase.Controllers.", "").Split(" (")[0])));
}

app.Run();
```

Kode 4.6 adalah satu-satunya controller. Setiap aksi halaman hanya satu baris: ia menyerahkan kepada view model yang diperlukannya (fakta, layanan, atau daftar kerja sama) dan tidak lebih. Aksi pelacakan dan galat milik Soal 4.2.

Kode 4.6 `Controllers/ShowcaseController.cs`: satu controller ber-*attribute route* untuk seluruh etalase.

```
using LaundryShowcase.Data;
using LaundryShowcase.Models;
using Microsoft.AspNetCore.Diagnostics;
using Microsoft.AspNetCore.Mvc;

namespace LaundryShowcase.Controllers;

[Route("laundry")]
public class ShowcaseController(SiteContent site, LaundryData data)
    : Controller
{
    [HttpGet("")]                                   // /laundry
    public IActionResult Home() => View(data.Facts());

    [HttpGet("profile")]                            // /laundry/profile
    public IActionResult Profile() => View(data.Facts());

    [HttpGet("company")]                            // /laundry/company
    public IActionResult Company() => View(data.Facts());

    [HttpGet("services")]                           // /laundry/services
    public IActionResult Services() => View(data.Services);

    [HttpGet("collaboration")]                      // /laundry/collaboration
    public IActionResult Collaboration() => View(site.Collaboration);

    [HttpGet("track")]                              // the form, and its answer
    public IActionResult TrackForm(string? code)
    {
        if (string.IsNullOrWhiteSpace(code))
        {
            return View();
        }
        return RedirectToAction(nameof(Track), new { code = code.Trim() });
    }
```

Kode 4.6 (Lanjutan) *Controllers/ShowcaseController.cs: satu controller ber-attribute route untuk seluruh etalase.*

```
    [HttpGet(@"track/{code:regex(^B\d{{4}}$)}")]  // /laundry/track/B0072
    public IActionResult Track(string code)
    {
        var order = data.FindOrder(code);
        if (order is null)
        {
            return NotFound();
        }
        var service = data.FindService(order.ServiceId);
        ViewData["Service"] = service?.Name;
        ViewData["Unit"] = service?.Unit;
        ViewData["Outlet"] = data.FindOutlet(order.OutletId)?.Name;
        return View(order);
    }

    [HttpGet("error/{status:int}")]
    public IActionResult Error(int status)
    {
        var original = HttpContext.Features.Get<IStatusCodeReExecuteFeature>();
        ViewData["Path"] = original?.OriginalPath ?? Request.Path.Value;
        Response.StatusCode = status;
        return View(status);
    }
}
```

Kode 4.7 membuat konten tersedia bagi setiap view, dan **Kode 4.8** adalah *layout*-nya. Karena setiap aksi milik controller yang sama, fungsi `Here` pada menu hanya membandingkan nama aksi, dan `asp-action` tanpa `asp-controller` berarti "aksi milik controller saat ini".

Kode 4.7 `Views/_ViewImports.cshtml`: `@inject` menaruh konten situs ke setiap view.

```
@using LaundryShowcase
@using LaundryShowcase.Models
@inject SiteContent Site
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Kode 4.8 `Views/Shared/_Layout.cshtml`: bingkai dan menu etalase.

```
@{
    string? Here(string action) =>
        ViewContext.RouteData.Values["action"] as string == action
            ? "here" : null;
}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - @Site.Name</title>
    <link rel="icon" href="data:," />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
</head>
<body>
    <header>
        <a class="brand" asp-action="Home">@Site.Name</a>
        <nav>
            <a class="@Here("Home")" asp-action="Home">Home</a>
            <a class="@Here("Profile")" asp-action="Profile">Profile</a>
            <a class="@Here("Company")" asp-action="Company">Company</a>
            <a class="@Here("Services")" asp-action="Services">Services</a>
            <a class="@Here("Collaboration")"
               asp-action="Collaboration">Work with us</a>
            <a class="@Here("TrackForm")" asp-action="TrackForm">Track</a>
        </nav>
    </header>
    <main>
        @RenderBody()
    </main>
    <footer>
        <p>@Site.Name &middot; @Site.City &middot; @Site.Tagline</p>
    </footer>
</body>
</html>
```

Kelima view halaman menyusul: **Kode 4.9**, **Kode 4.10**, **Kode 4.11**, **Kode 4.12**, dan **Kode 4.13**. Isinya tidak memuat angka maupun aturan, hanya di mana menampilkan apa: `@Model.Orders.ToString("N0")` memformat 2953 menjadi “2,953”, dan di **Kode 4.11** gelung `@foreach (var (outlet, count) in Model.OrdersPerOutlet)` memecah setiap entri kamus menjadi dua bagiannya.

Kode 4.9 `Views/Showcase/Home.cshtml`: slogan dan hitungan yang dihitung dari data.

```
@model ShopFacts
@{
    ViewData["Title"] = "Home";
}
<section class="hero">
    <h1>@Site.Tagline</h1>
    <p>@Model.Outlets outlets in @Site.City, @Model.Services services,
       and @Model.Orders.ToString("N0") orders since
       @Model.FirstOrder.ToString("MMMM yyyy").</p>
    <p>
        <a class="button" asp-action="Services">See the prices</a>
        <a class="button" asp-action="TrackForm">Track an order</a>
        <a class="button light" asp-action="Collaboration">Work with us</a>
    </p>
</section>
```

Kode 4.10 `Views/Showcase/Profile.cshtml`: kisah dan tiga janji, dari `site.json`.

```
@model ShopFacts
@{
    ViewData["Title"] = "Profile";
}
<h1>Who we are</h1>
@foreach (var paragraph in Site.Story)
{
    <p>@paragraph</p>
}
<h2>What we promise</h2>
<div class="cards">
    @foreach (var value in Site.Values)
    {
        <div class="card"><h3>@value.Title</h3><p>@value.Text</p></div>
    }
</div>
```

Kode 4.11 `Views/Showcase/Company.cshtml`: misi, angka dari data, dan tim.

```
@model ShopFacts
@{
    ViewData["Title"] = "Company";
    int people = Site.Team.Sum(t => t.People);
}
<h1>The company</h1>
<p class="lead">@Site.Mission</p>
<h2>In numbers</h2>
<table class="list">
    <tr><th>Orders from @Model.FirstOrder.ToString("d MMM yyyy")
        to @Model.LastOrder.ToString("d MMM yyyy")</th>
        <td class="num">@Model.Orders.ToString("N0")</td></tr>
    <tr><th>Average order</th>
        <td class="num">@Money.Rp(Model.AverageTicket)</td></tr>
    @foreach (var (outlet, count) in Model.OrdersPerOutlet)
    {
        <tr><th>Orders at @outlet</th>
            <td class="num">@count.ToString("N0")</td></tr>
    }
</table>
<h2>The team: @people people</h2>
<ul>
    @foreach (var t in Site.Team)
    {
        <li>@t.Role: @t.People</li>
    }
</ul>
```

Kode 4.12 `Views/Showcase/Services.cshtml`: daftar harga.

```
@model IReadOnlyList<Service>
@{
    ViewData["Title"] = "Services";
}
<h1>Services and prices</h1>
<table class="list">
    <thead><tr><th>Service</th><th class="num">Price</th></tr></thead>
    <tbody>
    @foreach (var s in Model)
    {
        <tr><td>@s.Name</td><td class="num">@Money.Rp(s.Price) / @s.Unit</td></tr>
    }
    </tbody>
</table>
```

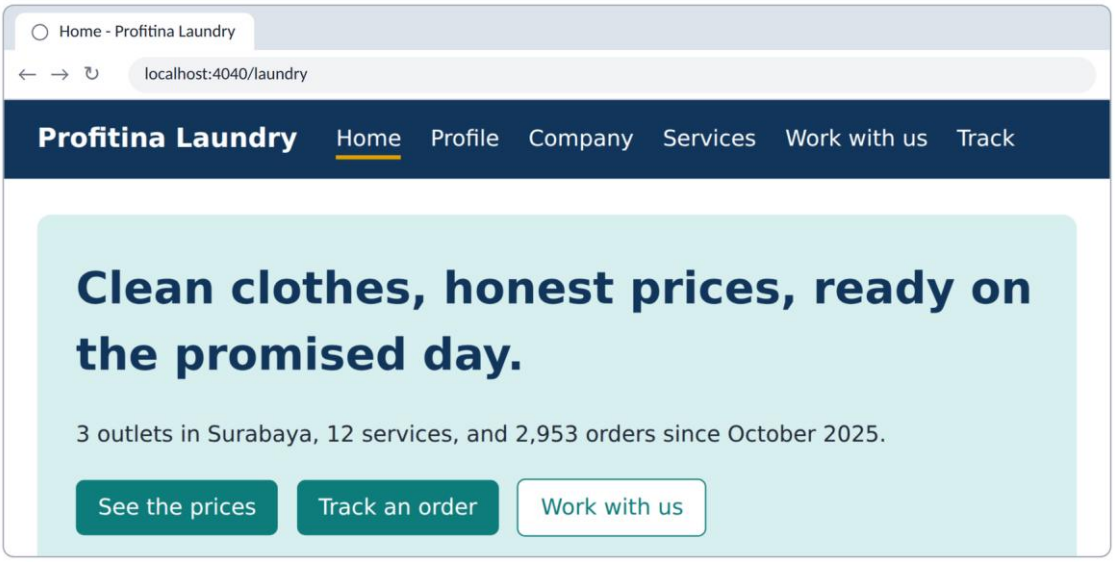
Kode 4.13 `Views/Showcase/Collaboration.cshtml`: apa yang ditawarkan dan diminta usaha ini dari setiap khalayak.

```
@model List<Opening>
@{
    ViewData["Title"] = "Work with us";
}
<h1>Work with us</h1>
<div class="cards">
    @foreach (var o in Model)
    {
        <div class="card">
            <h3>@o.Audience</h3>
            <p><b>We offer:</b> @o.Offer</p>
            <p><b>We ask:</b> @o.Ask</p>
            <p><a href="mailto:@o.Contact">@o.Contact</a></p>
        </div>
    }
</div>
```

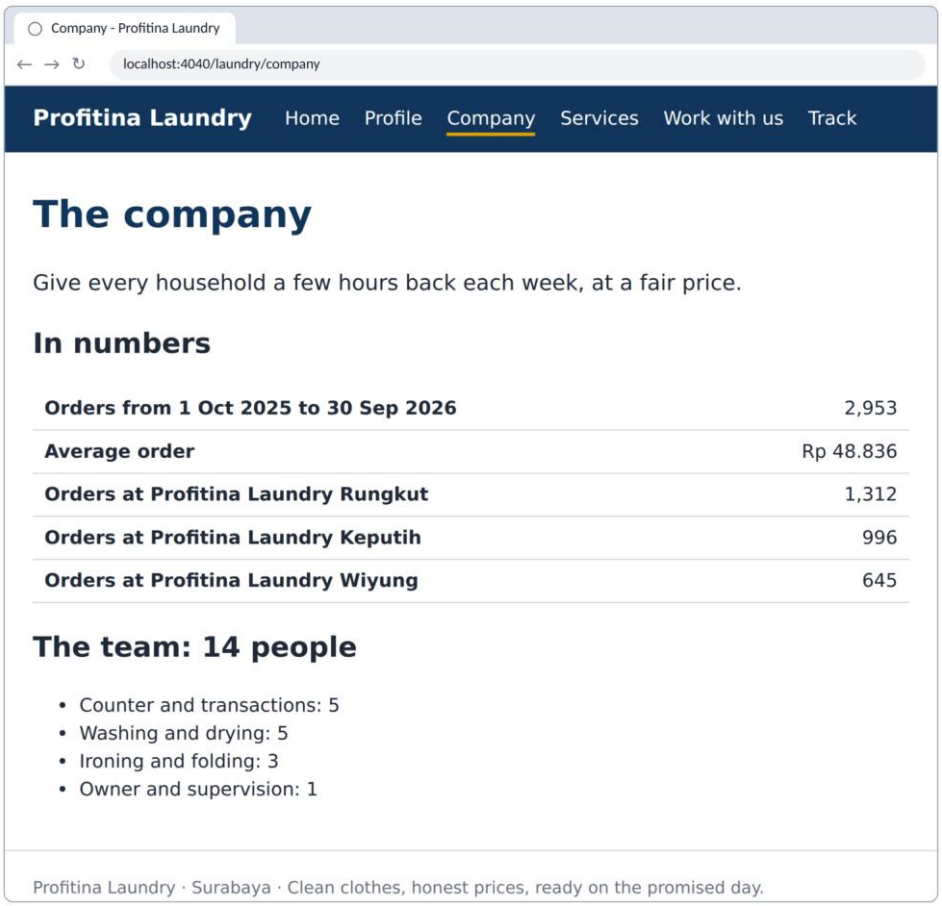
4.3.5 Menjalankannya

```
cd Chapter04/aspnet/LaundryShowcase
dotnet run
```

Gambar 4.3 adalah `/laundry`: slogan dari `site.json` dan tiga angka yang dihitung dari data. **Gambar 4.4** adalah halaman perusahaan. Datanya adalah data contoh basis data Profitina Laundry, 2.953 pesanan dari 1 Oktober 2025 sampai 30 September 2026, dengan rata-rata Rp 48.836 per pesanan; gerai Rungkut, yang tertua, menerima 1.312 di antaranya, Keputih 996, dan Wiyung, yang dibuka Januari 2026, 645. Tak satu pun angka ini muncul di kode maupun di berkas konten.

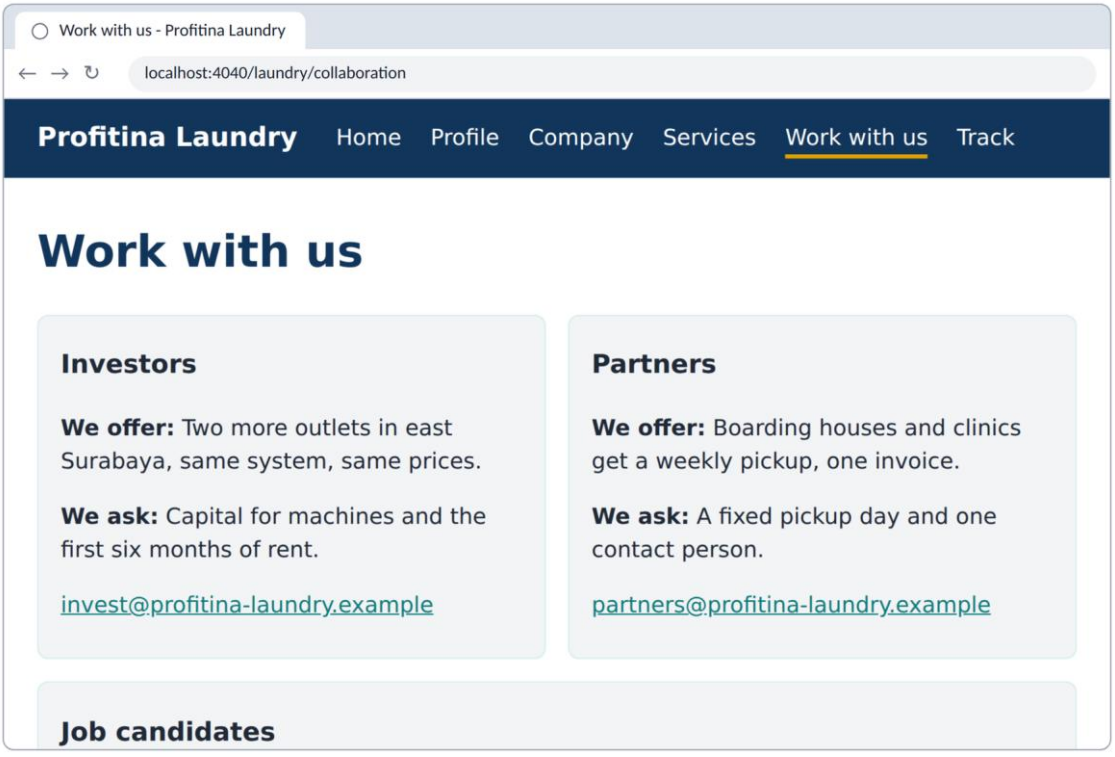


Gambar 4.3 Beranda etalase: slogannya berasal dari `site.json`, hitungannya dari data.



Gambar 4.4 Halaman perusahaan: setiap angka dihitung `LaundryData.Facts()` dari data pesanan.

Gambar 4.5 adalah halaman kerja sama, dengan tiga khalayak dari rumusan soal berdampingan.



Gambar 4.5 Halaman kerja sama: tawaran dan permintaan untuk investor, mitra, dan calon karyawan, dari `site.json`.

Tabel alamatnya diperiksa oleh `verify.sh`. Baris-baris keluarannya yang relevan adalah:

```
302  /
200  /laundry
200  /laundry/profile
200  /laundry/company
200  /laundry/services
200  /laundry/collaboration
200  /laundry/Profile
404  /profile
404  /Showcase/Profile
```

dan pengalihan dari akar membawa header `Location: /laundry`. Setiap baris **Tabel 4.2** terpenuhi.

4.3.6 Penyelesaian keliru yang menggoda

Kesalahan yang menggoda di sini hanya satu karakter. Pengembang yang menulis rute profil dengan garis miring di depan, `[HttpGet("/profile")]`, "karena alamat selalu diawali garis miring", telah menulis rute *mutlak*: rute itu mengabaikan prefiks controller. **Kode 4.14** adalah program terkecil yang memperlihatkannya.

Kode 4.14 `Tempting/AbsoluteRoute/Program.cs`: satu garis miring di depan memindahkan profil keluar dari `/laundry`.

```
// Tempting wrong solution for Problem 4.1: a leading "/" on an action's
// route throws away the controller's "laundry" prefix.
using Microsoft.AspNetCore.Mvc;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllers();
var app = builder.Build();
app.MapControllers();
app.Run();

[Route("laundry")]
public class ShowcaseController : Controller
{
    [HttpGet("")]
    public string Home() => "home";

    [HttpGet("/profile")]              // WRONG: absolute, not "profile"
    public string Profile() => "profile";
}
```

Program ini terkompilasi tanpa peringatan, dan berjalan:

```
200  /laundry  home
404  /laundry/profile
200  /profile  profile
```

Halamannya ada, di alamat yang salah, dan tautan menu (yang dihasilkan dari rute) bahkan menunjuk ke sana, sehingga mengklik ke sana kemari di situs tidak memperlihatkan kesalahan apa pun. Hanya tabel alamat yang menangkapnya. Bab 3 memakai garis miring di depan dengan sengaja, untuk `/api/services/{id}` dan `/error/{code:int}`; di sini hal itu kecelakaan. Aturannya: di dalam controller yang punya `[Route(...)]`, templat sebuah aksi diawali garis miring hanya jika memang dimaksudkan keluar dari prefiks.

4.3.7 Uji diri

Variasi. Pemilik menginginkan etalase di `/profitina-laundry` alih-alih `/laundry`, dengan alamat lama tetap berfungsi untuk sementara. Apa yang berubah?

Jawaban. Satu atribut: `[Route("profitina-laundry")]`. Setiap tautan di view dihasilkan dari rute oleh *tag helper*, sehingga menu mengikuti dengan sendirinya. Untuk mempertahankan alamat lama, tambahkan atribut kedua, `[Route("laundry")]`, pada kelas itu; sebuah controller boleh punya beberapa prefiks rute, dan setiap aksi lalu punya dua alamat. Pengalihan di `Program.cs` sebaiknya menunjuk ke yang baru.

4.4 Soal 4.2: Di Mana Cucian Saya?

4.4.1 Soalnya

> **SOAL 4.2.** Setiap nota pesanan Profitina Laundry memuat sebuah kode: huruf B diikuti empat angka, misalnya B0072. Tambahkan ke etalase sebuah halaman tempat pelanggan mengetik kode itu dan melihat status pesanannya (diterima, siap beserta nomor raknya, atau sudah diambil), layanannya, gerainya, tanggal-tanggalnya, dan jumlah tagihannya. Formulirnya di `/laundry/track`; setiap pesanan punya alamatnya sendiri, `/laundry/track/B0072`, agar jawabannya dapat ditandai (*bookmark*). Kode boleh diketik dengan huruf kecil. Alamat yang tidak mungkin berupa kode pesanan sama sekali tidak boleh mencapai kode pelacakan; kode yang mungkin ada tetapi ternyata tidak ada harus mendapat halaman "tidak ditemukan" yang sopan dengan status 404.

4.4.2 Memahaminya secara manual

Ada tiga jenis pesanan di data, satu untuk setiap status. **Tabel 4.3** dikerjakan secara manual dari tiga baris `Bill.csv`, dan itulah persis yang harus ditampilkan halaman.

Tabel 4.3 Tiga pesanan dari `Bill.csv`, satu untuk setiap status, dikerjakan secara manual. Jumlah untuk B0072 cocok: $7 \times 8000 = 56000$.

Kode	Status	Layanan, satuan	Gerai	Diterima	Selesai	Diambil	Jumlah
B0001	departed	S0008 Bed cover, 1	O01 Rungkut	1 Okt 2025, 09.46	4 Okt 2025, 07.16	5 Okt 2025, 07.24	35.000
B0072	finished (rak A01)	S0001 Wash, dry & fold, 7 kg	O01 Rungkut	14 Okt 2025, 12.11	16 Okt 2025, 07.45	belum	56.000
B2953	queued	S0006 Dress/kebaya, 1	O01 Rungkut	30 Sep 2026, 20.49	belum	belum	45.000

Tabel alamat soal ini adalah **Tabel 4.4**. Baris yang menarik adalah yang gagal: `X12` dan `B00721` tidak mungkin menjadi kode pesanan, sedangkan `B9999` mungkin saja, tetapi tidak ada.

Tabel 4.4 Tabel alamat Soal 4.2.

Alamat	Yang diharapkan
`/laundry/track`	200, formulirnya
`/laundry/track?code=b0072`	302 ke `/laundry/track/b0072`
`/laundry/track/B0072`, `/laundry/track/b0001`, `/laundry/track/B2953`	200, pesanannya
`/laundry/track/B9999`	404, halaman ramah: kode yang mungkin, tetapi pesanannya tidak ada
`/laundry/track/X12`, `/laundry/track/B00721`	404: bukan kode yang mungkin, tidak ada rute yang cocok

4.4.3 Merancangnya di atas kertas

"Tidak mungkin berupa kode pesanan" adalah batasan rute. Aturan "B diikuti tepat empat angka" adalah ekspresi reguler, `^B\d{4}$`: `^` dan `$` mengikatnya ke seluruh segmen, `\d` adalah satu angka, `{4}` berarti tepat empat. Sebagai batasan rute, `{code:regex(...)}`, aturan ini membuat alamat seperti `X12` tidak cocok dengan rute mana pun, sehingga aksi pelacakan hanya berjalan untuk kode yang mungkin. ASP.NET Core mengevaluasi batasan ekspresi reguler *tanpa membedakan huruf besar-kecil* [3], sehingga kode berhuruf kecil sudah tertangani tanpa pola kedua.

Satu rincian sintaksis menentukan segalanya. Di dalam templat rute, kurung kurawal berarti "parameter". Karena itu kurung kurawal milik ekspresi reguler di `{4}` harus digandakan, `{{4}}`, begitu pula kurung siku, `[[` dan `]]`, karena di *attribute route* kurung siku menandai token seperti `[controller]` [3]. Templatnya menjadi `track/{code:regex(^B\d{{4}}$)}`, ditulis sebagai *verbatim string* agar garis miring terbalik sampai ke ekspresi reguler tanpa berubah.

Formulirnya `GET` yang mengalihkan. Formulir di `/laundry/track` mengirim `?code=…` ke alamat yang sama. Aksinya tidak menampilkan pesanan itu sendiri; ia menjawab dengan **pengalihan** ke `/laundry/track/{code}`, sehingga alamat di peramban menjadi alamat milik pesanan itu, seperti diminta soal. `RedirectToAction(nameof(Track), new { code })` meminta alamat itu kepada sistem perutean, persis seperti yang dilakukan *tag helper*.

Privasi. Halaman pelacakan bersifat publik: siapa pun yang mengetik sebuah kode melihat jawabannya. Kode itu pendek dan mudah ditebak (hanya ada 10.000 kemungkinan), sehingga halaman tidak boleh menampilkan apa pun yang mengenali pelanggan: tidak ada nama, nomor telepon, atau alamat. Itulah sebabnya `Order` di **Kode 4.3** sama sekali tidak punya isian pelanggan; yang tidak dimuat tidak dapat bocor. Undang-Undang Pelindungan Data Pribadi Indonesia, UU No. 27 Tahun 2022, menuntut kehati-hatian semacam ini terhadap data pribadi [4]. Nomor rak ditampilkan, karena membantu pelanggan di konter dan tidak memberi tahu apa pun kepada orang asing.

4.4.4 Penyelesaiannya

Kedua aksinya ada di **Kode 4.6**. `TrackForm` menampilkan formulir jika tidak ada kode yang diberikan, dan jika ada, merapikan kodenya lalu mengalihkan. `Track` mencari pesanannya dan mengembalikan `NotFound()` jika tidak ada, yang diubah *middleware* kode status menjadi halaman ramah berstatus 404; jika ada, ia menaruh nama dan satuan layanan serta nama gerai ke `ViewData` lalu menampilkan pesanannya. Aksi `Error` sama dengan milik Bab 3, di bawah prefiks:

pipeline menjalankan ulang `/laundry/error/404` [5]. **Kode 4.15** adalah formulirnya dan **Kode 4.16** halaman pesanannya.

Kode 4.15 `Views/Showcase/TrackForm.cshtml`: formulir `GET` dengan satu isian.

```
@{
    ViewData["Title"] = "Track an order";
}
<h1>Where is my laundry?</h1>
<form asp-action="TrackForm" method="get" class="track">
    <label for="code">Order code on your ticket, for example B0072</label>
    <input id="code" name="code" maxlength="5" autofocus />
    <button type="submit">Track</button>
</form>
```

Kode 4.16 `Views/Showcase/Track.cshtml`: status dalam kata-kata, lalu fakta pesanannya, tanpa data pribadi.

```
@model Order
@{
    ViewData["Title"] = $"Order {Model.Id}";
    string When(DateTime? t) => t?.ToString("ddd d MMM yyyy, HH:mm") ?? "-";
}
<h1>Order @Model.Id</h1>
<p class="status @Model.Status">
    @switch (Model.Status)
    {
        case "queued": <text>Received and waiting to be washed.</text> break;
        case "finished":
            <text>Ready: collect it from rack @Model.Rack.</text> break;
        default: <text>Collected. Thank you!</text> break;
    }
</p>
<table class="list">
    <tr><th>Service</th>
        <td>@ViewData["Service"], @Model.Units @ViewData["Unit"]</td></tr>
    <tr><th>Outlet</th><td>@ViewData["Outlet"]</td></tr>
    <tr><th>Received</th><td>@When(Model.Arrived)</td></tr>
    <tr><th>Finished</th><td>@When(Model.Finished)</td></tr>
    <tr><th>Collected</th><td>@When(Model.Departed)</td></tr>
    <tr><th>Amount</th><td>@Money.Rp(Model.Payment)</td></tr>
</table>
<p><a asp-action="TrackForm">Track another order</a></p>
```

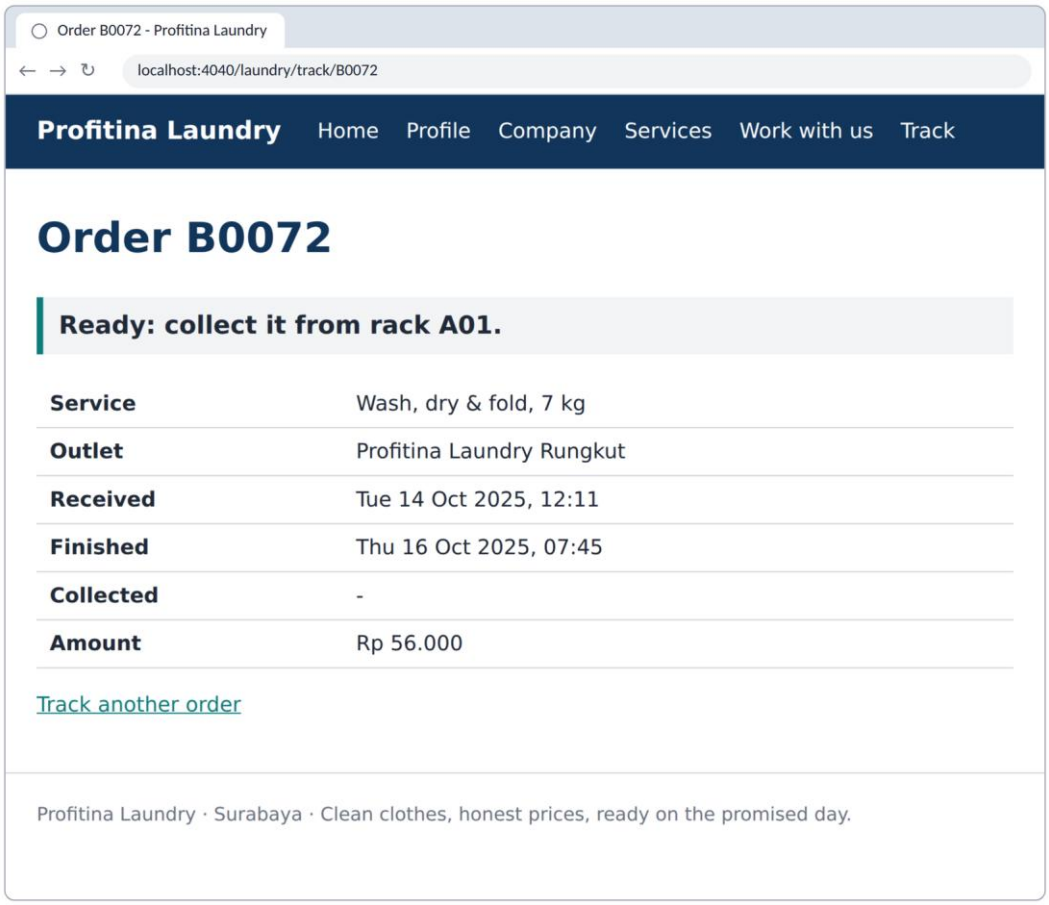
View ini memperlihatkan dua fitur Razor yang belum pernah terlihat. `@switch` memilih kalimat untuk status, dan `<text>…</text>` menandai teks biasa di dalam kode C#, tempat Razor semestinya mengharapkan sebuah tag. Fungsi lokal `When` memformat tanggal yang mungkin kosong, dengan mencetak "-" untuk `null`. Kelas paragraf status, `status @Model.Status`, membuat lembar gaya dapat mewarnai pesanan yang masih antre dengan kuning. **Kode 4.17** adalah view galatnya.

Kode 4.17 `Views/Shared/Error.cshtml`: satu kalimat untuk halaman atau pesanan yang tidak ada, beserta alamat yang diminta.

```
@model int
@{
    ViewData["Title"] = Model == 404 ? "Not found" : "Something went wrong";
}
<h1>@ViewData["Title"]</h1>
<p>@(Model == 404 ? "There is no page or order at" : "We could not answer")
   <code>@ViewData["Path"]</code>.</p>
<p><a asp-controller="Showcase" asp-action="Home">Back to the home page</a></p>
```

4.4.5 Menjalankannya

Gambar 4.6 adalah halaman untuk B0072, dan halaman itu sepakat dengan **Tabel 4.3** baris demi baris.



Gambar 4.6 Halaman pelacakan pesanan B0072: siap di rak A01, dengan layanan, gerai, tanggal, dan jumlahnya, tanpa apa pun tentang pelanggan.

`verify.sh` menanyakan alamat-alamat pada **Tabel 4.4**:

```
200  /laundry/track
302  /laundry/track?code=b0072
200  /laundry/track/B0072
200  /laundry/track/b0001
200  /laundry/track/B2953
404  /laundry/track/B9999
404  /laundry/track/X12
404  /laundry/track/B00721
```

Pengalihannya `Location: /laundry/track/b0072`, dengan kode seperti yang diketik: batasan rute menerimanya, dan pencarian di kamus tidak membedakan huruf besar-kecil. Teks ketiga halaman pesanan, dengan tag dibuang, adalah:

```
== text of /laundry/track/B0072
Order B0072
Ready: collect it from rack A01.
Service
Wash, dry &amp;amp; fold, 7 kg
OutletProfitina Laundry Rungkut
ReceivedTue 14 Oct 2025, 12:11
FinishedThu 16 Oct 2025, 07:45
Collected-
AmountRp 56.000
Track another order
== text of /laundry/track/B2953
Order B2953
Received and waiting to be washed.
Service
Dry cleaning - dress/kebaya, 1 item
OutletProfitina Laundry Rungkut
ReceivedWed 30 Sep 2026, 20:49
Finished-
Collected-
AmountRp 45.000
Track another order
== text of /laundry/track/B0001
Order B0001
Collected. Thank you!
Service
Bed cover, any size, 1 item
OutletProfitina Laundry Rungkut
ReceivedWed 1 Oct 2025, 09:46
FinishedSat 4 Oct 2025, 07:16
CollectedSun 5 Oct 2025, 07:24
AmountRp 35.000
Track another order
== text of /laundry/track/B9999
Not found
There is no page or order at
/laundry/track/B9999.
Back to the home page
```

(Sel-sel tabel menyambung, “OutletProfitina…”, hanya karena skrip membuang tag di antaranya.) Ketiga status dan pesanan yang tidak ada sesuai dengan rancangannya.

4.4.6 Penyelesaian keliru yang menggoda

Kesalahan yang menggoda adalah menulis ekspresi reguler seperti yang biasa dipelajari, dengan kurung kurawal tunggal: `{code:regex(^B\d{4}$)}`. **Kode 4.18** melakukan persis itu.

Kode 4.18 `Tempting/RegexBraces/Program.cs`: ekspresi reguler dengan kurung kurawal tunggal di dalam templat rute.

```
// Tempting wrong solution for Problem 4.2: single braces inside a regex
// constraint. In a route template { and } mean "parameter".
using Microsoft.AspNetCore.Mvc;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllers();
var app = builder.Build();
app.MapControllers();
app.Run();

[Route("laundry")]
public class ShowcaseController : Controller
{
    [HttpGet(@"track/{code:regex(^B\d{4}$)}")]     // WRONG: needs {{4}}
    public string Track(string code) => $"order {code}";
}
```

Penganalisis rute milik kompilator sudah memperingatkan, dua kali, saat *build*:

```
Program.cs(14,28): warning ASP0017: Route issue: In a route parameter, '{' and '}' must be
escaped with '{{' and '}}'. (https://aka.ms/aspnet/analyzers)
Program.cs(14,41): warning ASP0017: Route issue: There is an incomplete parameter in the
route template. Check that each '{' character has a matching '}' character.
(https://aka.ms/aspnet/analyzers)
```

Peringatan mudah diabaikan, jadi programnya tetap dijalankan. Program itu tidak selamat dari permintaan pertamanya: ketika perutean membangun tabel rute, pengurai templat menjumpai `{4}` di dalam parameter, dan aplikasi berhenti dengan *exception*. `curl` sama sekali tidak mendapat jawaban (statusnya tercetak `000`), dan lognya berbunyi:

```
      Application startup exception
      Microsoft.AspNetCore.Routing.Patterns.RoutePatternException: In a route parameter,
'{' and '}' must be escaped with '{{' and '}}'.
         at Microsoft.AspNetCore.Routing.Patterns.RoutePatternParser.Parse(String pattern)
```

Ini kesalahan yang gagal dengan lantang, jenis kesalahan yang terbaik. Pelajarannya punya dua sisi: bacalah peringatan *build* (penganalisisnya sudah tahu), dan ingatlah bahwa templat rute punya sintaksisnya sendiri, tempat `{`, `}`, `[`, dan `]` bermakna khusus.

4.4.7 Uji diri

Variasi. Tahun depan toko berencana beralih ke kode lima angka (B00001), sementara nota lama berkode empat angka masih beredar. Tuliskan batasannya.

Jawaban. `track/{code:regex(^B\d{{4,5}}$)}`: `{4,5}` berarti empat atau lima angka, dan kedua kurung kurawalnya digandakan. Tidak ada yang lain yang berubah, karena pencarian di kamus tidak peduli panjangnya. Alternatif yang menggoda, `{code:length(5,6)}`, juga akan menerima `XXXXX`.

4.5 Soal 4.3: Tabel Rute secara Manual

4.5.1 Soalnya

> **SOAL 4.3.** Tanpa menjalankan program, gunakan **Kode 4.5** dan **Kode 4.6** untuk meramalkan, bagi masing-masing dari dua belas permintaan di bawah ini, *endpoint* mana yang menanganinya dan kode status apa yang kembali: `GET /`, `/laundry`, `/laundry/Profile`, `/profile`, `/Showcase/Profile`, `/laundry/track`, `/laundry/track?code=b0072`, `/laundry/track/b0001`, `/laundry/track/B9999`, `/laundry/track/X12`,

> `/laundry/track/B00721`, `/css/site.css`. Lalu buat aplikasi menampilkan tabel rutenya sendiri, dan periksa setiap ramalan terhadap program yang berjalan.

4.5.2 Memahaminya secara manual

Soal ini adalah router dari Bab 2, yang dijalankan di kepala. Untuk setiap path, daftarlah templat yang mungkin cocok, terapkan batasannya, dan jika beberapa cocok, terapkan prioritas. *Endpoint*-nya berasal dari tiga tempat di `Program.cs`: `MapStaticAssets()` (berkas-berkas di `wwwroot`), `MapGet("/", …)` (pengalihan), dan `MapControllers()` (setiap *attribute route* milik `ShowcaseController`). Tidak ada rute konvensional.

4.5.3 Merancangnya di atas kertas

Tabel 4.5 adalah ramalannya, satu baris per permintaan, beserta alasannya.

Tabel 4.5 Tabel rute yang diramalkan secara manual untuk dua belas permintaan. Dua alasan berbeda bersembunyi di balik 404: tidak ada rute, atau aksi yang mengembalikan `NotFound()`.

Permintaan	Templat yang cocok	*Endpoint*	Status ramalan
`/`	`/`	pengalihan	302
`/laundry`	`laundry`	`Home`	200
`/laundry/Profile`	`laundry/profile` (huruf besar-kecil diabaikan)	`Profile`	200
`/profile`	tidak ada (tanpa prefiks)	tidak ada	404
`/Showcase/Profile`	tidak ada (tidak ada rute konvensional)	tidak ada	404
`/laundry/track`	`laundry/track`	`TrackForm`	200
`/laundry/track?code=b0072`	`laundry/track` (*query* diabaikan perutean)	`TrackForm`	302
`/laundry/track/b0001`	`laundry/track/{code:regex(...)}`, huruf besar-kecil diabaikan	`Track`	200
`/laundry/track/B9999`	sama; pesanannya tidak ada	`Track`	404
`/laundry/track/X12`	tidak ada: batasannya gagal	tidak ada	404
`/laundry/track/B00721`	tidak ada: lima angka	tidak ada	404
`/css/site.css`	aset statis	berkasnya	200

Dua baris patut dilihat dua kali. *Query string* tidak berperan dalam perutean: `/laundry/track?code=b0072` cocok dengan templat yang sama dengan `/laundry/track`, dan aksinyalah yang memutuskan apa yang dilakukan dengan `code`. Dan `B9999` serta `X12` sama-sama berakhir 404, tetapi dengan alasan berbeda: `B9999` *mencapai* aksinya, yang mencari kode itu dan gagal; `X12` tidak pernah mencapai aksi mana pun. Pengunjung melihat halaman ramah yang sama; log dan banyaknya pekerjaan yang berbeda.

4.5.4 Penyelesaiannya: biarkan aplikasi menampilkan rutenya

Perutean menyimpan semua *endpoint* dalam layanan bernama `EndpointDataSource`. Bagian terakhir **Kode 4.5** memetakan alamat khusus pengembangan, `/_routes`, yang *handler*-nya menerima layanan itu lewat injeksi (parameter *handler minimal API* di-*resolve* dari container), menyaring *endpoint* rute, mengurutkannya berdasarkan templat, dan mencetak setiap templat di samping nama *endpoint*-nya. Alamat ini hanya didaftarkan jika

`app.Environment.IsDevelopment()`, karena tabel rute adalah rincian internal yang tidak semestinya diungkap situs publik.

4.5.5 Menjalankannya

```
curl http://localhost:4040/_routes
{**path:file}                          Fallback {**path:file}
/                                      HTTP: GET /
/_routes                               HTTP: GET /_routes
laundry                                ShowcaseController.Home
laundry/collaboration                  ShowcaseController.Collaboration
laundry/company                        ShowcaseController.Company
laundry/error/{status:int}             ShowcaseController.Error
laundry/profile                        ShowcaseController.Profile
laundry/services                       ShowcaseController.Services
laundry/track                          ShowcaseController.TrackForm
laundry/track/{code:regex(^B\d{{4}}$)} ShowcaseController.Track
```

Inilah tabel rute sesungguhnya, dalam kata-kata router itu sendiri. Rute untuk masing-masing berkas di `wwwroot/css` disaring oleh *handler*; baris pertama adalah *fallback* penampung semua yang ditambahkan `MapStaticAssets` di Development, agar berkas yang belum tercantum dalam manifes waktu-*build*-nya tetap dapat dilayani. Setiap templat pada **Tabel 4.5** ada di sana, dengan batasan persis seperti ditulis. Kode status dari `verify.sh` (Subbab 4.3.5 dan 4.4.5, serta `200 /css/site.css`) sepakat dengan kedua belas ramalan.

4.5.6 Penyelesaian keliru yang menggoda

Misalkan permintaan berikutnya meminta halaman per gerai di `/laundry/track/{outlet}`, dan seorang pengembang menambahkannya di samping rute pesanan tanpa memikirkan tabelnya. **Kode 4.19** memperlihatkan versi terkecilnya: dua templat dengan bentuk yang persis sama.

Kode 4.19 `Tempting/Ambiguous/Program.cs`: dua *attribute route* yang cocok dengan alamat yang persis sama.

```
// Tempting wrong solution for Problem 4.3: two actions whose routes
// match exactly the same addresses.
using Microsoft.AspNetCore.Mvc;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllers();
var app = builder.Build();
app.MapControllers();
app.Run();

[Route("laundry")]
public class ShowcaseController : Controller
{
    [HttpGet("track/{code}")]
    public string Track(string code) => $"order {code}";

    [HttpGet("track/{outlet}")]          // WRONG: same shape as the one above
    public string Outlet(string outlet) => $"outlet {outlet}";
}
```

Lagi-lagi penganalisis memperingatkan saat *build* (ASP0023, “Route ‘track/{code}’ conflicts with another action route”) [6]. Saat berjalan, programnya menyala, dan permintaan pertama gagal dengan status 500:

```
500  /laundry/track/B0072
fail: Microsoft.AspNetCore.Server.Kestrel[13]
      Connection id "…", Request id "…": An unhandled exception was thrown by the
application.
      Microsoft.AspNetCore.Routing.Matching.AmbiguousMatchException: The request matched
multiple endpoints. Matches:

      ShowcaseController.Track (Ambiguous)
      ShowcaseController.Outlet (Ambiguous)
```

Prioritas tidak dapat menolong: kedua templat punya literal `track` dan satu parameter tanpa batasan, sehingga tidak ada yang lebih khusus. Router menolak menebak, dan itu keputusan yang benar; perbaikannya adalah membuat alamat-alamatnya dapat dibedakan: batasan pada salah satunya (`{code:regex(^B\d{{4}}$)}` dan `{outlet:regex(^O\d{{2}}$)}`), atau literal yang berbeda (`/laundry/outlets/{id}`). Tabel alamat pada langkah 2 akan memperlihatkan bentroknya sebelum satu baris kode pun ditulis.

4.5.7 Uji diri

Variasi. Setelah perbaikan dengan dua ekspresi reguler, *endpoint* mana yang menangani `/laundry/track/O01`, `/laundry/track/B0072`, dan `/laundry/track/Z99`?

Jawaban. `O01` hanya cocok dengan templat gerai, `B0072` hanya dengan templat pesanan, dan `Z99` tidak dengan keduanya, sehingga mendapat 404 tanpa mencapai aksi mana pun. Batasan yang saling meniadakan menghapus kerancuan sepenuhnya.

4.6 Soal 4.4: Header Waktu Respons

4.6.1 Soalnya

> **SOAL 4.4.** Untuk memantau kecepatan situs dari luar, Dimas ingin setiap respons membawa header `X-Response-Time` berisi jumlah milidetik yang dihabiskan server untuknya, misalnya `X-Response-Time: 12 ms`. Tulislah sekali, untuk setiap halaman, berkas, dan galat.

4.6.2 Memahaminya secara manual

"Untuk setiap respons" berarti *middleware*, yang diletakkan paling depan agar membungkus semuanya. Durasinya baru diketahui setelah sisa *pipeline* selesai bekerja, sehingga rencana yang wajar adalah *middleware* pencatatan dari Bab 3: nyalakan *stopwatch*, `await next(context)`, lalu isi headernya. Rencana itu keliru, dan memahami mengapa adalah inti soal ini.

Respons HTTP dikirim sebagai *baris status*, lalu *header*, lalu *badan* [7]. Begitu byte pertama badan meninggalkan server, status dan header sudah terkirim dan tidak dapat berubah lagi. View Razor menulis HTML-nya ke badan respons selama merender, sehingga ketika `await next(context)` kembali, respons biasanya sudah *dimulai*: `context.Response.HasStarted` bernilai benar, dan header menjadi hanya-baca [8].

4.6.3 Merancangnya di atas kertas

Kita perlu menjalankan kode pada saat terakhir *sebelum* header dikirim, kapan pun itu. ASP.NET Core menyediakan kait (*hook*) tepat untuk itu: `context.Response.OnStarting(callback)` mendaftarkan fungsi yang dipanggil server tepat sebelum mengirim header [8]. *Middleware* menyalakan *stopwatch*, mendaftarkan *callback* yang membaca *stopwatch* dan mengisi header, lalu

memanggil sisa *pipeline*. Waktu yang terukur adalah dari awal *middleware* sampai saat header dikirim, yaitu waktu sampai byte pertama, angka yang diinginkan Dimas.

Menulisnya sebagai kelas, alih-alih lambda di `Program.cs`, menjaga `Program.cs` tetap mudah dibaca dan membuat *middleware* itu dapat dipakai ulang. Kelas *middleware* ASP.NET Core memerlukan konstruktor yang menerima `RequestDelegate` untuk komponen berikutnya, dan metode `InvokeAsync(HttpContext)`; `app.UseMiddleware<T>()` menambahkannya ke *pipeline*.

4.6.4 Penyelesaiannya

Kode 4.20 `Middleware/ResponseTimeMiddleware.cs`: header diisi dalam *callback* `OnStarting`, tepat sebelum header dikirim.

```
using System.Diagnostics;

namespace LaundryShowcase.Middleware;

// Adds "X-Response-Time: 12 ms" to every response.
public class ResponseTimeMiddleware(RequestDelegate next)
{
    public async Task InvokeAsync(HttpContext context)
    {
        var clock = Stopwatch.StartNew();
        // Headers must be set before the first byte of the body is sent,
        // so we ask to be called back at exactly that moment.
        context.Response.OnStarting(() =>
        {
            context.Response.Headers["X-Response-Time"] =
                $"{clock.ElapsedMilliseconds} ms";
            return Task.CompletedTask;
        });
        await next(context);
    }
}
```

Di **Kode 4.5** ia menjadi baris pertama *pipeline*, `app.UseMiddleware<ResponseTimeMiddleware>()`, sehingga setiap respons, termasuk berkas, pengalihan, dan halaman galat, melewatinya.

4.6.5 Menjalankannya

```
curl -s -D - -o /dev/null http://localhost:4040/laundry
```

`-D -` mencetak header respons, `-o /dev/null` membuang badannya (di Windows PowerShell tulis `-o NUL`). Jawaban sesungguhnya:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Server: Kestrel
Transfer-Encoding: chunked
X-Response-Time: 2 ms
```

Angkanya berubah dari permintaan ke permintaan, dan permintaan pertama setelah awal jalan lebih lambat, karena view dimuat pada pemakaian pertama.

4.6.6 Penyelesaian keliru yang menggoda

Kode 4.21 adalah rencana wajar dari Subbab 4.6.2, dalam program minimal yang satu-satunya *endpoint*-nya mengembalikan teks pendek.

Kode 4.21 `Tempting/LateHeader/Program.cs`: header diisi setelah sisa *pipeline* sudah memulai responsnya.

```
// Tempting wrong solution for Problem 4.4: setting a header after the
// rest of the pipeline has already started sending the response.
using System.Diagnostics;

var app = WebApplication.CreateBuilder(args).Build();

app.Use(async (context, next) =>
{
    var clock = Stopwatch.StartNew();
    await next(context);
    context.Response.Headers["X-Response-Time"] =         // WRONG: too late
        $"{clock.ElapsedMilliseconds} ms";
});
app.MapGet("/laundry", () => "Clean clothes, honest prices.");
app.Run();
```

Tidak ada peringatan *build*; kesalahan ini tidak terlihat sampai program dijalankan. Dari sisi klien permintaannya berhasil (status 200 dan teksnya tiba, tanpa header itu), dan server mencatat sebuah *exception* untuk setiap permintaan:

```
200  /laundry  Clean clothes, honest prices.
fail: Microsoft.AspNetCore.Server.Kestrel[13]
      Connection id "…", Request id "…": An unhandled exception was thrown by the
application.
      System.InvalidOperationException: Headers are read-only, response has already
started.
         at Program.<>c.<<Main>$>b__0_0>d.MoveNext() in
…/Tempting/LateHeader/Program.cs:line 11
```

Inilah jenis bug terburuk: situs tampak berjalan, headernya diam-diam hilang, dan log penuh galat yang tidak dibaca siapa pun. Aturan yang sama berlaku untuk segala yang masuk ke header, seperti *cookie*, pengalihan, dan kode status: putuskan semuanya sebelum badan dimulai, atau daftarkan *callback* `OnStarting`.

4.6.7 Uji diri

Variasi. Apakah versi lambda pada **Kode 4.21** akan berjalan jika *endpoint*-nya mengembalikan `Results.NoContent()` (status 204, tanpa badan)?

Jawaban. Ya, dan justru di situlah jebakannya. Percobaan dengan `app.MapGet("/laundry", () => Results.NoContent())` menjawab `204 No Content` dengan `X-Response-Time: 9 ms` dan tidak mencatat galat: tanpa badan, tidak ada yang memaksa header terkirim sebelum *middleware* kembali, sehingga header masih dapat ditulis. Itulah sebabnya kesalahan ini lolos pengujian pada *endpoint* sederhana dan gagal pada halaman sungguhan. Kode yang benar hanya ketika badannya kebetulan kosong bukanlah kode yang benar; `OnStarting` yang benar.

4.7 Soal 4.5: Mode Pemeliharaan yang Tetap Bergaya

4.7.1 Soalnya

> **SOAL 4.5.** Selama situs sedang diperbarui, pemilik ingin mengalihkannya ke mode pemeliharaan tanpa menyunting kode: setiap halaman harus menjawab dengan status 503 (“Service Unavailable”), header `Retry-After` sebesar satu jam, dan halaman singkat yang menyatakan bahwa gerai-gerai tetap buka seperti biasa. Halaman pemeliharaan itu harus bergaya dengan lembar gaya milik situs sendiri, jadi lembar gayanya harus tetap berfungsi sementara semua yang lain ditutup.

4.7.2 Memahaminya secara manual

Ada tiga tuntutan tersembunyi dalam rumusan itu. *Tanpa menyunting kode* berarti konfigurasi: sebuah pengaturan yang dapat diubah di `appsettings.json` atau, lebih baik, dari luar dengan variabel lingkungan. *Setiap halaman* berarti *middleware* lagi. Dan *lembar gayanya harus tetap berfungsi* berarti *middleware* harus dapat membedakan permintaan untuk `/css/site.css` dari permintaan untuk sebuah halaman. Status 503 dengan `Retry-After` adalah cara baku HTTP untuk menyatakan "sementara tidak tersedia, kembalilah nanti", yang dipahami mesin pencari dan tidak dianggap sebagai situs yang menghilang [7].

4.7.3 Merancangnya di atas kertas

Sakelarnya. `appsettings.json` mendapat bagian `"Showcase": { "Maintenance": false }`. *Middleware* membaca `app.Configuration.GetValue<bool>("Showcase:Maintenance")` pada setiap permintaan, sehingga nilai pengaturan saat itu selalu berlaku. ASP.NET Core juga membaca variabel lingkungan ke dalam konfigurasi yang sama, dengan garis bawah ganda sebagai pengganti titik dua: `Showcase__Maintenance=true` menimpa berkasnya tanpa menyentuhnya [9].

Membedakan berkas dari halaman. *Middleware* dapat membandingkan path dengan `/css/`, tetapi jawaban yang lebih bersih memakai apa yang sudah diketahui perutean. Setelah `UseRouting()`, *endpoint* yang terpilih tersedia sebagai `context.GetEndpoint()`, dan untuk aset statis pola rutenya diawali `css/`. Karena itu *middleware* pemeliharaan diletakkan **sesudah** `UseRouting()`, menanyakan *endpoint*-nya, dan meloloskan aset statis. Jika diletakkan sebelum perutean, ia sama sekali tidak akan menemukan *endpoint*.

4.7.4 Penyelesaiannya

Middleware pemeliharaan adalah `app.Use(...)` langsung sesudah `app.UseRouting()` di **Kode 4.5**. Jika sakelarnya menyala dan *endpoint*-nya bukan aset CSS, ia mengisi status dan header `Retry-After`, menulis halaman HTML kecil yang berdiri sendiri dan menautkan `/css/site.css`, lalu kembali *tanpa memanggil* `next`, sehingga permintaan berakhir di situ: inilah **short-circuit** (hubungan pendek). Halaman itu ditulis sebelum badan apa pun dimulai, sehingga mengisi status dan header aman dilakukan, tidak seperti pada Soal 4.4. **Kode 4.22** adalah berkas konfigurasinya.

Kode 4.22 `appsettings.json` milik LaundryShowcase: sakelar pemeliharaan, mati secara bawaan.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "Showcase": {
    "Maintenance": false
  }
}
```

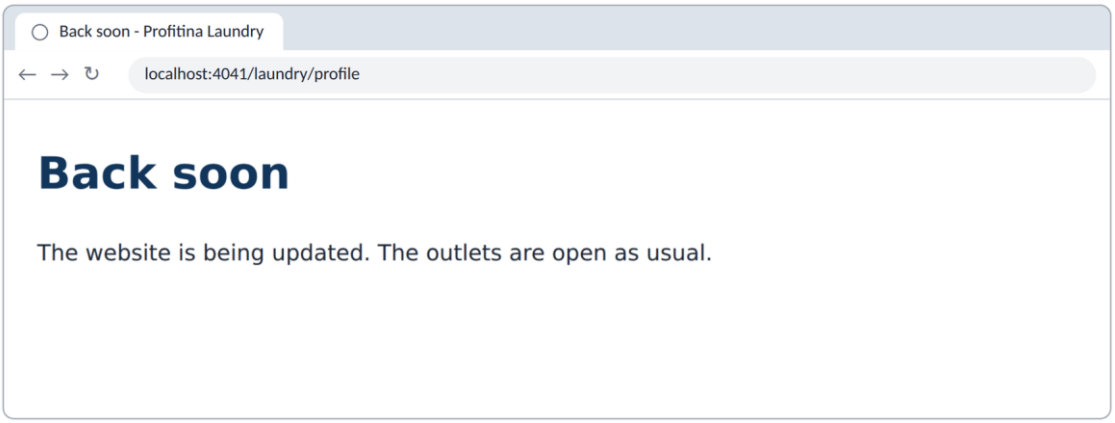
4.7.5 Menjalankannya

Sakelarnya dinyalakan dari terminal, tanpa menyunting berkas apa pun:

```
Showcase__Maintenance=true dotnet run            # macOS, Linux
$env:Showcase__Maintenance="true"; dotnet run    # Windows PowerShell
```

```
503  /laundry
503  /laundry/profile
200  /css/site.css
HTTP/1.1 503 Service Unavailable
Retry-After: 3600
<body><main><h1>Back soon</h1>
```

Halaman-halaman menjawab 503 dengan headernya, dan lembar gaya tetap menjawab 200, sehingga halaman pemeliharaan bergaya, seperti diperlihatkan **Gambar 4.7**.



Gambar 4.7 Halaman pemeliharaan, dijawab dengan status 503 dan diberi gaya oleh lembar gaya situs sendiri, yang diloloskan mode pemeliharaan.

4.7.6 Penyelesaian keliru yang menggoda

Kesalahan yang menggoda adalah menaruh pemeriksaan pemeliharaan di tempat "pemeriksaan" biasanya ditaruh, paling depan di *pipeline*. **Kode 4.23** melakukannya, dengan semua yang lain tidak berubah.

Kode 4.23 `Tempting/MaintenanceFirst/Program.cs`: pemeriksaan pemeliharaan sebelum `UseRouting`, ketika belum ada *endpoint* yang dipilih.

```
// Tempting wrong solution for Problem 4.5: the maintenance check runs
// before UseRouting, so it cannot know that /css/site.css is a file.
var app = WebApplication.CreateBuilder(args).Build();

app.Use(async (context, next) =>                // WRONG: before UseRouting
{
    var endpoint = context.GetEndpoint() as RouteEndpoint;   // always null
    bool isAsset = endpoint?.RoutePattern.RawText?.StartsWith("css/") == true;
    if (!isAsset)
    {
        context.Response.StatusCode = StatusCodes.Status503ServiceUnavailable;
        await context.Response.WriteAsync("Back soon");
        return;
    }
    await next(context);
});
app.UseRouting();
app.MapStaticAssets();
app.MapGet("/laundry", () => "home");
app.Run();
```

```
503  /laundry  Back soon
503  /css/site.css  Back soon
```

Sebelum perutean, `GetEndpoint()` mengembalikan `null`, sehingga setiap permintaan, termasuk lembar gaya, tampak seperti halaman dan ikut ditutup. Di peramban, halaman pemeliharaan akan

tampil tanpa gaya, karena lembar gayanya sendiri sedang "dipelihara". Kode pemeriksaannya sama persis di kedua versi; hanya posisinya di *pipeline* yang berbeda, dan itulah aturan Bab 3, "urutan itu penting", dalam bentuknya yang paling murni.

4.7.7 Uji diri

Variasi. Selama pemeliharaan, pemilik ingin pelanggan tetap dapat melacak pesanan. Apa yang berubah?

Jawaban. Uji di dalam *middleware*: loloskan setiap *endpoint* yang polanya diawali `css/` *atau* `laundry/track`. Soal Latihan 2 mengerjakannya dan memperlihatkan jawaban nyatanya.

4.8 Memakainya Sungguhan

4.8.1 Etalase untuk usaha Anda sendiri

LaundryShowcase ditulis agar sebagian besar penyesuaian sungguhan terjadi di dua berkas. Tulis ulang `content/site.json` untuk usaha Anda sendiri: kisah sebuah *startup*, janji-janjinya, misinya, timnya, serta apa yang ditawarkan dan diminta dari investor, mitra, dan calon rekan kerja. Ganti berkas di `data/` dengan ekspor milik Anda, atau ubah `LaundryData.Facts()` untuk menghitung angka yang penting bagi usaha Anda: pelanggan yang dilayani, panen yang terjual, pasien yang ditangani. Ubah prefiks di `[Route("laundry")]`, dan warna di `site.css`. Karena setiap tautan dihasilkan dari rute dan setiap kata berasal dari berkas konten, controller dan view jarang perlu berubah. Kelima soal bab ini juga menjadi daftar periksa yang baik untuk situs kecil mana pun: tabel alamat beserta kegagalannya, batasan di setiap alamat yang punya format, privasi pada setiap halaman publik yang mencari sesuatu, satu-dua header dari *middleware*, dan sakelar pemeliharaan yang menjaga lembar gaya tetap hidup.

4.8.2 Halaman pelacakan dan data pribadi

Soal 4.2 menjauhkan data pribadi dari halaman publik dengan tidak pernah memuatnya. Itulah bentuk paling sederhana *minimisasi data*, salah satu prinsip Undang-Undang Pelindungan Data Pribadi (UU PDP), yang mewajibkan data pribadi diproses secara terbatas dan spesifik, untuk tujuan yang jelas [4]. Halaman pelacakan sungguhan biasanya juga memerlukan pemeriksaan kedua, karena empat angka dapat ditebak: Profitina Laundry dapat mencetak satu kata verifikasi acak yang pendek di nota dan memintanya bersama kodenya, dan aksi pelacakan menjawab 404 kecuali keduanya cocok. Pola ini berlaku pula untuk klinik (nomor antrean ditambah tanggal lahir), sekolah (nomor pendaftaran ditambah PIN), dan pencarian publik lainnya.

4.8.3 Profitina dalam praktik

Profitina menjalankan ketiga mekanisme bab ini di produksi. Sakelar pemeliharaan adalah bagian dari setiap rilis bertahap: selama beberapa menit ketika versi baru dipasang di *host* produksinya yang berjalan di Windows, halaman publik menjawab bahwa layanan akan segera kembali, sementara mesin yang menyajikan aset situs tetap bekerja. *Middleware* menambahkan header dan menegakkan batas pada setiap permintaan, termasuk pembatasan laju login yang diwajibkan pekerjaan kepatuhan UU PDP-nya. Dan setiap pencarian publik dirancang dengan memikirkan apa yang *tidak* boleh ditampilkan: halaman komunitas menampilkan sebuah unggahan, nama publik anggota yang menulisnya, dan balasannya, dan tidak ada apa pun yang tidak dipilih anggota itu untuk diterbitkan.

4.9 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

Kelima penyelesaian keliru yang menggoda adalah lima kesalahan paling mendidik di bab ini; **Tabel 4.6** mengumpulkannya beserta gejala masing-masing dan apa yang menangkapnya.

Tabel 4.6 Kelima penyelesaian keliru yang menggoda di bab ini, apa yang dilakukan masing-masing, dan apa yang menangkapnya.

Kesalahan	Gejala	Tertangkap oleh
Garis miring di depan rute aksi	halaman menjawab di alamat yang salah, 404 di alamat yang benar	tabel alamat (tanpa peringatan)
Kurung kurawal tunggal dalam batasan `regex`	*exception* saat awal jalan, tidak ada permintaan yang dijawab	peringatan *build* ASP0017, lalu macet
Dua rute berbentuk sama	500 `AmbiguousMatchException` pada permintaan pertama yang cocok	peringatan *build* ASP0023, lalu macet
Mengisi header sesudah `await next`	header diam-diam hilang, *exception* tercatat per permintaan	membaca log; `curl -D -`
Memeriksa *endpoint* sebelum `UseRouting`	setiap permintaan ditutup, termasuk lembar gaya	menguji `/css/site.css` selama pemeliharaan

Tiga kesalahan lagi sering muncul pada proyek seperti ini.

Mengetik mati angka yang berasal dari data. Etalase yang menulis "hampir 3.000 pesanan setahun" di teksnya sudah keliru sehari setelah ditulis. Hitunglah angka semacam itu, seperti yang dilakukan `LaundryData.Facts()`, atau jangan dicantumkan.

Membiarkan perkakas pengembangan menyala di produksi. Daftar rute pada Soal 4.3 berguna bagi pengembang dan juga bagi penyerang. Petakan *endpoint* semacam itu hanya di dalam `if (app.Environment.IsDevelopment())`; percobaan di lingkungan Production menjawab `/_routes` dengan 404 sementara bagian situs lainnya berjalan normal.

Menganggap halaman galat yang ramah sebagai keberhasilan. View galat yang lupa `Response.StatusCode = status` mengirim "Not found" dengan status 200, dan mesin pencari mengindeksnya sebagai halaman sungguhan. Selalu pertahankan statusnya.

4.10 Ringkasan Bab

- Rutinitas untuk soal kerangka kerja: baca dan rumuskan ulang, tulis tabel alamat (beserta alamat yang harus gagal), sketsa halaman dan data, tulis controller ramping dengan aturan di layanan, jalankan dan uji dengan peramban dan `curl`, tinjau kembali untuk menemukan penyelesaian keliru yang menggoda.
- Soal 4.1 membangun situs lima halaman di bawah satu prefiks *attribute route*, dengan kata-katanya di berkas JSON dan angkanya dihitung dari data, keduanya didaftarkan di container dan disuntikkan ke controller dan view.
- Soal 4.2 menambahkan pelacakan pesanan dengan batasan rute `regex` (kurung kurawal digandakan di dalam templat rute, huruf besar-kecil diabaikan), formulir `GET` yang mengalihkan ke alamat yang dapat ditandai, halaman 404 yang mempertahankan statusnya, dan tanpa data pribadi di halaman publik.
- Soal 4.3 meramalkan tabel rute secara manual dan memeriksanya terhadap daftar milik router sendiri, yang diambil dari `EndpointDataSource` pada rute khusus pengembangan; rute yang rancu ditolak dengan 500.

- Soal 4.4 mengisi header respons dengan benar melalui `Response.OnStarting`, karena header tidak dapat berubah setelah badan dimulai.
- Soal 4.5 menambahkan sakelar pemeliharaan yang dibaca dari konfigurasi dan variabel lingkungan, diletakkan sesudah `UseRouting` agar dapat meloloskan aset statis, dan menjawab 503 dengan `Retry-After`.

4.11 Soal Latihan

1. Tambahkan halaman `/laundry/outlets` ke etalase yang mendaftar ketiga gerai beserta alamat dan teleponnya, dan tambahkan ke menu.

2. Selama pemeliharaan, biarkan pelanggan tetap dapat melacak pesanannya. Lalu minta `/laundry/track/B9999` dan `/laundry/track/X12` dengan pemeliharaan menyala. Apa yang terjadi, dan mengapa? Perbaiki.

3. Jalankan LaundryShowcase di lingkungan Production dan minta `/_routes`. Apa jawabannya, dan baris mana di **Kode 4.5** yang bertanggung jawab?

4. Tambahkan header kedua, `X-Served-By: LaundryShowcase`, pada setiap respons, di *middleware* yang sama dengan `X-Response-Time`.

5. Tuliskan tabel alamat untuk fitur baru: satu halaman per layanan di `/laundry/services/{id}`, dengan id berbentuk seperti `S0004`. Adakah bentrok dengan rute yang sudah ada? Tuliskan atributnya.

6. Halaman perusahaan menampilkan rata-rata pesanan. Satu tempat mana yang Anda ubah untuk menampilkan median, dan mengapa median kadang merupakan angka yang lebih baik untuk sebuah laundry?

4.12 Penyelesaian Soal Latihan

1. Satu aksi di `ShowcaseController`, satu view, dan satu baris menu. Aksinya:

```
    [HttpGet("outlets")]                          // /laundry/outlets
    public IActionResult Outlets() => View(data.Outlets);
```

View `Views/Showcase/Outlets.cshtml`:

```
@model IReadOnlyList<Outlet>
@{
    ViewData["Title"] = "Outlets";
}
<h1>Our outlets</h1>
<table class="list">
    @foreach (var o in Model)
    {
        <tr><th>@o.Name</th><td>@o.Address<br />@o.Phone</td></tr>
    }
</table>
```

dan di `_Layout.cshtml`, `<a class="@Here("Outlets")" asp-action="Outlets">Outlets</a>`. Menjalankan aplikasi yang diubah itu menjawab `/laundry/outlets` dengan ketiga gerai: Rungkut, Jl. Kenari Raya No. 21, 031-0000-2101; Keputih, Jl. Flamboyan No. 8, 031-0000-2102; Wiyung, Jl. Cempaka Indah No. 45, 031-0000-2103.

2. Di *middleware* pemeliharaan, ganti ujinya dengan:

```
    string pattern = endpoint?.RoutePattern.RawText ?? "";
    bool open = pattern.StartsWith("css/") || pattern.StartsWith("laundry/track");
    if (closed && !open)
```

Dengan pemeliharaan menyala, sebuah percobaan memberi `503` untuk `/laundry` dan `/laundry/outlets`, dan `200` untuk `/laundry/track`, `/laundry/track/B0072`, dan `/css/site.css`; `/laundry/track?code=B0072` memberi `302`-nya. Tetapi `/laundry/track/B9999` dan `/laundry/track/X12` sama-sama menjawab `503` dengan "Back soon" alih-alih 404 yang ramah. Untuk `X12` tidak ada *endpoint* yang cocok, sehingga polanya kosong dan permintaannya ditutup. Untuk `B9999` aksi pelacakan berjalan dan mengembalikan 404; *middleware* kode status lalu menjalankan ulang permintaan sebagai `/laundry/error/404`, dan permintaan yang dijalankan ulang itu melewati *middleware* pemeliharaan sekali lagi, tempat `laundry/error/{status:int}` tidak diloloskan. Perbaikannya adalah meloloskan rute galat juga, `|| pattern.StartsWith("laundry/error")`, dan memperlakukan "tidak ada *endpoint*" sebagai 404 alih-alih sebagai halaman: `if (closed && endpoint is not null && !open)`. Percobaan dengan kedua perubahan itu menjawab `/laundry/track/B9999` dan `/laundry/track/X12` dengan halaman ramah "Not found" berstatus 404, `/laundry/track/B0072` dengan 200, dan `/laundry` tetap 503; alamat tak dikenal seperti `/nothing` kini mendapat 404 alih-alih 503, dan itulah jawaban yang jujur. Pelajarannya dari Bab 3: eksekusi ulang menjalankan bagian akhir *pipeline* untuk kedua kalinya, dan setiap *middleware* di bagian itu harus siap menghadapinya.

3. `404`, dengan halaman ramah "Not found", sementara `/laundry` menjawab `200` seperti biasa; keduanya dari menjalankan `ASPNETCORE_ENVIRONMENT=Production dotnet run --no-launch-profile`. Rute itu hanya dipetakan di dalam `if (app.Environment.IsDevelopment())`, sehingga di Production rute itu tidak ada.

4. Di dalam *callback* `OnStarting` pada **Kode 4.20**, tambahkan satu baris:

```
            context.Response.Headers["X-Served-By"] = "LaundryShowcase";
```

Mengisinya langsung sebelum `await next(context)` juga akan berjalan di sini, karena pada titik itu belum ada yang terkirim; menaruh kedua header di *callback* menjaga seluruh pekerjaan header *middleware* itu di satu tempat.

5.

Metode	Alamat	Aksi	Yang diharapkan
GET	`/laundry/services/S0004`	`ServiceDetails`	200, satu layanan
GET	`/laundry/services/s0004`	`ServiceDetails`	200, huruf besar-kecil diabaikan
GET	`/laundry/services/S0099`	`ServiceDetails`	404, halaman ramah
GET	`/laundry/services/abc`	tidak ada	404, batasan gagal

Tidak ada bentrok: `laundry/services` yang sudah ada memiliki satu segmen lebih sedikit. Atributnya `[HttpGet(@"services/{id:regex(^S\d{{4}}$)}")]`, dengan kurung kurawal digandakan seperti pada Soal 4.2.

6. Hanya `LaundryData.Facts()`, yang menghitung angkanya; view menampilkan apa pun yang dipegang model (dan label "Average order" di `Company.cshtml` akan menjadi "Typical order"). Median adalah nilai tengah ketika semua pesanan diurutkan berdasarkan jumlahnya. Beberapa pesanan yang sangat besar, seperti linen mingguan sebuah hotel, menarik rata-rata ke atas tanpa mengubah apa yang dibayar rumah tangga biasa; median tidak terpengaruh olehnya, sehingga lebih baik menggambarkan pelanggan yang umum. Menampilkan keduanya adalah pilihan yang paling jujur.

Daftar Pustaka

[1] G. Pólya, *How to Solve It: A New Aspect of Mathematical Method*, 2nd ed. Princeton, NJ, USA: Princeton University Press, 1957.

[2] Microsoft, "JSON serialization and deserialization in .NET: overview," *.NET Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/dotnet/standard/serialization/system-text-json/overview

[3] Microsoft, "Routing in ASP.NET Core: route constraints and regular expressions," *ASP.NET Core Documentation*, for ASP.NET Core 10.0, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/fundamentals/routing

[4] Republik Indonesia, *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*, Lembaran Negara Republik Indonesia Tahun 2022 Nomor 196, Oct. 17, 2022. [Online]. Available: https://peraturan.bpk.go.id/Details/229798/uu-no-27-tahun-2022

[5] Microsoft, "Handle errors in ASP.NET Core," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/fundamentals/error-handling

[6] Microsoft, "ASP0023: Route conflict detected between controller actions," *ASP.NET Core Diagnostics*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/diagnostics/asp0023

[7] R. Fielding, M. Nottingham, and J. Reschke, "HTTP Semantics," RFC 9110, Internet Engineering Task Force, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9110

[8] Microsoft, "Write custom ASP.NET Core middleware" and "ASP.NET Core middleware," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/fundamentals/middleware/write

[9] Microsoft, "Configuration in ASP.NET Core," *ASP.NET Core Documentation*, accessed Sep. 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/fundamentals/configuration/

Bab 5

ASP.NET Core MVC II: Entity Framework Core dan CRUD Sungguhan

Bab 3 memberi Profitina Laundry sebuah situs web yang menampilkan layanan, gerai, dan penawaran harga, semuanya dibaca dari dua berkas CSV. Bab ini memberi toko itu ingatan. Di akhir bab Anda akan mampu menjelaskan apa yang dikerjakan sebuah *object-relational mapper* dan mengapa Entity Framework Core (EF Core) menjadi pilihan para pengembang ASP.NET Core; merancang basis data relasional kecil untuk pelanggan, layanan, pesanan, dan pembayaran; memasang EF Core 10 ke proyek Bab 3; menggambarkan tabel-tabel itu dengan kelas C# dan sebuah `DbContext`; mengubah kelas-kelas itu menjadi basis data SQLite sungguhan melalui *migration* (migrasi); mengisinya dari berkas CSV Bab 3; menulis *query* LINQ yang diterjemahkan EF Core menjadi SQL; serta membangun halaman *create, read, update, delete* (CRUD) yang lengkap untuk pelanggan, daftar harga bagi pemilik, dan meja kasir yang mencatat pesanan, menggerakkannya melewati proses pencucian, dan menerima pembayaran. Sepanjang jalan Anda akan melihat janji Bab 3 ditepati: *controller* katalog yang ditulis di sana tetap bekerja, tanpa diubah sedikit pun, di atas basis data.

5.1 Posisi Kita Sekarang: Toko Tanpa Ingatan

Bayangkan meja kasir gerai Profitina Laundry di Keputih pukul tujuh pagi. Seorang mahasiswa membawa sekantong pakaian, kasir menimbangnya di timbangan digital (4,5 kg), menulis nama dan nomor telepon pelanggan pada nota karbon bernomor, menyobek lembar teratas untuk pelanggan, lalu mendorong kantong itu ke keranjang biru. Pukul sembilan sudah ada dua puluh keranjang. Tengah hari seorang pelanggan kembali menanyakan apakah cuciannya sudah selesai; kasir membolak-balik buku nota, menemukan halamannya, lalu berjalan ke belakang untuk memeriksa. Malam harinya pemilik menghitung uang di laci dan mencocokkannya, halaman demi halaman, dengan nota. Sebagian pelanggan membayar saat menitipkan, sebagian saat mengambil, sebagian membayar separuh. Kalau ada halaman yang sobek atau jumlah yang salah hitung, tidak ada yang tahu siapa masih berutang berapa.

Situs web Bab 3 tidak membantu sama sekali dalam urusan ini. Situs itu menampilkan dua belas layanan toko, dari *Wash, dry & fold* seharga Rp 8.000 per kg sampai *Helmet cleaning*, tiga gerai di Rungkut, Keputih, dan Wiyung, serta halaman penawaran harga yang menerapkan aturan harga toko. Semua yang diketahuinya berasal dari `data/Service.csv` dan `data/Outlet.csv`, dibaca sekali ketika program dijalankan. Situs itu tidak dapat mengingat satu pelanggan pun, dan ketika pemilik mengubah harga, ia harus menyunting berkas di server lalu menyalakan ulang program.

Yang dibutuhkan toko adalah buku nota, papan harga, dan laci uang di dalam program: tempat setiap pelanggan, setiap pesanan, dan setiap pembayaran dicatat saat itu juga, dapat ditemukan lagi dalam sedetik, dan masih ada besok ketika komputer dinyalakan kembali. Tempat itu adalah basis data, dan bab ini menghubungkan situs web ke sana.

Orang-orang yang terbantu mudah disebutkan. Kasir menemukan pesanan lewat nama atau nomor telepon pelanggan tanpa membolak-balik halaman. Petugas cuci melihat keranjang mana yang masih antre dan menandainya selesai. Pelanggan mendengar sisa tagihan yang tepat. Pemilik melihat di halaman kasir berapa pesanan masuk hari ini dan berapa uang yang diterima, serta mengubah harga cukup sekali, di satu tempat, untuk semua halaman situs.

Bab 3 sengaja menyiapkan momen ini. *Controller*-nya tidak pernah membaca berkas CSV sendiri; mereka meminta layanan dan gerai kepada sebuah *interface*, `ILaundryCatalog`, dan satu-satunya kelas yang tahu soal berkas CSV adalah `CsvCatalog`. Komentarnya berisi janji: Bab 5 menukar berkas CSV di belakangnya dengan basis data sungguhan tanpa menyentuh satu *controller* pun. Subbab 5.8 menepati janji itu.

5.2 Gagasannya dengan Kata-Kata Sederhana

5.2.1 Tabel di satu sisi, objek di sisi lain

Basis data relasional menyimpan data dalam tabel. Sebuah tabel memiliki kolom-kolom bernama dan baris sebanyak apa pun; tabel `Customers` misalnya berkolom `Id`, `Name`, `Phone`, dan `Address`, dengan satu baris per pelanggan. Baris dari tabel yang berbeda dihubungkan lewat nilai: sebuah baris pesanan menyimpan `Id` pelanggannya di kolom bernama `CustomerId`. Bahasa untuk menanyakan apa pun kepada basis data relasional adalah SQL: `SELECT Name FROM Customers WHERE Phone = '081200000101'`.

Program C#, sebaliknya, bekerja dengan objek. Objek `Customer` memiliki properti `Name` dan `Phone`, dan objek `Order` dapat memegang rujukan langsung ke objek `Customer`-nya, bukan sekadar angka. Program ingin menulis `order.Customer.Name`, bukan menjalankan *query* kedua.

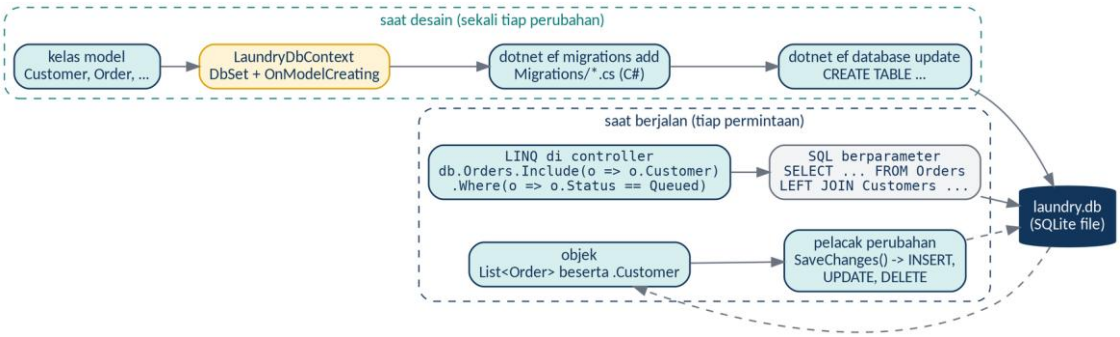
Bab 2 telah memperkenalkan alat yang menjembatani dua dunia itu: *object-relational mapper* (ORM, pemeta objek-relasional). Anda menggambarkan kelas-kelas Anda sekali, lalu ORM menulis SQL-nya, mengirimkannya lengkap dengan parameter, membaca baris-baris yang kembali, dan mengubahnya menjadi objek. Ketika Anda mengubah sebuah objek dan meminta ORM menyimpannya, ORM menentukan perintah SQL `INSERT`, `UPDATE`, atau `DELETE` yang diperlukan. EF Core adalah ORM Microsoft untuk .NET, bersifat sumber terbuka dan dipelihara bersama .NET; versi 10 dirilis bersama .NET 10 pada November 2025 dan, seperti .NET 10, didukung sebagai rilis *long-term support* (dukungan jangka panjang) [1], [2].

> **DEFINISI.** Dalam EF Core, *entity* (entitas) adalah kelas C# yang objek-objeknya disimpan sebagai baris sebuah tabel. *DbContext* adalah kelas yang mewakili satu sesi kerja dengan basis data; ia memiliki properti `DbSet<T>` untuk setiap tabel, misalnya `DbSet<Customer> Customers`. *Migration* (migrasi) adalah berkas C# hasil bangkitan yang menggambarkan satu perubahan struktur basis data (membuat tabel, menambah kolom) dan dapat menerapkan maupun membatalkannya. *Change tracker* (pelacak perubahan) adalah bagian DbContext yang mengingat entitas mana yang dimuat, ditambahkan, diubah, atau dihapus, sehingga `SaveChanges()` tahu SQL apa yang harus dikirim.

5.2.2 Dua saat: saat desain dan saat berjalan

Ada baiknya kita memisahkan dua saat, seperti tampak pada **Gambar 5.1**. Pada *saat desain*, sekali untuk setiap perubahan model, Anda menulis atau menyunting kelas entitas dan DbContext, menjalankan `dotnet ef migrations add` agar EF Core membandingkan model dengan migrasi sebelumnya dan menulis berkas migrasi baru, lalu menjalankan `dotnet ef database update`

untuk menerapkannya, yang mengirim perintah `CREATE TABLE` atau `ALTER TABLE` ke basis data. Pada *saat berjalan*, di setiap permintaan, sebuah *controller* menulis *query* LINQ terhadap sebuah `DbSet`; EF Core menerjemahkannya menjadi satu perintah SQL berparameter, SQLite menjalankannya, dan baris-baris hasilnya kembali sebagai objek. Ketika *controller* mengubah objek lalu memanggil `SaveChangesAsync()`, pelacak perubahan mengubah perubahan-perubahan itu menjadi perintah `INSERT`, `UPDATE`, dan `DELETE`.



Gambar 5.1 Cara EF Core menghubungkan kelas dengan basis data: migrasi pada saat desain, LINQ yang diterjemahkan menjadi SQL dan pelacak perubahan pada saat berjalan.

5.2.3 LINQ adalah bahasa yang sama di mana-mana

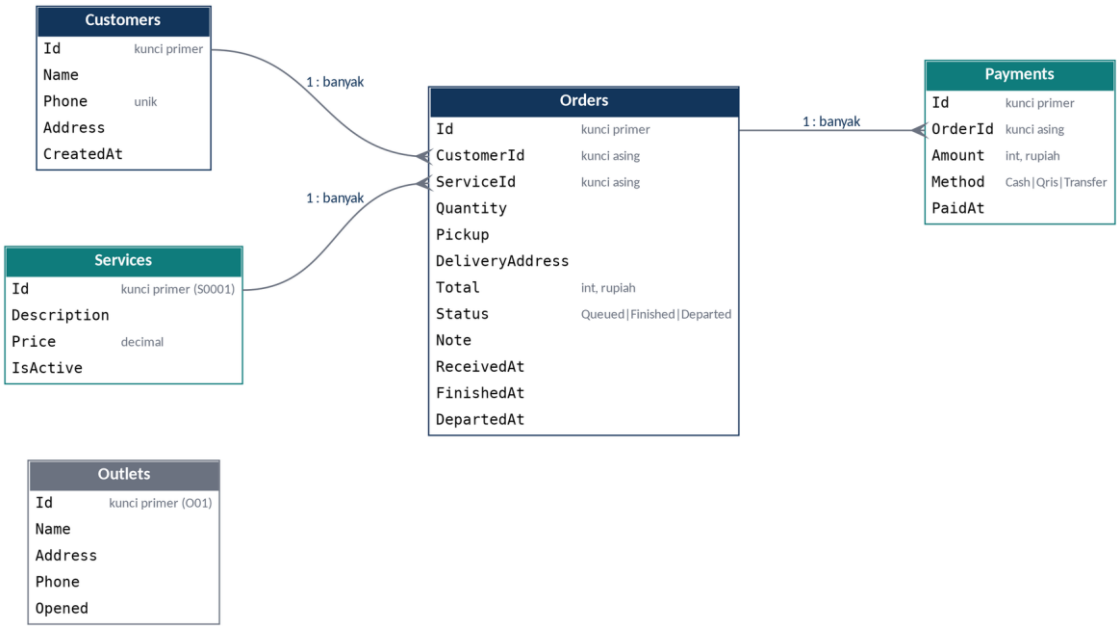
LINQ (*Language-Integrated Query*) adalah himpunan metode seperti `Where`, `OrderBy`, `Select`, `GroupBy`, `Sum`, dan `Count` yang dimiliki setiap koleksi C#. Ketika dipanggil pada sebuah `List<Order>`, C# menjalankannya di memori, satu elemen demi satu elemen. Ketika dipanggil pada sebuah `DbSet<Order>`, pemanggilan metode yang sama itu sama sekali tidak dijalankan: EF Core membacanya sebagai ekspresi, menerjemahkan seluruh rangkaiannya menjadi SQL, dan membiarkan basis data yang bekerja. Kodenya terlihat sama; tempat ia dijalankan berbeda. Itulah sebabnya Subbab 5.4 dimulai tanpa basis data sama sekali: *query* yang Anda pelajari pada *list* adalah *query* yang akan Anda kirim ke SQLite.

5.2.4 Mengapa SQLite

SQLite adalah basis data SQL lengkap dalam satu berkas, tanpa server yang perlu dipasang. SQLite juga menjadi bawaan proyek Laravel di Bab 7, dan memang itulah yang dapat dipakai toko kecil dengan satu komputer di tiap meja kasir. EF Core mendukung banyak basis data melalui *provider* (penyedia); penyedia SQLite adalah paket NuGet `Microsoft.EntityFrameworkCore.Sqlite`, dan pindah ke SQL Server, PostgreSQL, atau MySQL di kemudian hari berarti mengganti paket itu dan satu baris di `Program.cs` (Subbab 5.14).

5.3 Merancang Data

Sebelum satu baris kode pun, datanya dulu. **Gambar 5.2** memperlihatkan lima tabel bab ini dan cara tabel-tabel itu saling terhubung.



Gambar 5.2 Basis data Profitina Laundry pada bab ini: setiap pesanan milik satu pelanggan dan satu layanan, dan sebuah pesanan dapat memiliki banyak pembayaran.

Services dan **Outlets** adalah dua berkas CSV Bab 3, kini berupa tabel. Keduanya mempertahankan kode yang sudah dipakai toko, `S0001` sampai `S0012` dan `O01` sampai `O03`, sebagai *primary key* (kunci primer): kolom yang nilainya mengenali sebuah baris, dan kode yang sudah biasa ditulis orang di nota adalah kunci yang baik. Kolom baru `IsActive` memungkinkan pemilik berhenti menawarkan sebuah layanan tanpa menghapusnya, karena pesanan lama masih merujuk kepadanya.

Customers mendapat nomor yang dibagikan oleh basis data (`1`, `2`, `3`, …). Nomor telepon bersifat *unik*: dua pelanggan tidak boleh memakai nomor yang sama, karena kasir mencari pelanggan lewat nomor telepon.

Orders adalah jantung toko. Setiap pesanan menyimpan `Id` pelanggannya di `CustomerId` dan kode layanannya di `ServiceId`; kolom semacam itu disebut *foreign key* (kunci asing), dan basis data menolak pesanan yang `CustomerId`-nya tidak ada. Pesanan juga menyimpan jumlah (kilogram, potong, atau pasang), apakah toko menjemput dan mengantar, alamat antar, catatan, harga total, status, dan tiga cap waktu. Status mengikuti alur baku toko: `Queued` saat kantong diterima, `Finished` saat sudah dicuci dan dilipat, `Departed` saat sudah meninggalkan toko bersama pemiliknya.

Payments masing-masing milik satu pesanan. Pembayaran bisa lebih dari satu: pelanggan membayar Rp 20.000 saat menitipkan dan sisanya saat mengambil. Setiap pembayaran mencatat jumlah, cara bayar (tunai, QRIS, atau transfer bank), dan waktunya.

Dua keputusan rancangan layak dijelaskan dengan satu kalimat masing-masing. Pertama, pesanan menyimpan `Total`-nya pada saat dibuat, alih-alih menghitungnya ulang dari harga layanan setiap kali. Harga berubah; nota bulan lalu tidak boleh ikut berubah. Kedua, jumlah yang sudah dibayar pelanggan sama sekali tidak disimpan di pesanan; nilainya adalah jumlah semua pembayaran pesanan itu, dan sisa tagihan mengikutinya:

$$\text{Sisa} = \text{Total} - \sum_{p \in \text{pembayaran}} \text{Jumlah}_p \tag{5.1}$$

Menyimpannya dua kali (kolom `Paid` *dan* pembayaran-pembayarannya) mengundang keduanya berselisih suatu hari nanti. Persamaan (5.1) dihitung oleh properti `Paid` dan `Balance` pada kelas `Order` di Subbab 5.6.

Uang disimpan sebagai rupiah utuh dalam kolom `int`. Dalam keseharian rupiah tidak mengenal koin di bawah Rp 100, sehingga pecahan tidak pernah muncul, dan ada pula alasan praktis: SQLite tidak memiliki tipe desimal, dan penyedia SQLite EF Core dapat membaca, menulis, dan membandingkan nilai `decimal` untuk kesamaan, tetapi tidak dapat mengurutkan atau menjumlahkannya di dalam SQL [3]. Dengan kolom `int`, `SUM(Amount)` dijalankan di basis data. Harga layanan tetap bertipe `decimal` seperti di Bab 3, karena hanya dibaca dan ditulis, tidak pernah dijumlahkan.

Tabel 5.1 memperlihatkan bagaimana potongan-potongan itu saling berpadanan.

Tabel 5.1 Padanan antara model C# dan tabel SQLite.

Di C#	Di basis data	Contoh
kelas `Customer`	tabel `Customers`	
objek `new Customer { Name = "Dewi Lestari" }`	satu baris	`3, 'Dewi Lestari', '081200000103', …`
properti `Phone`	kolom `Phone TEXT NOT NULL`	`'081200000103'`
properti `Orders` (sebuah `List<Order>`)	tidak ada: baris-baris `Orders` dengan `CustomerId = 3`	properti navigasi
properti `CustomerId` di `Order`	kolom kunci asing	`REFERENCES Customers(Id)`
enum `OrderStatus.Queued`	teks `'Queued'`	konverter nilai
`db.Customers.Add(c); SaveChanges()`	`INSERT INTO Customers …`	

5.4 LINQ Dulu, Tanpa Basis Data

Cara tercepat untuk memercayai LINQ adalah melihatnya bekerja pada *list* biasa. `LinqDemo` adalah program konsol yang mengompilasi kelas-kelas model milik aplikasi web itu sendiri (berkas proyeknya menyertakan `../LaundryMvc/Models/*.cs`) dan menjalankan jenis *query* yang akan dikirim *controller* bab ini ke basis data: pencarian, penyaringan menurut status, pendapatan per layanan, dan sisa tagihan. **Kode 5.1** adalah seluruh programnya.

Kode 5.1 `LinqDemo/Program.cs`: *query* bab ini pada *list* di memori, dengan kelas-kelas model milik aplikasi web.

```
using LaundryMvc.Models;

// The same LINQ the controllers send to EF Core, run here on plain lists.
var fold = new Service { Id = "S0001", Price = 8000,
                         Description = "Wash, dry & fold (per kg)" };
var express = new Service { Id = "S0002", Price = 12000,
                            Description = "Express wash & fold, 1 day (per kg)" };
var cover = new Service { Id = "S0008", Price = 35000,
                          Description = "Bed cover, any size (per item)" };
var sari = new Customer { Id = 1, Name = "Sari Wulandari",
                          Phone = "081200000101" };
var budi = new Customer { Id = 2, Name = "Budi Santoso",
                          Phone = "081200000102" };
var dewi = new Customer { Id = 3, Name = "Dewi Lestari",
                          Phone = "081200000103" };

var orders = new List<Order>
{
    new() { Id = 1, Customer = sari, Service = fold, Quantity = 4.5m,
            Total = 36000, Status = OrderStatus.Finished,
            Payments = [ new() { Amount = 36000,
                                 Method = PaymentMethod.Qris } ] },
    new() { Id = 2, Customer = budi, Service = express, Quantity = 3m,
            Total = 36000,
            Payments = [ new() { Amount = 20000,
                                 Method = PaymentMethod.Cash } ] },
    new() { Id = 3, Customer = budi, Service = cover, Quantity = 2m,
            Pickup = true, Total = 80000 },
    new() { Id = 4, Customer = dewi, Service = fold, Quantity = 6m,
            Total = 48000, Status = OrderStatus.Departed,
            Payments = [ new() { Amount = 48000,
                                 Method = PaymentMethod.Transfer } ] },
};
var customers = new List<Customer> { sari, budi, dewi };

Console.WriteLine("1. Customers whose name or phone contains \"an\":");
var search = "an";
foreach (var c in customers.Where(c => c.Name.Contains(search)
                                    || c.Phone.Contains(search))
                           .OrderBy(c => c.Name))
{
    Console.WriteLine($"   {c.Name,-16} {c.Phone}");
}
```

Kode 5.1 (Lanjutan) *LinqDemo/Program.cs: query bab ini pada list di memori, dengan kelas-kelas model milik aplikasi web.*

```
Console.WriteLine("2. Orders still in the queue:");
foreach (var o in orders.Where(o => o.Status == OrderStatus.Queued))
{
    Console.WriteLine($"   #{o.Id} {o.Customer!.Name,-16} {o.Service!.Name}");
}

Console.WriteLine("3. Revenue per service (sum of order totals):");
var perService = orders
    .GroupBy(o => o.Service!.Name)
    .Select(g => new { Service = g.Key, Orders = g.Count(),
                       Revenue = g.Sum(o => o.Total) })
    .OrderByDescending(r => r.Revenue);
foreach (var r in perService)
{
    Console.WriteLine($"   {r.Service,-26} {r.Orders} order(s) "
                      + $"{Money.Rp(r.Revenue),10}");
}

Console.WriteLine("4. Orders with money still owed:");
foreach (var o in orders.Where(o => o.Balance > 0))
{
    Console.WriteLine($"   #{o.Id} total {Money.Rp(o.Total),9}"
                      + $"  paid {Money.Rp(o.Paid),9}"
                      + $"  owes {Money.Rp(o.Balance),9}");
}

Console.WriteLine($"5. Total owed to the shop: "
                  + Money.Rp(orders.Sum(o => o.Balance)));
```

Jalankan dengan `dotnet run` di folder `LinqDemo`. Inilah keluaran aslinya:

```
1. Customers whose name or phone contains "an":
   Budi Santoso     081200000102
   Sari Wulandari   081200000101
2. Orders still in the queue:
   #2 Budi Santoso     Express wash & fold, 1 day
   #3 Budi Santoso     Bed cover, any size
3. Revenue per service (sum of order totals):
   Wash, dry & fold           2 order(s)  Rp 84.000
   Bed cover, any size        1 order(s)  Rp 80.000
   Express wash & fold, 1 day 1 order(s)  Rp 36.000
4. Orders with money still owed:
   #2 total Rp 36.000  paid Rp 20.000  owes Rp 16.000
   #3 total Rp 80.000  paid      Rp 0  owes Rp 80.000
5. Total owed to the shop: Rp 96.000
```

Bacalah program itu perlahan, karena setiap gagasannya akan muncul lagi. `Where(c => c.Name.Contains(search) || c.Phone.Contains(search))` mempertahankan pelanggan yang membuat *lambda* bernilai `true`; *lambda* `c => …` adalah fungsi kecil yang ditulis di tempat, dengan `c` mewakili satu pelanggan. `OrderBy(c => c.Name)` mengurutkan. `GroupBy(o => o.Service!.Name)` membuat satu kelompok per nama layanan, dan `Select(g => new { … g.Count(), g.Sum(o => o.Total) })` mengubah setiap kelompok menjadi objek ringkasan kecil. *Query* 4 memakai `o.Balance`, properti yang dihitung dari pembayaran pesanan, persis seperti bunyi Persamaan (5.1): pesanan 2 berharga Rp 36.000, sudah dibayar Rp 20.000, masih kurang Rp 16.000.

Perhatikan pula apa yang diperlihatkan keluaran itu tentang aturan harga Bab 3: pesanan *bed cover* (dua potong dengan antar-jemput) berharga Rp 80.000, yaitu 2 × Rp 35.000 ditambah biaya

antar-jemput Rp 10.000, karena subtotalnya di bawah Rp 100.000, batas antar-jemput menjadi gratis.

5.5 Memasang Entity Framework Core

EF Core bukan bagian dari *shared framework* ASP.NET Core; ia didistribusikan sebagai paket NuGet. Buka terminal di folder `LaundryMvc` (di VS Code: Terminal → New Terminal) lalu tambahkan dua paket dan satu alat:

```
dotnet add package Microsoft.EntityFrameworkCore.Sqlite --version 10.0.12
dotnet add package Microsoft.EntityFrameworkCore.Design --version 10.0.12
dotnet tool install --global dotnet-ef --version 10.0.12
```

Paket pertama adalah penyedia SQLite, yang sekaligus membawa EF Core. Paket kedua berisi layanan saat desain yang dibutuhkan `dotnet ef` untuk membaca model Anda dan menulis migrasi; paket ini ditandai `PrivateAssets=all` agar tidak ikut terkirim bersama aplikasi yang dipublikasikan. Baris ketiga memasang perintah `dotnet ef` sekali per komputer. Versi 10.0.12, dirilis 8 September 2026, adalah *patch* terbaru EF Core 10 ketika bab ini ditulis [4]; *patch* 10.0.x yang lebih baru bekerja dengan cara yang sama, dan `dotnet tool update --global dotnet-ef` menjaga alat itu tetap mutakhir. **Kode 5.2** memperlihatkan berkas proyek sesudahnya.

Kode 5.2 `LaundryMvc.csproj` setelah dua paket EF Core ditambahkan.

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

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

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite"
                      Version="10.0.12" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design"
                      Version="10.0.12">
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
  </ItemGroup>

</Project>
```

Connection string (untai koneksi), yaitu teks yang memberi tahu penyedia basis data mana yang harus dibuka, diletakkan di `appsettings.json` (**Kode 5.3**). `Data Source=laundry.db` berarti berkas bernama `laundry.db` di folder tempat aplikasi dijalankan. Kategori log tambahan membuat EF Core mencetak setiap perintah SQL yang dikirimnya, cara terbaik untuk mempelajari menjadi apa LINQ Anda.

Kode 5.3 `appsettings.json`: untai koneksi dan log SQL.

```
{
  "ConnectionStrings": {
    "Laundry": "Data Source=laundry.db"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning",
      "Microsoft.EntityFrameworkCore.Database.Command": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

5.6 Kelas-Kelas Model

5.6.1 Service: dari *record* menjadi entitas

Di Bab 3, `Service` adalah *positional record*, `record Service(string Id, string Description, decimal Price)`. *Record* sangat cocok untuk data yang dibaca sekali dan tidak pernah berubah. Entitas yang disunting pemilik berbeda: EF Core memuatnya, *controller* mengubah `Price`-nya, lalu pelacak perubahan menyadari perubahan itu dan mengirim `UPDATE`. Untuk itu propertinya memerlukan *setter*. **Kode 5.4** adalah kelas barunya. Properti terhitung `Name`, `PerKg`, dan `Unit` disalin tanpa perubahan dari Bab 3, sehingga setiap *view* yang ditulis di sana tetap berfungsi; karena tidak memiliki *setter*, EF Core tidak memetakannya ke kolom.

Kode 5.4 `Models/Service.cs`: *record* Bab 3 yang diubah menjadi kelas entitas dengan *setter* dan penanda `IsActive`.

```
namespace LaundryMvc.Models;

// Chapter 3 read services from a CSV file as read-only records. From
// Chapter 5 on they are rows the owner edits, so the properties have setters.
public class Service
{
    public string Id { get; set; } = "";           // S0001, S0002, ...
    public string Description { get; set; } = "";  // "Ironing only (per kg)"
    public decimal Price { get; set; }             // rupiah per kg/item/pair
    public bool IsActive { get; set; } = true;

    // Worked out from Description, as in Chapter 3; not stored.
    public string Name => Description.Contains(" (")
        ? Description[..Description.LastIndexOf(" (")] : Description;

    public bool PerKg => Description.EndsWith("(per kg)");

    public string Unit => PerKg ? "kg"
        : Description.EndsWith("(per pair)") ? "pair" : "item";
}
```

Satu-satunya berkas Bab 3 lain yang harus berubah adalah `CsvCatalog`, yang dulu membuat layanan dengan `new Service(id, description, price)`; kini ia memakai *object initializer*, `new Service { Id = f[0], Description = f[1], Price = … }`.

5.6.2 Customer, Order, dan Payment

Kode 5.5, **Kode 5.6**, dan **Kode 5.7** adalah tiga entitas baru. EF Core mengenali banyak hal melalui *konvensi*, sehingga kelas-kelasnya tetap sederhana: properti bernama `Id` adalah kunci primer, dan karena bertipe `int`, SQLite memberi nomor baris baru secara otomatis; properti `CustomerId` di

samping properti `Customer` bertipe `Customer` adalah kunci asing beserta properti navigasinya; sebuah `List<Order>` di `Customer` adalah sisi lain dari relasi yang sama. `string` yang tidak *nullable* menjadi kolom `NOT NULL`, sedangkan `string?` menjadi kolom yang boleh kosong.

Kode 5.5 `Models/Customer.cs`: seorang pelanggan dan properti navigasi ke pesanan-pesanannya.

```
namespace LaundryMvc.Models;

public class Customer
{
    public int Id { get; set; }
    public string Name { get; set; } = "";
    public string Phone { get; set; } = "";
    public string? Address { get; set; }
    public DateTime CreatedAt { get; set; } = DateTime.Now;

    // Navigation property: every order this customer has brought in.
    public List<Order> Orders { get; set; } = new();
}
```

Kode 5.6 `Models/Order.cs`: pesanan dengan kunci asing, status, cap waktu, dan pembayarannya; `Paid` dan `Balance` menerapkan rumus sisa tagihan.

```
namespace LaundryMvc.Models;

public enum OrderStatus { Queued, Finished, Departed }

public class Order
{
    public int Id { get; set; }

    public int CustomerId { get; set; }          // foreign key
    public Customer? Customer { get; set; }      // navigation property

    public string ServiceId { get; set; } = "";  // S0001, ...
    public Service? Service { get; set; }

    public decimal Quantity { get; set; }        // kg, items or pairs
    public bool Pickup { get; set; }             // pickup and delivery
    public string? DeliveryAddress { get; set; }
    public int Total { get; set; }               // rupiah, fixed at creation
    public OrderStatus Status { get; set; } = OrderStatus.Queued;
    public string? Note { get; set; }

    public DateTime ReceivedAt { get; set; } = DateTime.Now;
    public DateTime? FinishedAt { get; set; }
    public DateTime? DepartedAt { get; set; }

    public List<Payment> Payments { get; set; } = new();

    // Worked out from the loaded payments; not stored.
    public int Paid => Payments.Sum(p => p.Amount);
    public int Balance => Total - Paid;
}
```

Kode 5.7 `Models/Payment.cs`: satu pembayaran untuk satu pesanan.

```
namespace LaundryMvc.Models;

public enum PaymentMethod { Cash, Qris, Transfer }

public class Payment
{
    public int Id { get; set; }
    public int OrderId { get; set; }
    public Order? Order { get; set; }
    public int Amount { get; set; }              // rupiah
    public PaymentMethod Method { get; set; }
    public DateTime PaidAt { get; set; } = DateTime.Now;
}
```

Properti navigasi `Customer?` dan `Service?` bersifat *nullable* karena hanya terisi ketika Anda meminta EF Core memuatnya (Subbab 5.10.2). `Paid` dan `Balance` menjumlahkan pembayaran apa pun yang ada di *list* `Payments`; jika pembayarannya tidak dimuat, *list* itu kosong dan sisa tagihan tampak seperti seluruh total. Ingat hal ini; Bab 8 menjadikannya soal yang harus dipecahkan.

5.7 DbContext

Kode 5.8 adalah DbContext milik laundry. Ia memiliki satu `DbSet` per tabel dan satu metode, `OnModelCreating`, tempat *fluent API* menyatakan apa yang tidak dapat ditebak oleh konvensi.

Kode 5.8 `Data/LaundryDbContext.cs`: lima tabel dan aturan-aturan yang tidak dapat ditebak konvensi.

```
using LaundryMvc.Models;
using Microsoft.EntityFrameworkCore;

namespace LaundryMvc.Data;

public class LaundryDbContext(DbContextOptions<LaundryDbContext> options)
    : DbContext(options)
{
    public DbSet<Service> Services => Set<Service>();
    public DbSet<Outlet> Outlets => Set<Outlet>();
    public DbSet<Customer> Customers => Set<Customer>();
    public DbSet<Order> Orders => Set<Order>();
    public DbSet<Payment> Payments => Set<Payment>();

    protected override void OnModelCreating(ModelBuilder model)
    {
        model.Entity<Service>(e =>
        {
            e.Property(s => s.Id).HasMaxLength(5);
            e.Property(s => s.Description).HasMaxLength(60);
            e.Property(s => s.Price).HasPrecision(10, 2);
        });

        model.Entity<Outlet>(e =>
        {
            e.Property(o => o.Id).HasMaxLength(3);
            e.Property(o => o.Name).HasMaxLength(60);
            e.Property(o => o.Address).HasMaxLength(120);
            e.Property(o => o.Phone).HasMaxLength(20);
        });

        model.Entity<Customer>(e =>
        {
            e.Property(c => c.Name).HasMaxLength(80);
            e.Property(c => c.Phone).HasMaxLength(20);
            e.Property(c => c.Address).HasMaxLength(200);
            e.HasIndex(c => c.Phone).IsUnique();
        });
```

Kode 5.8 (Lanjutan) *Data/LaundryDbContext.cs: lima tabel dan aturan-aturan yang tidak dapat ditebak konvensi.*

```
        model.Entity<Order>(e =>
        {
            e.Property(o => o.Quantity).HasPrecision(6, 2);
            e.Property(o => o.Status).HasConversion<string>().HasMaxLength(12);
            e.Property(o => o.DeliveryAddress).HasMaxLength(200);
            e.Property(o => o.Note).HasMaxLength(200);
            e.HasIndex(o => o.Status);
            e.HasOne(o => o.Customer).WithMany(c => c.Orders)
             .HasForeignKey(o => o.CustomerId)
             .OnDelete(DeleteBehavior.Restrict);
            e.HasOne(o => o.Service).WithMany()
             .HasForeignKey(o => o.ServiceId)
             .OnDelete(DeleteBehavior.Restrict);
        });

        model.Entity<Payment>(e =>
        {
            e.Property(p => p.Method).HasConversion<string>().HasMaxLength(10);
            e.HasOne(p => p.Order).WithMany(o => o.Payments)
             .HasForeignKey(p => p.OrderId)
             .OnDelete(DeleteBehavior.Cascade);
        });
    }
}
```

Telusuri `OnModelCreating` satu entitas demi satu entitas.

- `HasMaxLength` memberi panjang pada setiap kolom teks. SQLite mengabaikan panjang, tetapi SQL Server dan MySQL tidak, dan panjang itu juga mendokumentasikan rancangan.
- `HasIndex(c => c.Phone).IsUnique()` membuat indeks unik: basis data sendiri menolak pelanggan kedua dengan nomor telepon yang sama, sekalipun ada *bug* di *controller* yang lupa memeriksanya.
- `HasPrecision(6, 2)` menyatakan bahwa jumlah memiliki paling banyak enam digit, dua di antaranya di belakang koma, sehingga 4,5 kg dan 1250,75 kg dapat disimpan.
- `HasConversion<string>()` menyimpan enum `OrderStatus` sebagai teks `'Queued'`, `'Finished'`, atau `'Departed'`, bukan angka 0, 1, dan 2. Angka akan rusak diam-diam bila suatu saat seseorang mengubah urutan enum; teks juga mudah dibaca di penampil basis data mana pun.
- `HasOne(o => o.Customer).WithMany(c => c.Orders).HasForeignKey(o => o.CustomerId)` menuliskan relasi yang sebenarnya juga akan ditemukan konvensi, dan `OnDelete(DeleteBehavior.Restrict)` mengubah satu nilai bawaan yang penting. Untuk relasi wajib, EF Core secara bawaan memakai *cascade delete* (hapus berantai): menghapus seorang pelanggan akan menghapus semua pesanannya, dan bersamanya catatan uang yang telah diterima. `Restrict` membuat basis data menolak menghapus pelanggan yang memiliki pesanan. Sebaliknya, pembayaran memakai `Cascade`: pembayaran tidak bermakna tanpa pesanannya.

Konstruktor primer `LaundryDbContext(DbContextOptions<LaundryDbContext> options)` menerima opsinya (penyedia mana, untai koneksi mana) dari *dependency injection*, mekanisme yang dibahas di Bab 2. DbContext tidak pernah memutuskan sendiri basis data mana yang dibuka, sehingga kelas yang sama bekerja dengan SQLite saat pengembangan dan dengan basis data lain di produksi.

5.8 Mengisi Basis Data dan Menepati Janji Bab 3

5.8.1 Data awal

Basis data kosong tidak berguna untuk belajar. EF Core 9 memperkenalkan `UseSeeding` dan `UseAsyncSeeding`, dua *callback* yang dijalankan setiap kali basis data dibuat atau dimigrasikan, termasuk oleh `dotnet ef database update` [5]. `SeedData` (**Kode 5.9**) mengisi basis data kosong dalam dua langkah: layanan dan gerai dari berkas CSV Bab 3 itu sendiri, dibaca oleh `CsvCatalog` yang tidak berubah, lalu tiga pelanggan contoh beserta pesanan dan pembayarannya. Setiap langkah memeriksa dulu apakah tabelnya kosong, karena *callback* itu berjalan pada setiap migrasi, bukan hanya yang pertama.

Kode 5.9 `Data/SeedData.cs`: layanan dan gerai dari berkas CSV Bab 3, lalu pelanggan, pesanan, dan pembayaran contoh.

```
using LaundryMvc.Models;
using Microsoft.EntityFrameworkCore;

namespace LaundryMvc.Data;

// Fills an empty database: services and outlets from the CSV files of
// Chapter 3, then a few customers with orders and payments to look at.
public static class SeedData
{
    public static void Apply(LaundryDbContext db, string dataFolder)
    {
        if (!db.Services.Any())
        {
            var csv = new CsvCatalog(dataFolder);
            db.Services.AddRange(csv.Services());
            db.Outlets.AddRange(csv.Outlets());
            db.SaveChanges();
        }
        if (!db.Customers.Any())
        {
            db.Customers.AddRange(SampleCustomers());
            db.SaveChanges();
        }
    }

    public static async Task ApplyAsync(LaundryDbContext db, string dataFolder,
                                        CancellationToken ct)
    {
        if (!await db.Services.AnyAsync(ct))
        {
            var csv = new CsvCatalog(dataFolder);
            db.Services.AddRange(csv.Services());
            db.Outlets.AddRange(csv.Outlets());
            await db.SaveChangesAsync(ct);
        }
        if (!await db.Customers.AnyAsync(ct))
        {
            db.Customers.AddRange(SampleCustomers());
            await db.SaveChangesAsync(ct);
        }
    }
```

Kode 5.9 (Lanjutan) *Data/SeedData.cs: layanan dan gerai dari berkas CSV Bab 3, lalu pelanggan, pesanan, dan pembayaran contoh.*

```
    static List<Customer> SampleCustomers()
    {
        var today = DateTime.Today;
        return
        [
            new Customer
            {
                Name = "Sari Wulandari", Phone = "081200000101",
                Address = "Jl. Kertajaya 12, Surabaya",
                Orders =
                [
                    new Order
                    {
                        ServiceId = "S0001", Quantity = 4.5m, Total = 36000,
                        Status = OrderStatus.Finished,
                        ReceivedAt = today.AddDays(-1).AddHours(8),
                        FinishedAt = today.AddHours(9),
                        Payments = [ new Payment { Amount = 36000,
                            Method = PaymentMethod.Qris,
                            PaidAt = today.AddDays(-1).AddHours(8) } ]
                    }
                ]
            },
            new Customer
            {
                Name = "Budi Santoso", Phone = "081200000102",
                Orders =
                [
                    new Order
                    {
                        ServiceId = "S0002", Quantity = 3m, Total = 36000,
                        ReceivedAt = today.AddHours(7.5),
                        Payments = [ new Payment { Amount = 20000,
                            Method = PaymentMethod.Cash,
                            PaidAt = today.AddHours(7.5) } ]
                    },
                    new Order
                    {
                        ServiceId = "S0008", Quantity = 2m, Pickup = true,
                        DeliveryAddress = "Jl. Mulyosari 30, Surabaya",
                        Total = 80000, ReceivedAt = today.AddHours(10)
                    }
                ]
            },
            new Customer
            {
```

Kode 5.9 (Lanjutan) *Data/SeedData.cs: layanan dan gerai dari berkas CSV Bab 3, lalu pelanggan, pesanan, dan pembayaran contoh.*

```
                Name = "Dewi Lestari", Phone = "081200000103",
                Address = "Jl. Manyar 7, Surabaya"
            }
        ];
    }
}
```

Perhatikan bagaimana satu graf utuh ditambahkan dengan satu `AddRange`: setiap objek `Customer` membawa `Orders`-nya, dan setiap pesanan membawa `Payments`-nya. EF Core menyisipkan pelanggan, membaca kembali `Id` barunya, mengisi `CustomerId` pada pesanan, menyisipkan pesanan, dan melakukan hal yang sama untuk pembayaran, semuanya di dalam satu transaksi.

5.8.2 Katalog di atas basis data

Kode 5.10 adalah kelas yang dijanjikan Bab 3. Kelas ini mengimplementasikan *interface* yang sama dengan `CsvCatalog` dan menjawab empat pertanyaan yang sama dari basis data, bukan dari *list* yang dibaca saat program dimulai. `Services()` kini hanya menampilkan layanan yang ditawarkan; `FindService` mengubah kode menjadi huruf besar agar `/services/s0004` tetap berfungsi seperti ketika memakai berkas CSV.

Kode 5.10 `Data/EfCatalog.cs`: *interface* katalog Bab 3 yang dijawab oleh EF Core.

```
using LaundryMvc.Models;
using Microsoft.EntityFrameworkCore;

namespace LaundryMvc.Data;

// The same questions CsvCatalog answered in Chapter 3, now answered by the
// database. The controllers of Chapter 3 do not notice the difference.
public class EfCatalog(LaundryDbContext db) : ILaundryCatalog
{
    public IReadOnlyList<Service> Services() =>
        db.Services.AsNoTracking().Where(s => s.IsActive)
                   .OrderBy(s => s.Id).ToList();

    public Service? FindService(string id) =>
        db.Services.AsNoTracking()
                   .FirstOrDefault(s => s.Id == id.ToUpperInvariant());

    public IReadOnlyList<Outlet> Outlets() =>
        db.Outlets.AsNoTracking().OrderBy(o => o.Id).ToList();

    public Outlet? FindOutlet(string id) =>
        db.Outlets.AsNoTracking()
                  .FirstOrDefault(o => o.Id == id.ToUpperInvariant());
}
```

Interface Bab 3 bersifat sinkron (`Services()` mengembalikan *list*, bukan `Task`), sehingga `EfCatalog` memakai `ToList()` dan `FirstOrDefault()` yang sinkron. Itu tidak masalah untuk katalog dua belas baris; *controller* baru di bawah memakai versi asinkron, yang membebaskan *thread* server selama basis data bekerja.

5.8.3 Mendaftarkan semuanya di Program.cs

Kode 5.11 adalah `Program.cs` setelah perubahan. Dibandingkan dengan Bab 3, ada dua hal baru: `AddDbContext` mendaftarkan `LaundryDbContext` dengan penyedia SQLite, untai koneksi, dan *callback* pengisian data, sedangkan `AddScoped<ILaundryCatalog, EfCatalog>()` menggantikan `CsvCatalog` yang berupa *singleton*.

Kode 5.11 `Program.cs`: *pipeline* Bab 3 dengan DbContext yang didaftarkan dan katalog yang ditukar.

```
using System.Diagnostics;
using LaundryMvc.Data;
using LaundryMvc.Models;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// 1. Services: everything the application needs, registered once.
builder.Services.AddControllersWithViews();

// Chapter 5: one LaundryDbContext per request, on the SQLite file named in
// appsettings.json. The CSV files of Chapter 3 now only seed the database.
var dataFolder = Path.Combine(builder.Environment.ContentRootPath, "data");
builder.Services.AddDbContext<LaundryDbContext>(options =>
    options.UseSqlite(builder.Configuration.GetConnectionString("Laundry"))
           .UseSeeding((db, _) =>
               SeedData.Apply((LaundryDbContext)db, dataFolder))
           .UseAsyncSeeding((db, _, ct) =>
               SeedData.ApplyAsync((LaundryDbContext)db, dataFolder, ct)));
builder.Services.AddScoped<ILaundryCatalog, EfCatalog>();
builder.Services.AddSingleton<PriceCalculator>();

var app = builder.Build();

// 2. Pipeline: what happens to every request, in this order.
app.Use(async (context, next) =>
{
    var clock = Stopwatch.StartNew();
    await next(context);                 // everything later runs in here
    app.Logger.LogInformation("{Method} {Path}{Query} -> {Status} in {Ms} ms",
        context.Request.Method, context.Request.Path,
        context.Request.QueryString, context.Response.StatusCode,
        clock.ElapsedMilliseconds);
});

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/error/500");
}
app.UseStatusCodePagesWithReExecute("/error/{0}");

app.UseRouting();
```

Kode 5.11 (Lanjutan) *Program.cs: pipeline Bab 3 dengan DbContext yang didaftarkan dan katalog yang ditukar.*

```
app.MapStaticAssets();
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}")
    .WithStaticAssets();

app.Run();
```

Mengapa *scoped*? `AddDbContext` mendaftarkan DbContext dengan masa hidup *scoped*: satu instans per permintaan HTTP, dibuat ketika permintaan membutuhkannya dan dibuang ketika permintaan selesai. DbContext tidak aman dipakai bersama oleh beberapa *thread* dan pelacak perubahannya membesar dengan setiap entitas yang dimuat, sehingga satu instans yang dipakai bersama oleh semua permintaan (*singleton*) akan salah sekaligus lambat. `EfCatalog` memegang sebuah DbContext, jadi ia juga harus *scoped*; mendaftarkannya sebagai *singleton* akan gagal saat program dimulai dengan pesan tentang memakai layanan *scoped* dari *singleton*, dan itu adalah ASP.NET Core yang melindungi Anda dari kesalahan tersebut.

Sekarang lihat `HomeController`, `ServicesController`, dan `OutletsController` dari Bab 3. Tidak ada satu baris pun yang berubah. Mereka tetap meminta layanan dan gerai kepada `ILaundryCatalog`, dan *dependency injection* kini memberi mereka sebuah `EfCatalog`. Halaman penawaran harga tetap memakai `PriceCalculator` Bab 3, begitu pula meja kasir di bawah. Inilah makna praktis dari "bergantunglah pada *interface*, bukan pada kelas".

5.9 Migrasi: Dari Kelas ke Tabel

Setelah model dan DbContext siap, buat migrasi pertama lalu terapkan:

```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

Perintah pertama membangun proyek, membaca model melalui `LaundryDbContext`, lalu menulis tiga berkas ke folder baru `Migrations`: `<timestamp>_InitialCreate.cs` yang berisi metode `Up` (membuat lima tabel beserta kunci, kunci asing, dan indeksnya) dan metode `Down` (menghapusnya lagi), sebuah berkas *designer* berisi metadata, dan `LaundryDbContextModelSnapshot.cs`, gambaran lengkap model sebagaimana adanya sekarang [6]. Perintah kedua membuka `laundry.db` (membuat berkasnya bila belum ada), menerapkan setiap migrasi yang belum diterapkan, mencatat setiap migrasi yang diterapkan di tabel bernama `__EFMigrationsHistory`, lalu menjalankan *callback* pengisian data.

Buka `Migrations/<timestamp>_InitialCreate.cs` dan bacalah; isinya C# biasa. Anda akan menemukan `migrationBuilder.CreateTable(name: "Customers", …)` dengan satu `table.Column<…>` per properti, `table.PrimaryKey("PK_Customers", x => x.Id)`, `table.ForeignKey(…, onDelete: ReferentialAction.Restrict)` untuk pelanggan pesanan, dan `migrationBuilder.CreateIndex(name: "IX_Customers_Phone", …, unique: true)`. Setiap aturan di Subbab 5.7 ada di sana, ditulis sebagai langkah yang dapat diterapkan dan dibatalkan.

Snapshot itulah yang membuat migrasi *berikutnya* kecil. Misalkan kelak pemilik ingin mencatat di rak mana kantong yang sudah selesai menunggu. Anda menambahkan `public string? Rack { get; set; }` ke `Order` lalu menjalankan `dotnet ef migrations add AddRackToOrders`. EF Core membandingkan model dengan *snapshot*, menemukan tepat satu perbedaan, dan menulis migrasi yang `Up`-nya menambah satu kolom dan `Down`-nya menghapusnya. `dotnet ef database update` hanya menerapkan migrasi itu, karena tabel riwayat menunjukkan bahwa `InitialCreate` sudah dijalankan. Bab 8 melakukan persis hal ini pada Soal 8.7.

> **WAWASAN KUNCI:** Migrasi bagi basis data sama dengan *commit* Git bagi kode. Migrasi itu kecil, berurutan, dapat ditinjau, dan disimpan di repositori, sehingga setiap komputer (laptop rekan kerja, server uji, server toko) dapat dibawa ke struktur yang persis sama dengan satu perintah. Jangan pernah mengubah tabel secara manual di penampil basis data; ubah modelnya, tambahkan migrasi, dan biarkan setiap salinan basis data mengikutinya.

Dua perintah lagi patut diketahui. `dotnet ef migrations remove` menghapus migrasi terakhir bila belum diterapkan, cara biasa untuk memperbaiki migrasi yang baru saja keliru dibuat. `dotnet ef migrations script` mencetak SQL dari semua migrasi, berguna ketika pengelola basis data ingin membaca apa yang akan terjadi sebelum terjadi.

5.10 CRUD Sungguhan untuk Pelanggan

CRUD adalah empat hal yang dilakukan setiap halaman data: **C**reate (membuat), **R**ead (membaca), **U**pdate (memperbarui), dan **D**elete (menghapus). **Tabel 5.2** memuat delapan *action* `CustomersController` beserta alamatnya dan SQL yang ditimbulkan masing-masing.

Tabel 5.2 *Action* `CustomersController`, alamatnya, dan SQL yang ditimbulkannya.

Action	Alamat dan metode	Yang dikirim EF Core
`Index(search)`	`GET /Customers?search=Sari`	`SELECT … WHERE instr(Name, @search) > 0 OR …`
`Details(id)`	`GET /Customers/Details/3`	satu `SELECT` dengan `LEFT JOIN` ke `Orders` dan `Services`
`Create()`	`GET /Customers/Create`	tidak ada: formulir kosong
`Create(customer)`	`POST /Customers/Create`	pemeriksaan telepon, lalu `INSERT INTO Customers …`
`Edit(id)`	`GET /Customers/Edit/3`	`SELECT … WHERE Id = @id`
`Edit(id, input)`	`POST /Customers/Edit/3`	`SELECT`, pemeriksaan telepon, `UPDATE Customers SET …`
`Delete(id)`	`GET /Customers/Delete/3`	`SELECT` beserta pesanan, untuk halaman konfirmasi
`DeleteConfirmed(id)`	`POST /Customers/Delete/3`	`SELECT`, pemeriksaan pesanan, `DELETE FROM Customers …`

Kode 5.12 adalah seluruh *controller*-nya. Bacalah sekali dari atas ke bawah sebelum penjelasannya.

Kode 5.12 `Controllers/CustomersController.cs`: CRUD lengkap untuk pelanggan dengan EF Core.

```
using LaundryMvc.Data;
using LaundryMvc.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace LaundryMvc.Controllers;

public class CustomersController(LaundryDbContext db) : Controller
{
    // GET /Customers?search=Sari
    public async Task<IActionResult> Index(string? search)
    {
        var query = db.Customers.AsNoTracking();
        if (!string.IsNullOrWhiteSpace(search))
        {
            query = query.Where(c => c.Name.Contains(search)
                                  || c.Phone.Contains(search));
        }
        ViewData["Search"] = search;
        return View(await query.OrderBy(c => c.Name).ToListAsync());
    }

    // GET /Customers/Details/3
    public async Task<IActionResult> Details(int id)
    {
        var customer = await db.Customers
            .Include(c => c.Orders).ThenInclude(o => o.Service)
            .AsNoTracking()
            .FirstOrDefaultAsync(c => c.Id == id);
        return customer is null ? NotFound() : View(customer);
    }

    public IActionResult Create() => View(new Customer());

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Create(
        [Bind("Name,Phone,Address")] Customer customer)
    {
        await CheckPhoneIsFreeAsync(customer.Phone, customer.Id);
        if (!ModelState.IsValid) return View(customer);
```

Kode 5.12 (Lanjutan) *Controllers/CustomersController.cs: CRUD lengkap untuk pelanggan dengan EF Core.*

```
        db.Customers.Add(customer);
        await db.SaveChangesAsync();          // INSERT, then customer.Id is set
        TempData["Message"] = $"Customer {customer.Name} added.";
        return RedirectToAction(nameof(Details), new { id = customer.Id });
    }

    public async Task<IActionResult> Edit(int id)
    {
        var customer = await db.Customers.FindAsync(id);
        return customer is null ? NotFound() : View(customer);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(int id,
        [Bind("Id,Name,Phone,Address")] Customer input)
    {
        if (id != input.Id) return BadRequest();
        var customer = await db.Customers.FindAsync(id);
        if (customer is null) return NotFound();

        await CheckPhoneIsFreeAsync(input.Phone, id);
        if (!ModelState.IsValid) return View(input);

        customer.Name = input.Name;           // the tracked entity is changed,
        customer.Phone = input.Phone;         // so SaveChanges sends an UPDATE
        customer.Address = input.Address;     // for exactly these columns
        await db.SaveChangesAsync();
        TempData["Message"] = $"Customer {customer.Name} saved.";
        return RedirectToAction(nameof(Details), new { id });
    }

    public async Task<IActionResult> Delete(int id)
    {
        var customer = await db.Customers.Include(c => c.Orders)
            .AsNoTracking().FirstOrDefaultAsync(c => c.Id == id);
        return customer is null ? NotFound() : View(customer);
    }
```

Kode 5.12 (Lanjutan) *Controllers/CustomersController.cs: CRUD lengkap untuk pelanggan dengan EF Core.*

```
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var customer = await db.Customers.FindAsync(id);
        if (customer is null) return NotFound();
        if (await db.Orders.AnyAsync(o => o.CustomerId == id))
        {
            TempData["Error"] = "A customer with orders cannot be deleted.";
            return RedirectToAction(nameof(Details), new { id });
        }
        db.Customers.Remove(customer);
        await db.SaveChangesAsync();          // DELETE
        TempData["Message"] = $"Customer {customer.Name} deleted.";
        return RedirectToAction(nameof(Index));
    }

    async Task CheckPhoneIsFreeAsync(string phone, int ownId)
    {
        if (await db.Customers.AnyAsync(c => c.Phone == phone && c.Id != ownId))
        {
            ModelState.AddModelError(nameof(Customer.Phone),
                "Another customer already uses this phone number.");
        }
    }
}
```

5.10.1 Mendapatkan DbContext

`public class CustomersController(LaundryDbContext db) : Controller` memakai konstruktor primer: ASP.NET Core membuat *controller* untuk setiap permintaan dan memberinya DbContext milik permintaan itu. *Controller* tidak pernah menulis `new LaundryDbContext(…)`, tidak pernah membuka atau menutup koneksi, dan tidak pernah membuang apa pun; kontainer DI mengurus semuanya.

5.10.2 Read: *query*, AsNoTracking, dan Include

`Index` membangun *query* selangkah demi selangkah. `db.Customers.AsNoTracking()` adalah sebuah `IQueryable<Customer>`: *gambaran* sebuah *query*, belum hasilnya. `if` hanya menambahkan `Where` bila ada kata pencarian; `OrderBy` menambahkan pengurutan; dan baru `ToListAsync()` yang mengirim SQL dan menunggu baris-barisnya. Membangun *query* sepotong-sepotong seperti ini adalah LINQ yang wajar dan tidak memakan biaya apa pun, karena tidak ada yang dikirim sampai akhir.

`AsNoTracking()` memberi tahu EF Core agar tidak mendaftarkan pelanggan yang dimuat ke pelacak perubahan. Halaman daftar hanya menampilkan; pelacakan hanya membuang memori dan waktu. Pakailah pada setiap pembacaan yang tidak akan diikuti `SaveChanges`.

`Contains(search)` diterjemahkan penyedia SQLite menjadi `instr(Name, @search) > 0`, dan nilai `search` dikirim sebagai parameter, tidak pernah ditempelkan ke teks SQL. Itulah yang membuat *query* EF Core aman dari *SQL injection*: pelanggan bernama `'; DROP TABLE Customers; --` hanyalah nama yang aneh. Ada satu efek samping `instr` yang perlu diketahui sekarang: fungsi itu membedakan huruf besar dan kecil, sehingga pencarian `sari` tidak menemukan *Sari Wulandari*. Bab 8 memperbaikinya pada Soal 8.4.

`Details` menampilkan satu pelanggan beserta semua pesanannya dan, untuk setiap pesanan, layanannya. Properti navigasi tidak dimuat secara otomatis; `Include(c => c.Orders)` meminta

pesanan, dan `ThenInclude(o => o.Service)` meminta layanan dari setiap pesanan itu. EF Core mengirim satu perintah SQL dengan `LEFT JOIN` dan menyusun graf objek dari baris-barisnya [7]. `FirstOrDefaultAsync(c => c.Id == id)` mengembalikan `null` bila pelanggan itu tidak ada, dan *action* menjawab dengan `NotFound()`, halaman 404 dari halaman kode status Bab 3.

5.10.3 Create: mengikat, memeriksa, menyimpan

Versi `GET` dari `Create` mengembalikan formulir kosong. Versi `POST` menerima formulir yang sudah diisi sebagai objek `Customer`, dibangun oleh *model binding* dari isian formulir. Atribut `[Bind("Name,Phone,Address")]` mendaftar satu-satunya properti yang boleh diambil dari permintaan. Tanpa atribut itu, pengunjung yang menambahkan isian `CreatedAt=2020-01-01` atau `Id=7` ke formulir dapat mengisi nilai yang tidak pernah ditawarkan halaman; serangan ini disebut *overposting* [8]. Bab 6 mengganti `[Bind]` dengan kelas formulir tersendiri, solusi yang lebih bersih.

`CheckPhoneIsFreeAsync` bertanya kepada basis data apakah pelanggan lain sudah memakai nomor telepon itu, dengan `AnyAsync`, yang menjadi `SELECT EXISTS(…)`. Bila ya, ia menambahkan kesalahan ke `ModelState`, kamus yang mengumpulkan semua yang salah pada permintaan. `ModelState.IsValid` bernilai `false` bila ada yang ditambahkan, termasuk kesalahan yang ditemukan sendiri oleh *model binding*; misalnya, `Name` adalah `string` yang tidak *nullable*, sehingga ASP.NET Core sudah menganggap nama kosong sebagai kesalahan. Bila model tidak sah, *action* menampilkan formulir lagi, dengan nilai yang diketik pengguna dan pesan di samping isiannya.

Ketika semuanya sah, `db.Customers.Add(customer)` memasukkan objek baru ke pelacak perubahan dengan keadaan *Added*, dan `await db.SaveChangesAsync()` mengirim `INSERT`. SQLite memberikan `Id` baru, EF Core menyalinnya ke `customer.Id`, dan *action* mengalihkan ke halaman pelanggan baru itu. Pola *POST, lalu redirect, lalu GET* (PRG) berarti menekan F5 di halaman hasil hanya memuat ulang halaman, bukan mengirim formulir untuk kedua kalinya. `TempData["Message"]` bertahan tepat untuk satu pengalihan itu dan ditampilkan *layout* sebagai bilah hijau.

5.10.4 Update: ubah entitas yang dilacak

`Edit(id, input)` memuat pelanggan dengan `FindAsync(id)`, yang dilacak (tanpa `AsNoTracking` di sini), menyalin tiga nilai yang boleh disunting dari formulir ke entitas itu, lalu memanggil `SaveChangesAsync()`. Pelacak perubahan membandingkan nilai sekarang dengan nilai yang dilihatnya ketika memuat baris itu, menemukan yang berubah, dan mengirim `UPDATE` hanya untuk kolom-kolom tersebut [9]. Bila pengguna hanya mengubah alamat, hanya alamat yang ditulis.

Baris `if (id != input.Id) return BadRequest();` menjaga dari formulir yang isian tersembunyi `Id`-nya diubah agar menunjuk pelanggan lain. Pemeriksaan kecil yang tidak memakan biaya.

5.10.5 Delete: konfirmasi, lalu tolak atau hapus

Menghapus memerlukan dua permintaan: `GET` yang menampilkan halaman konfirmasi dan `POST` yang menghapus. Tautan tidak boleh menghapus apa pun, karena peramban, mesin pencari, dan pratinjau tautan mengikuti tautan dengan sendirinya. `[HttpPost, ActionName("Delete")]` memungkinkan metodenya bernama `DeleteConfirmed` di C# (dua metode bernama `Delete` dengan parameter yang sama tidak akan dapat dikompilasi) sambil tetap menjawab alamat `/Customers/Delete/3`.

Action ini menolak menghapus pelanggan yang memiliki pesanan dan menjelaskan alasannya. Basis data juga akan menolak, karena `DeleteBehavior.Restrict`, tetapi dengan *exception*, bukan pesan yang ramah; memeriksa lebih dulu memberi kasir sebuah kalimat, bukan halaman galat.

5.10.6 *View*-nya

View-nya ditulis dengan Razor, seperti di Bab 3. **Kode 5.13** adalah daftar beserta formulir pencariannya, **Kode 5.14** isian yang dipakai bersama oleh halaman Create dan Edit, dan **Kode 5.15** halaman pelanggan beserta pesanannya.

Kode 5.13 `Views/Customers/Index.cshtml`: daftar pelanggan dengan kotak pencarian.

```
@model List<Customer>
@{
    ViewData["Title"] = "Customers";
}
<h1>Customers</h1>
<form method="get" class="search">
    <input name="search" value="@ViewData["Search"]"
           placeholder="Name or phone" />
    <button type="submit">Search</button>
    <a class="button" asp-action="Create">New customer</a>
</form>
<table class="list">
    <thead>
        <tr><th>Name</th><th>Phone</th><th>Address</th><th></th></tr>
    </thead>
    <tbody>
    @foreach (var c in Model)
    {
        <tr>
            <td><a asp-action="Details" asp-route-id="@c.Id">@c.Name</a></td>
            <td>@c.Phone</td>
            <td>@c.Address</td>
            <td><a asp-action="Edit" asp-route-id="@c.Id">Edit</a>
                <a asp-action="Delete" asp-route-id="@c.Id">Delete</a></td>
        </tr>
    }
    </tbody>
</table>
<p>@Model.Count customer(s).</p>
```

Kode 5.14 `Views/Customers/_Form.cshtml`: isian yang dipakai bersama oleh halaman Create dan Edit.

```
@model Customer
<div asp-validation-summary="ModelOnly" class="errors"></div>
<label asp-for="Name"></label>
<input asp-for="Name" />
<span asp-validation-for="Name" class="error"></span>

<label asp-for="Phone"></label>
<input asp-for="Phone" />
<span asp-validation-for="Phone" class="error"></span>

<label asp-for="Address"></label>
<input asp-for="Address" />
<span asp-validation-for="Address" class="error"></span>
```

Kode 5.15 `Views/Customers/Details.cshtml`: satu pelanggan dan pesanan-pesanannya, yang terbaru di atas.

```
@model Customer
@{
    ViewData["Title"] = Model.Name;
}
<h1>@Model.Name</h1>
<p>@Model.Phone · @(Model.Address ?? "no address")
   · customer since @Model.CreatedAt.ToString("d MMM yyyy")</p>
<p>
    <a class="button" asp-controller="Orders" asp-action="Create"
       asp-route-customerId="@Model.Id">New order</a>
    <a asp-action="Edit" asp-route-id="@Model.Id">Edit</a>
</p>
<h2>Orders</h2>
<table class="list">
    <thead>
        <tr><th>#</th><th>Received</th><th>Service</th><th>Total</th>
            <th>Status</th></tr>
    </thead>
    <tbody>
    @foreach (var o in Model.Orders.OrderByDescending(o => o.ReceivedAt))
    {
        <tr>
            <td><a asp-controller="Orders" asp-action="Details"
                   asp-route-id="@o.Id">@o.Id</a></td>
            <td>@o.ReceivedAt.ToString("dd/MM HH:mm")</td>
            <td>@o.Service?.Name</td>
            <td class="num">@Money.Rp(o.Total)</td>
            <td>@o.Status</td>
        </tr>
    }
    </tbody>
</table>
```

Tag helper dari Bab 3 mengerjakan bagian yang memerlukan ketelitian. `asp-for="Name"` memberi *input* `name` dan `id` yang benar, mengisi nilai saat ini dan, setelah pengiriman yang gagal, nilai yang diketik pengguna; `asp-validation-for` mencetak pesan untuk isian itu. Sebuah `<form method="post">` yang dibangun dengan *tag helper* juga otomatis menerima token *anti-forgery* tersembunyi, dan `[ValidateAntiForgeryToken]` pada *action* `POST` menolak pengiriman apa pun yang tidak membawanya. Itu melindungi toko dari *cross-site request forgery* (pemalsuan permintaan lintas situs): halaman di situs lain tidak dapat membuat peramban kasir yang sedang masuk mengirim formulir ke laundry.

View Create dan Edit (**Kode 5.16** memperlihatkan Create) hanya menambahkan judul halaman, tag formulir, dan tombol-tombol di sekeliling *partial view* bersama.

Kode 5.16 `Views/Customers/Create.cshtml`: halaman pelanggan baru di sekeliling formulir bersama.

```
@model Customer
@{
    ViewData["Title"] = "New customer";
}
<h1>New customer</h1>
<form asp-action="Create" method="post" class="form">
    <partial name="_Form" model="Model" />
    <button type="submit">Save</button>
    <a asp-action="Index">Cancel</a>
</form>
```

5.11 Pesanan, Pembayaran, dan Daftar Harga

5.11.1 Meja kasir

`OrdersController` (**Kode 5.17**) mengubah buku nota menjadi halaman-halaman: daftar yang dapat disaring menurut status, halaman untuk setiap pesanan, formulir pesanan baru, tombol yang menggerakkan pesanan di sepanjang alur kerja, dan formulir kecil untuk mencatat pembayaran.

Kode 5.17 `Controllers/OrdersController.cs`: daftar pesanan, halaman pesanan, pesanan baru yang dihargai oleh kalkulator Bab 3, alur kerja, dan pembayaran.

```
using LaundryMvc.Data;
using LaundryMvc.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;

namespace LaundryMvc.Controllers;

public class OrdersController(LaundryDbContext db, PriceCalculator calculator)
    : Controller
{
    // GET /Orders or /Orders?status=Queued
    public async Task<IActionResult> Index(OrderStatus? status)
    {
        var query = db.Orders
            .Include(o => o.Customer)
            .Include(o => o.Service)
            .Include(o => o.Payments)
            .AsNoTracking();
        if (status is not null)
        {
            query = query.Where(o => o.Status == status);
        }
        ViewData["Status"] = status;
        return View(await query.OrderByDescending(o => o.ReceivedAt)
                               .ToListAsync());
    }

    public async Task<IActionResult> Details(int id)
    {
        var order = await db.Orders
            .Include(o => o.Customer)
            .Include(o => o.Service)
            .Include(o => o.Payments)
            .AsNoTracking()
            .FirstOrDefaultAsync(o => o.Id == id);
        return order is null ? NotFound() : View(order);
    }

    public async Task<IActionResult> Create(int? customerId)
    {
        await FillListsAsync();
        return View(new Order { CustomerId = customerId ?? 0, Quantity = 1 });
    }
```

Kode 5.17 (Lanjutan) *Controllers/OrdersController.cs: daftar pesanan, halaman pesanan, pesanan baru yang dihargai oleh kalkulator Bab 3, alur kerja, dan pembayaran.*

```
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Create([Bind("CustomerId,ServiceId,"
        + "Quantity,Pickup,DeliveryAddress,Note")] Order order)
    {
        var service = await db.Services.FindAsync(order.ServiceId);
        if (service is null || !service.IsActive)
        {
            ModelState.AddModelError(nameof(Order.ServiceId), "Choose a service.");
        }
        if (!await db.Customers.AnyAsync(c => c.Id == order.CustomerId))
        {
            ModelState.AddModelError(nameof(Order.CustomerId),
                "Choose a customer.");
        }
        if (order.Quantity <= 0)
        {
            ModelState.AddModelError(nameof(Order.Quantity),
                "The quantity must be more than zero.");
        }
        if (!ModelState.IsValid || service is null)
        {
            await FillListsAsync();
            return View(order);
        }

        // The price rules of Chapter 3: minimum 3 kg, whole items, pickup fee.
        var quote = calculator.Calculate(service, order.Quantity, order.Pickup);
        order.Total = (int)decimal.Round(quote.Total,
                                         MidpointRounding.AwayFromZero);
        order.Status = OrderStatus.Queued;
        order.ReceivedAt = DateTime.Now;
        db.Orders.Add(order);
        await db.SaveChangesAsync();
        TempData["Message"] = $"Order #{order.Id} created: {Money.Rp(order.Total)}."
            + (quote.Notes.Count > 0 ? " " + string.Join(" ", quote.Notes) : "");
        return RedirectToAction(nameof(Details), new { id = order.Id });
    }
```

Kode 5.17 (Lanjutan) *Controllers/OrdersController.cs: daftar pesanan, halaman pesanan, pesanan baru yang dihargai oleh kalkulator Bab 3, alur kerja, dan pembayaran.*

```
    // Queued -> Finished -> Departed, each step with its time stamp.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Advance(int id)
    {
        var order = await db.Orders.FindAsync(id);
        if (order is null) return NotFound();
        switch (order.Status)
        {
            case OrderStatus.Queued:
                order.Status = OrderStatus.Finished;
                order.FinishedAt = DateTime.Now;
                break;
            case OrderStatus.Finished:
                order.Status = OrderStatus.Departed;
                order.DepartedAt = DateTime.Now;
                break;
            default:
                TempData["Error"] = "This order has already left the shop.";
                return RedirectToAction(nameof(Details), new { id });
        }
        await db.SaveChangesAsync();
        TempData["Message"] = $"Order #{id} is now {order.Status}.";
        return RedirectToAction(nameof(Details), new { id });
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> AddPayment(int id, int amount,
                                                PaymentMethod method)
    {
        if (!await db.Orders.AnyAsync(o => o.Id == id)) return NotFound();
        if (amount <= 0)
        {
            TempData["Error"] = "The amount must be more than zero.";
            return RedirectToAction(nameof(Details), new { id });
        }
        db.Payments.Add(new Payment
        {
            OrderId = id, Amount = amount, Method = method,
            PaidAt = DateTime.Now
        });
        await db.SaveChangesAsync();
        TempData["Message"] = $"Payment of {Money.Rp(amount)} recorded.";
        return RedirectToAction(nameof(Details), new { id });
    }
```

Kode 5.17 (Lanjutan) *Controllers/OrdersController.cs: daftar pesanan, halaman pesanan, pesanan baru yang dihargai oleh kalkulator Bab 3, alur kerja, dan pembayaran.*

```

    async Task FillListsAsync()
    {
        ViewBag.Customers = new SelectList(
            await db.Customers.OrderBy(c => c.Name).ToListAsync(),
            nameof(Customer.Id), nameof(Customer.Name));
        ViewBag.Services = new SelectList(
            await db.Services.Where(s => s.IsActive).OrderBy(s => s.Id)
                             .ToListAsync(),
            nameof(Service.Id), nameof(Service.Description));
    }
}
```

Beberapa rinciannya perlu diperhatikan.

- `Index(OrderStatus? status)` memperlihatkan *model binding* dengan enum: `/Orders?status=Queued` mengisi `status` dengan `OrderStatus.Queued`; nilai yang tidak ada atau tidak dikenal membiarkannya `null`, dan daftar menampilkan semuanya. Tiga pemanggilan `Include` memuat pelanggan, layanan, dan pembayaran setiap pesanan, karena daftar menampilkan nama pelanggan, layanan, dan sisa tagihan.
- `Create` mencari layanan dan pelanggan sebelum menyimpan. Formulir hanya menawarkan layanan yang aktif, tetapi formulir dapat diubah siapa saja dengan *developer tools* peramban, jadi server memeriksa lagi. Harga berasal dari `calculator.Calculate(service, quantity, pickup)`, `PriceCalculator` Bab 3 itu sendiri, yang disuntikkan ke *controller*. Kalkulator itu menerapkan minimum 3 kg untuk layanan per kg, membulatkan barang ke atas menjadi potong utuh, dan menambahkan biaya antar-jemput Rp 10.000 bila di bawah Rp 100.000. Catatan yang dikembalikannya ("Minimum 3 kg, so 2 kg is billed as 3 kg") ditampilkan kepada kasir setelah penyimpanan.
- `Advance` adalah alur kerjanya: sebuah `switch` memindahkan pesanan yang antre menjadi selesai dan pesanan yang selesai menjadi sudah diambil, lalu mengisi cap waktu yang sesuai. Ia memakai `FindAsync`, karena hanya baris pesanan yang diperlukan.
- `AddPayment` menambahkan baris `Payment` langsung melalui `db.Payments.Add`, dengan `Id` pesanan sebagai kunci asing; tidak perlu memuat pesanan dan daftar pembayarannya hanya untuk menambah satu lagi.

Halaman pesanan (**Kode 5.18**) menampilkan fakta-fakta pesanan, tombol untuk langkah alur berikutnya, pembayaran, dan, selama masih ada tagihan, formulir pembayaran yang sudah terisi sisa tagihan.

Kode 5.18 `Views/Orders/Details.cshtml`: satu pesanan, tombol alur kerjanya, pembayarannya, dan formulir pembayaran.

```
@model Order
@{
    ViewData["Title"] = $"Order #{Model.Id}";
}
<h1>Order #@Model.Id · @Model.Status</h1>
<dl class="facts">
    <dt>Customer</dt>
    <dd><a asp-controller="Customers" asp-action="Details"
           asp-route-id="@Model.CustomerId">@Model.Customer?.Name</a>
        (@Model.Customer?.Phone)</dd>
    <dt>Service</dt>
    <dd>@Model.Service?.Name, @Model.Quantity @Model.Service?.Unit</dd>
    <dt>Delivery</dt>
    <dd>@(Model.Pickup ? $"to {Model.DeliveryAddress}"
                       : "collected at the shop")</dd>
    <dt>Received</dt><dd>@Model.ReceivedAt.ToString("dd/MM/yyyy HH:mm")</dd>
    <dt>Finished</dt><dd>@Model.FinishedAt?.ToString("dd/MM/yyyy HH:mm")</dd>
    <dt>Departed</dt><dd>@Model.DepartedAt?.ToString("dd/MM/yyyy HH:mm")</dd>
    <dt>Note</dt><dd>@Model.Note</dd>
    <dt>Total</dt><dd>@Money.Rp(Model.Total)</dd>
    <dt>Paid</dt><dd>@Money.Rp(Model.Paid)</dd>
    <dt>Balance</dt><dd><strong>@Money.Rp(Model.Balance)</strong></dd>
</dl>

@if (Model.Status != OrderStatus.Departed)
{
    <form asp-action="Advance" asp-route-id="@Model.Id" method="post">
        <button type="submit">
            @(Model.Status == OrderStatus.Queued ? "Mark finished"
                                                 : "Mark picked up")
        </button>
    </form>
}

<h2>Payments</h2>
<table class="list">
    <thead><tr><th>Paid at</th><th>Method</th><th class="num">Amount</th></tr>
    </thead>
    <tbody>
    @foreach (var p in Model.Payments.OrderBy(p => p.PaidAt))
    {
        <tr><td>@p.PaidAt.ToString("dd/MM HH:mm")</td><td>@p.Method</td>
            <td class="num">@Money.Rp(p.Amount)</td></tr>
    }
    </tbody>
</table>
```

Kode 5.18 (Lanjutan) *Views/Orders/Details.cshtml: satu pesanan, tombol alur kerjanya, pembayarannya, dan formulir pembayaran.*

```
@if (Model.Balance > 0)
{
    <form asp-action="AddPayment" asp-route-id="@Model.Id" method="post"
          class="inline">
        <input name="amount" type="number" min="1" value="@Model.Balance" />
        <select name="method"
                asp-items="Html.GetEnumSelectList<PaymentMethod>()"></select>
        <button type="submit">Record payment</button>
    </form>
}
```

`Html.GetEnumSelectList<PaymentMethod>()` membangun pilihan *Cash*, *Qris*, dan *Transfer* dari enum, sehingga cara bayar baru hanya memerlukan satu nilai enum baru tanpa HTML baru.

Formulir pesanan baru (**Kode 5.19**) mengisi dua daftar pilihan dari `ViewBag.Customers` dan `ViewBag.Services`, yang disiapkan metode `FillListsAsync` di *controller* sebagai objek `SelectList`. `asp-items` mengubah `SelectList` menjadi elemen `<option>` dan memilih nilai saat ini.

Kode 5.19 `Views/Orders/Create.cshtml`: formulir kasir untuk pesanan baru.

```
@model Order
@{
    ViewData["Title"] = "New order";
}
<h1>New order</h1>
<form asp-action="Create" method="post" class="form">
    <div asp-validation-summary="ModelOnly" class="errors"></div>
    <label asp-for="CustomerId">Customer</label>
    <select asp-for="CustomerId" asp-items="ViewBag.Customers">
        <option value="0">- choose -</option>
    </select>
    <span asp-validation-for="CustomerId" class="error"></span>

    <label asp-for="ServiceId">Service</label>
    <select asp-for="ServiceId" asp-items="ViewBag.Services">
        <option value="">- choose -</option>
    </select>
    <span asp-validation-for="ServiceId" class="error"></span>

    <label asp-for="Quantity">Quantity (kg, items or pairs)</label>
    <input asp-for="Quantity" type="number" step="0.01" min="0.01" />
    <span asp-validation-for="Quantity" class="error"></span>

    <label class="check"><input asp-for="Pickup" /> Pickup and delivery</label>
    <label asp-for="DeliveryAddress">Delivery address</label>
    <input asp-for="DeliveryAddress" />

    <label asp-for="Note"></label>
    <input asp-for="Note" />

    <button type="submit">Create order</button>
    <a asp-action="Index">Cancel</a>
</form>
```

5.11.2 Daftar harga pemilik dan halaman kasir

`PricesController` (**Kode 5.20**) memungkinkan pemilik menambah layanan serta mengubah deskripsi, harga, atau penanda `IsActive`. Ia bekerja pada tabel `Services` yang sama dengan yang kini dibaca halaman publik `/services` dari Bab 3, sehingga harga yang diubah langsung tampil di halaman publik. Layanan baru mendapat kode bebas berikutnya: `MaxAsync(s => (string?)s.Id)` mencari kode terbesar (`S0012`), dan `$"S{… + 1:D4}"` memformat angka berikutnya dengan empat digit (`S0013`). *Cast* ke `string?` membuat `MaxAsync` mengembalikan `null`, bukan melempar *exception*, ketika tabelnya kosong.

Kode 5.20 `Controllers/PricesController.cs`: daftar harga pemilik, dengan kode layanan otomatis.

```
using LaundryMvc.Data;
using LaundryMvc.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace LaundryMvc.Controllers;

// The owner's price list: the same Service rows that /services shows.
public class PricesController(LaundryDbContext db) : Controller
{
    public async Task<IActionResult> Index() =>
        View(await db.Services.AsNoTracking().OrderBy(s => s.Id)
                              .ToListAsync());

    public IActionResult Create() => View(new Service());

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Create(
        [Bind("Description,Price,IsActive")] Service service)
    {
        if (!ModelState.IsValid) return View(service);

        // The next free code: S0012 -> S0013.
        var last = await db.Services.MaxAsync(s => (string?)s.Id) ?? "S0000";
        service.Id = $"S{int.Parse(last[1..]) + 1:D4}";
        db.Services.Add(service);
        await db.SaveChangesAsync();
        TempData["Message"] = $"{service.Id} {service.Name} added.";
        return RedirectToAction(nameof(Index));
    }

    public async Task<IActionResult> Edit(string id)
    {
        var service = await db.Services.FindAsync(id);
        return service is null ? NotFound() : View(service);
    }
```

Kode 5.20 (Lanjutan) *Controllers/PricesController.cs: daftar harga pemilik, dengan kode layanan otomatis.*

```
    // A new price counts for new orders only: every order keeps its Total.
    // A service is never deleted; it is switched off with IsActive = false.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(string id,
        [Bind("Id,Description,Price,IsActive")] Service input)
    {
        if (id != input.Id) return BadRequest();
        var service = await db.Services.FindAsync(id);
        if (service is null) return NotFound();
        if (!ModelState.IsValid) return View(input);

        service.Description = input.Description;
        service.Price = input.Price;
        service.IsActive = input.IsActive;
        await db.SaveChangesAsync();
        TempData["Message"] = $"{service.Id} {service.Name} saved.";
        return RedirectToAction(nameof(Index));
    }
}
```

Sengaja tidak ada *action* hapus. Pesanan lama menunjuk ke layanannya; menghapus layanan akan ditolak oleh kunci asing atau, lebih buruk lagi, menghilangkan riwayat. Mematikan `IsActive`

menyembunyikan layanan dari meja kasir dan dari daftar publik, sementara setiap pesanan lama tetap tahu layanan apa yang dipakainya.

Terakhir, `CounterController` (**Kode 5.21**) menjawab pertanyaan pemilik di malam hari dengan empat angka: pesanan yang antre, pesanan yang siap diambil, pesanan yang diterima hari ini, dan uang yang diterima hari ini. Setiap angka adalah satu perintah SQL kecil: `CountAsync` menjadi `SELECT COUNT(*)` dan `SumAsync` menjadi `SELECT COALESCE(SUM("p"."Amount"), 0)`, yang berhasil karena jumlah uang disimpan dalam kolom `int` (Subbab 5.3). "Hari ini" adalah rentang dari tengah malam sampai tengah malam berikutnya, `>= today && < tomorrow`, yang dapat dibandingkan basis data langsung pada cap waktu yang tersimpan.

Kode 5.21 `Controllers/CounterController.cs`: empat angka untuk meja kasir, masing-masing satu perintah SQL.

```
using LaundryMvc.Data;
using LaundryMvc.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace LaundryMvc.Controllers;

// GET /Counter: what the cashier needs to know at a glance.
public class CounterController(LaundryDbContext db) : Controller
{
    public async Task<IActionResult> Index()
    {
        var today = DateTime.Today;
        var tomorrow = today.AddDays(1);
        var model = new CounterViewModel(
            Queued: await db.Orders
                .CountAsync(o => o.Status == OrderStatus.Queued),
            ReadyForPickup: await db.Orders
                .CountAsync(o => o.Status == OrderStatus.Finished),
            ReceivedToday: await db.Orders
                .CountAsync(o => o.ReceivedAt >= today && o.ReceivedAt < tomorrow),
            PaidToday: await db.Payments
                .Where(p => p.PaidAt >= today && p.PaidAt < tomorrow)
                .SumAsync(p => p.Amount));
        return View(model);
    }
}
```

5.12 Menjalankannya

Di komputer Anda, dengan akses internet, jalankan perintah-perintah Subbab 5.9 lalu `dotnet run`, dan buka `http://localhost:4050`. Halaman beranda, layanan, dan gerai tampak persis seperti di Bab 3, kini dibaca dari basis data; kepala halaman memiliki baris tautan kedua untuk halaman staf. Tur pertama yang baik:

1. **Counter** menampilkan data contoh: dua pesanan antre, satu siap diambil, dua diterima hari ini, dan uang yang diterima hari ini.
2. **Customers** mendaftar Budi Santoso, Dewi Lestari, dan Sari Wulandari. Cari `Sari`, lalu `sari`, dan perhatikan perbedaan yang dijelaskan di Subbab 5.10.2.
3. Tambahkan seorang pelanggan, lalu coba tambahkan pelanggan kedua dengan nomor telepon yang sama; formulir kembali dengan pesan dari `CheckPhoneIsFreeAsync`.
4. **Orders → New order**: pilih Dewi Lestari dan *Wash, dry & fold*, isi 2 kg, lalu buat. Pesannya menyebut pesanan itu berharga Rp 24.000 dan menjelaskan bahwa 2 kg ditagih sebagai 3 kg.

5. Di halaman pesanan, catat pembayaran Rp 10.000, lalu tekan **Mark finished** dan **Mark picked up**, dan lihat cap waktunya muncul.
6. **Prices**: ubah harga *Ironing only* lalu buka `/services` di tab lain; harga barunya ada di sana. Pesanan lama mempertahankan totalnya.
7. Coba hapus Budi Santoso; halamannya menjelaskan bahwa pelanggan yang memiliki pesanan tidak dapat dihapus.

Perhatikan terminal sambil mengeklik. Setiap permintaan mencetak baris pewaktuan dari *middleware* Bab 3 dan, sebelumnya, SQL yang dikirim EF Core, dengan parameternya ditampilkan menurut nama, seperti `@search` atau `@p0` (nilainya disembunyikan kecuali Anda menyalakan *sensitive-data logging*, yang sebaiknya hanya dilakukan di komputer Anda sendiri). Membaca log itu adalah cara tercepat mempelajari menjadi apa LINQ Anda.

Proyek ini ditulis di tempat yang tidak dapat menjangkau NuGet, sehingga di sana proyek itu tidak dapat di-*restore* maupun dijalankan. Apa yang dilakukan di sana dinyatakan di README bab ini: `LinqDemo` dibangun dan dijalankan (keluarannya di Subbab 5.4 adalah keluaran asli), dan kode C# serta *view* Razor aplikasi web dikompilasi terhadap tiruan kecil API EF Core yang khusus untuk kompilasi, sebuah pemeriksaan tipe menyeluruh yang tidak menemukan kesalahan. Langkah-langkah di atas adalah yang akan Anda lihat di komputer biasa; satu-satunya angka yang disebut di dalamnya berasal dari data contoh dan aturan harga.

5.13 Membuatnya Lebih Baik

- **Baca tanpa pelacakan, tulis dengan pelacakan.** Setiap halaman daftar dan rincian memakai `AsNoTracking()`. Hanya *action* yang mengubah data yang memuat entitas terlacak.
- **Tetap asinkron.** *Controller* baru memakai `ToListAsync`, `FirstOrDefaultAsync`, `AnyAsync`, dan `SaveChangesAsync`. Selama SQLite membaca disk, *thread* server melayani permintaan lain. Di meja kasir dengan tiga kasir hal ini hampir tidak berpengaruh; di situs dengan ribuan pengunjung, hal ini menentukan berapa banyak permintaan yang dapat ditangani satu server.
- **Beri indeks pada yang dicari.** Indeks unik pada `Phone` sekaligus mempercepat pencarian telepon, dan indeks pada `Orders.Status` membantu daftar pesanan yang disaring. Tambahkan indeks ketika sebuah *query* nyata lambat, bukan jauh-jauh hari.
- **Awasi jumlah *query*.** Daftar pesanan memakai tiga `Include` dan memuat setiap pembayaran dari setiap pesanan, hanya untuk menampilkan satu sisa tagihan. Dengan 20 pesanan itu masih satu *query*, tetapi yang dibaca lebih banyak daripada yang diperlukan halaman, dan dengan ribuan pesanan daftarnya sendiri menjadi terlalu panjang. Bab 8 (Soal 8.3 dan 8.4) menggantinya dengan proyeksi dan halaman.
- **Perhatikan pemisah desimal.** *Model binding* membaca nilai formulir dengan *culture* server saat itu. Di komputer Windows yang diatur ke bahasa Indonesia, `4.5` dari isian angka peramban dapat menjadi 45, karena koma adalah pemisah desimal di `id-ID`. Biarkan *culture* server pada bawaannya, atau atur `CultureInfo.DefaultThreadCurrentCulture` secara eksplisit, dan ujilah dengan nilai yang memiliki desimal.

5.14 Memakainya Sungguhan

- **Toko Anda sendiri.** Ganti `data/Service.csv` dan `data/Outlet.csv` dengan layanan dan cabang Anda sebelum `database update` pertama, atau ubah kemudian di halaman Prices. Penjahit akan memiliki layanan seperti *Hemming (per item)*; tempat cuci kendaraan *Motorbike wash (per item)*. Aturan per kg pada kalkulator Bab 3 hanya berlaku untuk deskripsi yang berakhiran `(per kg)`.

- **Basis data lain.** Untuk SQL Server, ganti paketnya dengan `Microsoft.EntityFrameworkCore.SqlServer`, ubah `UseSqlite` menjadi `UseSqlServer`, ubah untai koneksinya, hapus folder `Migrations` dan buat `InitialCreate` lagi, karena migrasi ditulis untuk satu penyedia. Untuk MySQL atau MariaDB, penyedia komunitas `Pomelo.EntityFrameworkCore.MySql` adalah pilihan yang lazim; periksa di NuGet bahwa versinya mendukung EF Core 10 sebelum Anda beralih.
- **Cadangkan berkasnya.** Dengan SQLite seluruh toko adalah satu berkas, `laundry.db`. Salin setiap malam ke disk lain atau penyimpanan awan; salinan yang diambil saat aplikasi sedang tidak dipakai adalah cadangan yang lengkap. Bab 6 meletakkan berkas itu di folder tetap di server.
- **Data pribadi.** Nama, nomor telepon, dan alamat adalah data pribadi menurut Undang-Undang Pelindungan Data Pribadi (UU PDP, UU No. 27 Tahun 2022). Kumpulkan hanya yang diperlukan toko (model ini tidak meminta nomor identitas maupun tanggal lahir), simpan berkas basis data di komputer yang terkunci, dan hapus pelanggan yang memintanya dan tidak memiliki pesanan terbuka; halaman hapus sudah mendukung hal itu.

5.15 Profitina dalam Praktik

Platform Profitina sendiri ditulis dengan Python dan FastAPI, bukan C#, tetapi dibangun dari bagian-bagian yang sama dengan bab ini. Datanya berada di PostgreSQL di produksi, sementara sebagian jalur pengujiannya berjalan di SQLite, persis pembagian antara basis data server dan basis data berkas yang dijelaskan di Subbab 5.2.4 dan 5.14; sebuah penjaga di rangkaian ujinya memeriksa bahwa SQL yang dipakai tetap sah di kedua dialek, masalah yang dipecahkan oleh penyedia-penyedia EF Core bagi pengembang .NET. Riwayat skor mingguan per merek adalah tabel yang terus bertambah dan di-*query* dasbor dengan agregat, jenis `GROUP BY` dan `SUM` yang sama dengan halaman kasir dan demo LINQ. Dan, seperti Profitina Laundry, setiap perubahan struktur basis datanya melalui migrasi berversi yang diterapkan proses rilis sebelum versi baru diizinkan menjawab permintaan.

5.16 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Lupa `SaveChangesAsync()`.** `Add`, `Remove`, dan mengubah properti hanya mengubah pelacak perubahan. Tidak ada yang sampai ke basis data sampai `SaveChanges` atau `SaveChangesAsync` dipanggil.
- **Mengubah model tanpa migrasi.** Setelah Anda menambah properti, basis data belum memiliki kolomnya. Sejak EF Core 9, `Migrate()` dan `database update` berhenti dengan galat *"The model … has pending changes"*; jalankan `dotnet ef migrations add <Nama>` terlebih dahulu.
- **Mengira properti navigasi terisi sendiri.** `order.Customer` bernilai `null` dan `order.Payments` kosong kecuali *query*-nya menyebut `Include`. Sisa tagihan yang dihitung dari *list* kosong salah begitu saja, tanpa galat apa pun.
- **Melacak segalanya.** Membaca seribu baris tanpa `AsNoTracking()` untuk halaman daftar memboroskan memori. Sebaliknya, memanggil `SaveChanges` setelah *query* `AsNoTracking` tidak menyimpan apa-apa, karena tidak ada yang dilacak.
- **Mendaftarkan DbContext sebagai *singleton*.** Satu DbContext untuk seluruh aplikasi akan rusak begitu dua permintaan datang bersamaan. `AddDbContext` sudah mendaftarkannya dengan benar sebagai *scoped*; biarkan begitu, dan jadikan layanan yang memakainya *scoped* juga.

- **Hapus berantai yang tidak disengaja.** Bawaan EF Core untuk relasi wajib adalah hapus berantai. Putuskan dengan sadar, untuk setiap relasi, apakah menghapus induk boleh menghapus anak-anaknya; untuk pelanggan dan pesanan, tidak boleh.
- **Menghapus lewat tautan.** Permintaan `GET` tidak boleh mengubah data. Pakailah formulir dengan `method="post"` dan token *anti-forgery* untuk setiap *action* yang menulis.
- **Menjumlahkan uang bertipe `decimal` di SQLite.** EF Core tidak dapat menerjemahkan `Sum` atas kolom `decimal` untuk SQLite dan melempar *exception* saat berjalan. Simpan jumlah yang akan dijumlahkan dalam kolom bilangan bulat, atau konversikan.

5.17 Ringkasan Bab

- ORM memetakan kelas ke tabel dan objek ke baris. EF Core 10 adalah ORM untuk .NET 10; penyedia SQLite dan alat desainnya adalah paket NuGet, dan `dotnet ef` adalah alat global.
- Model adalah kelas C# biasa. Konvensi menemukan kunci, kunci asing, dan relasi; *fluent API* di `OnModelCreating` menambahkan panjang, indeks unik, konversi nilai, dan perilaku hapus.
- `DbContext` didaftarkan sekali dengan `AddDbContext` dan disuntikkan, satu per permintaan, ke *controller* dan layanan.
- Migrasi mengubah perubahan model menjadi berkas C# yang berurutan dan dapat dibatalkan; `database update` menerapkannya dan mencatatnya di `__EFMigrationsHistory`. `UseSeeding` mengisi basis data kosong, di sini dari berkas CSV Bab 3.
- LINQ pada `DbSet` diterjemahkan menjadi SQL berparameter; tidak ada yang dikirim sampai metode seperti `ToListAsync` meminta hasilnya. `Include` memuat data terkait; `AsNoTracking` melewati pelacak perubahan untuk pembacaan.
- *Create* menambah lalu menyimpan; *update* mengubah entitas terlacak lalu menyimpan, dan EF Core hanya menulis kolom yang berubah; *delete* menghapus lalu menyimpan, dijaga oleh pemeriksaan dan oleh `Restrict` pada relasinya.
- Katalog Bab 3 kini membaca basis data melalui `EfCatalog`, dan tidak satu pun *controller* Bab 3 berubah.

5.18 Soal Latihan

1. Dengan tangan, terapkan Persamaan (5.1) pada pesanan 2 dari data contoh setelah dua pembayaran lagi sebesar Rp 10.000 dan Rp 6.000. Berapa `Paid` dan `Balance`, dan apa yang ditampilkan halaman pesanan sebagai pengganti formulir pembayaran?
2. Untuk masing-masing berikut, sebutkan perintah SQL yang dikirim EF Core dan kapan: (a) `var q = db.Orders.Where(o => o.Status == OrderStatus.Queued);` (b) `var n = await q.CountAsync();` (c) `var list = await q.ToListAsync();` (d) `list[0].Note = "fragile"; await db.SaveChangesAsync();` di *controller* yang *query*-nya *tidak* memakai `AsNoTracking`.
3. Tulis *query* LINQ untuk halaman kasir yang mengembalikan tiga pelanggan dengan belanja terbesar (jumlah total pesanan mereka), berisi nama dan jumlahnya, dari yang terbesar. Ujilah dengan menambahkannya ke `LinqDemo`.
4. Pemilik ingin tahu di gerai mana setiap pesanan diterima. Tambahkan relasinya ke model: properti apa yang masuk ke `Order`, apa yang masuk ke `OnModelCreating`, perintah migrasi apa yang menyusul, dan apa yang berubah di formulir pesanan?
5. Seorang kasir menghapus pembayaran yang tercatat dua kali. Tambahkan *action* `DeletePayment(int id, int paymentId)` ke `OrdersController`, yang hanya dapat dicapai lewat `POST`, yang menghapus pembayaran itu bila memang milik pesanan tersebut.

6. Jelaskan dengan kata-kata Anda sendiri mengapa `EfCatalog` didaftarkan sebagai *scoped* sedangkan `PriceCalculator` tetap *singleton*. Apa yang terjadi bila keduanya *singleton*?

5.19 Penyelesaian Soal Latihan

1. Pembayarannya Rp 20.000, Rp 10.000, dan Rp 6.000, sehingga `Paid` $= 20000 + 10000 + 6000 = 36000$ dan, menurut Persamaan (5.1), `Balance` $= 36000 - 36000 = 0$. *View* rincian hanya menampilkan formulir pembayaran `@if (Model.Balance > 0)`, jadi formulir itu hilang dan halaman menampilkan sisa tagihan Rp 0.

2. (a) Tidak ada: `q` hanyalah gambaran sebuah *query*. (b) `SELECT COUNT(*) FROM "Orders" AS "o" WHERE "o"."Status" = 'Queued'`, dikirim ketika `CountAsync` berjalan; enum dibandingkan sebagai teks karena `HasConversion<string>()`. (c) `SELECT "o"."Id", "o"."CustomerId", … FROM "Orders" AS "o" WHERE "o"."Status" = 'Queued'`, dikirim ketika `ToListAsync` berjalan; pesanan-pesanan itu dilacak. (d) `UPDATE "Orders" SET "Note" = @p0 WHERE "Id" = @p1`, dikirim oleh `SaveChangesAsync` hanya dengan kolom yang berubah (penyedia SQLite menambahkan `RETURNING 1` untuk memastikan satu baris memang berubah).

3. Kelompokkan pesanan menurut pelanggan, jumlahkan, urutkan menurun, dan ambil tiga. Di basis data, *query* yang sama bekerja pada `db.Orders` dengan `await … .ToListAsync()` di akhir; `Total` bertipe `int`, jadi SQLite dapat menjumlahkannya. **Kode 5.22** memperlihatkan baris-baris untuk demonya.

Kode 5.22 Soal 5.3: tiga pelanggan terbaik, sebagai baris-baris yang ditambahkan ke `LinqDemo/Program.cs`.

```
var top = orders
    .GroupBy(o => o.Customer!.Name)
    .Select(g => new { Customer = g.Key, Spent = g.Sum(o => o.Total) })
    .OrderByDescending(r => r.Spent)
    .Take(3);
foreach (var r in top)
{
    Console.WriteLine($"   {r.Customer,-16} {Money.Rp(r.Spent),10}");
}
```

Dengan data demo, program mencetak Budi Santoso dengan Rp 116.000, Dewi Lestari dengan Rp 48.000, dan Sari Wulandari dengan Rp 36.000.

4. Tambahkan `public string OutletId { get; set; } = "";` dan `public Outlet? Outlet { get; set; }` ke `Order`. Di `OnModelCreating`, di dalam blok `Order`, tambahkan `e.HasOne(o => o.Outlet).WithMany().HasForeignKey(o => o.OutletId).OnDelete(DeleteBehavior.Restrict);`. Lalu jalankan `dotnet ef migrations add AddOutletToOrders`. Karena baris yang sudah ada memerlukan nilai untuk kolom baru yang tidak *nullable*, jadikan kolom itu *nullable* (`string?`) untuk migrasi ini, atau beri nilai bawaan di migrasinya (misalnya `O02` untuk gerai Keputih tempat sistem mulai dipakai) dan sunting pemanggilan `AddColumn` yang dihasilkan. Di formulir pesanan, tambahkan `OutletId` ke `[Bind]`, isi `ViewBag.Outlets` di `FillListsAsync` dari `db.Outlets`, tambahkan `<select asp-for="OutletId" asp-items="ViewBag.Outlets">`, dan periksa di `Create` bahwa gerainya ada. Diagram ERD pada **Gambar 5.2** kemudian mendapat relasi keempat, dari `Outlets` ke `Orders`.

5. Cari pembayaran dengan kedua kunci, sehingga `paymentId` palsu dari pesanan lain ditolak, lalu hapus (**Kode 5.23**):

Kode 5.23 Soal 5.5: *action* yang ditambahkan ke `OrdersController` untuk menghapus satu pembayaran dari satu pesanan.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeletePayment(int id, int paymentId)
{
    var payment = await db.Payments
        .FirstOrDefaultAsync(p => p.Id == paymentId && p.OrderId == id);
    if (payment is null) return NotFound();
    db.Payments.Remove(payment);
    await db.SaveChangesAsync();
    TempData["Message"] = $"Payment of {Money.Rp(payment.Amount)} removed.";
    return RedirectToAction(nameof(Details), new { id });
}
```

Di tabel pembayaran pada *view* rincian, tambahkan ke setiap baris sebuah formulir kecil: `<form asp-action="DeletePayment" asp-route-id="@Model.Id" asp-route-paymentId="@p.Id" method="post"><button type="submit">Remove</button></form>`.

6. `EfCatalog` memegang sebuah `LaundryDbContext`, dan DbContext bersifat *scoped*: satu per permintaan, tidak aman dipakai bersama oleh permintaan yang berjalan bersamaan, dan dibuang di akhir permintaannya. `EfCatalog` yang *singleton* akan memegang DbContext milik permintaan pertama selamanya; ASP.NET Core menolak hal ini saat program dimulai dalam mode Development dengan pesan bahwa layanan *scoped* tidak dapat dipakai dari *singleton*. `PriceCalculator` tidak memegang data maupun DbContext; metodenya hanya menghitung dari argumennya, sehingga satu instans bersama aman dan paling hemat. Bila keduanya *singleton*, kalkulator tetap baik-baik saja dan katalognya rusak.

Daftar Pustaka

[1] Microsoft, “Entity Framework Core overview,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/ef/core/

[2] Microsoft, “.NET and .NET Core support policy,” diakses Sep. 2026. [Daring]. Tersedia: https://dotnet.microsoft.com/platform/support/policy/dotnet-core

[3] Microsoft, “SQLite EF Core database provider limitations,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/ef/core/providers/sqlite/limitations

[4] NuGet Gallery, “Microsoft.EntityFrameworkCore.Sqlite 10.0.12,” dirilis 8 Sep. 2026. [Daring]. Tersedia: https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Sqlite

[5] Microsoft, “Data seeding,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/ef/core/modeling/data-seeding

[6] Microsoft, “Migrations overview,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/ef/core/managing-schemas/migrations/

[7] Microsoft, “Loading related data,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/ef/core/querying/related-data/

[8] Microsoft, “Model binding in ASP.NET Core,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/aspnet/core/mvc/models/model-binding

[9] Microsoft, “Change tracking in EF Core,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/ef/core/change-tracking/

Bab 6

ASP.NET Core MVC III: Validasi, Identity, dan Penerapan

Di akhir Bab 5 laundry sudah memiliki basis data dan halaman CRUD sungguhan, dan siapa pun yang dapat menjangkau situs web itu dapat memakai semuanya. Bab ini membuat aplikasi layak dipercaya untuk toko sungguhan. Di akhir bab Anda akan mampu memvalidasi setiap formulir di server dengan atribut *data annotation*, atribut validasi buatan sendiri, dan aturan yang melibatkan beberapa isian; menerima formulir melalui kelas formulir khusus alih-alih entitas; menjelaskan perbedaan antara *authentication* (autentikasi) dan *authorization* (otorisasi); menambahkan ASP.NET Core Identity sehingga staf masuk dengan kata sandi yang di-*hash*, penguncian akun, dan peran; melindungi halaman staf sementara halaman publik Bab 3 tetap terbuka; serta menerbitkan aplikasi ke server di belakang Nginx atau IIS, dengan pengaturan produksi, *endpoint* kesehatan, kunci cookie yang bertahan saat dinyalakan ulang, dan berkas basis data yang dicadangkan. Sepanjang jalan Anda akan menjalankan aturan validasi milik aplikasi dan *password hasher* Identity dalam program konsol dan membaca keluaran aslinya.

6.1 Posisi Kita Sekarang: Pintu yang Terbuka

Minggu pertama bersama aplikasi Bab 5 berjalan lancar, lalu terjadi tiga hal kecil. Seorang kasir paruh waktu yang baru, ketika mencari harga setrika, membuka *Prices*, mengetik di isian yang salah, dan menyimpan *Ironing only* seharga Rp 60 per kg; tidak ada yang sadar sampai malam. Nomor telepon seorang pelanggan tercatat `0812 0000 104`, dengan huruf O menggantikan angka nol, sehingga pesan WhatsApp yang memberitahukan cuciannya sudah selesai tidak pernah sampai. Dan sebuah pesanan dengan antar-jemput tersimpan tanpa alamat antar; sopirnya berdiri di depan meja kasir menanyakan ke mana harus pergi.

Tidak satu pun dari ketiganya adalah kesalahan pemrograman dalam arti biasa. Kodenya melakukan persis apa yang diperintahkan. Yang tidak ada adalah seperangkat aturan yang harus dilalui setiap permintaan: harga layanan paling sedikit Rp 1.000; nomor telepon adalah nomor seluler Indonesia; pengantaran memerlukan alamat; hanya pemilik yang boleh mengubah harga. Dan satu hal lagi belum ada: aplikasi masih berjalan di laptop pemilik, dinyalakan dengan tangan lewat `dotnet run`, dan berhenti setiap kali laptop tertidur.

Bab ini menambahkan aturan-aturan itu dan memindahkan aplikasi ke server. Bab ini membantu kasir, yang mendapat pesan jelas alih-alih kesalahan yang diam-diam; pelanggan, yang nomornya diperiksa sebelum disimpan; sopir, yang selalu mendapat alamat; dan pemilik, yang sendirian menentukan harga dan tak perlu lagi khawatir apakah laptopnya menyala.

6.2 Gagasannya dengan Kata-Kata Sederhana

Tiga gagasan menopang bab ini.

Validasi berarti memeriksa bahwa data dalam sebuah permintaan masuk akal sebelum ditindaklanjuti. Peramban dapat membantu, dengan menolak mengirim isian wajib yang kosong,

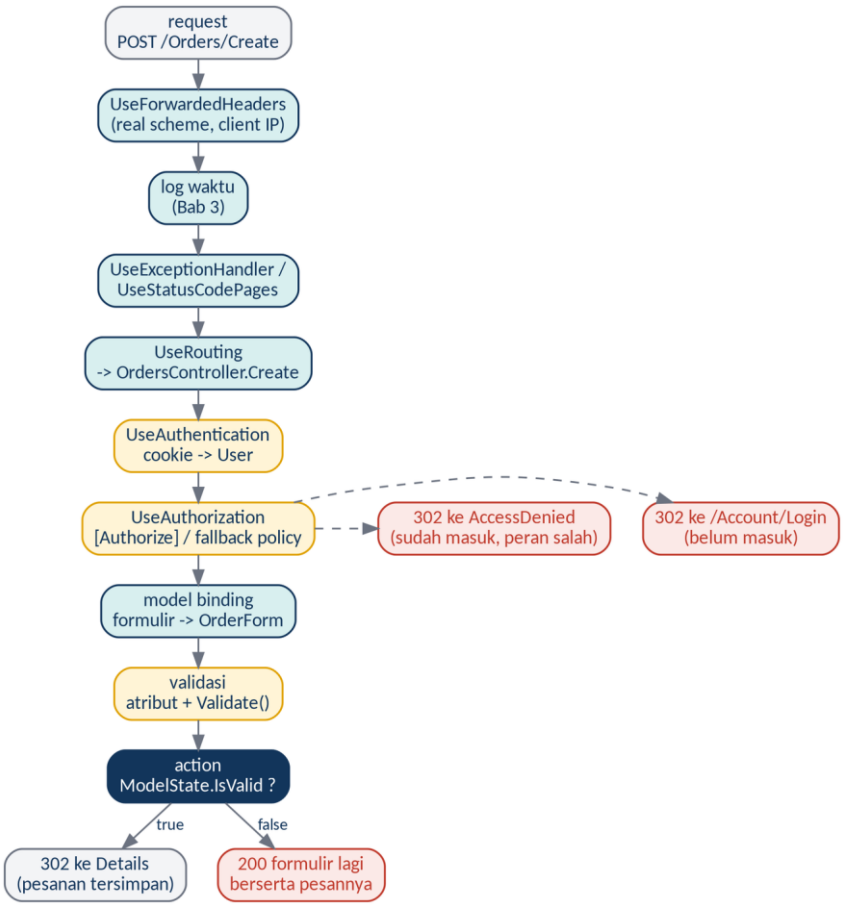
tetapi peramban milik pengguna, dan siapa pun dapat mematikan pemeriksaannya atau mengirim permintaan tanpa peramban sama sekali. Karena itu aturannya sederhana: setiap aturan diperiksa di server; pemeriksaan di peramban hanyalah kenyamanan yang menghemat satu kali bolak-balik.

Autentikasi dan otorisasi adalah dua pertanyaan yang berbeda. Autentikasi bertanya *siapa Anda?* dan dijawab dengan masuk (*sign in*): pengguna membuktikan identitasnya, di sini dengan alamat surel dan kata sandi, lalu menerima cookie yang menyatakannya pada setiap permintaan berikutnya. Otorisasi bertanya *apakah Anda boleh melakukan ini?* dan dijawab oleh aturan: kasir yang sudah masuk boleh mencatat pesanan tetapi tidak boleh mengubah harga. Permintaan yang gagal autentikasi dikirim ke halaman masuk; permintaan yang gagal otorisasi dari pengguna yang sudah masuk ditolak.

Penerapan (*deployment*) berarti menjalankan aplikasi di tempat ia dapat melayani toko setiap hari: komputer yang selalu menyala, server web di depannya yang menangani HTTPS, pengaturan produksi yang berbeda dari pengaturan di laptop pengembang, serta kebiasaan rutin untuk pembaruan dan pencadangan.

> **DEFINISI.** *ASP.NET Core Identity* adalah sistem keanggotaan ASP.NET Core: menyimpan pengguna, meng-*hash* dan memverifikasi kata sandinya, mengunci akun setelah percobaan gagal berulang, mengelola peran, serta memasukkan dan mengeluarkan pengguna dengan cookie autentikasi. *Role* (peran) adalah kelompok pengguna bernama, seperti *Owner* atau *Cashier*, yang dapat dirujuk aturan otorisasi. *Policy* (kebijakan) adalah aturan otorisasi bernama; *fallback policy* (kebijakan cadangan) adalah aturan yang diterapkan pada setiap *endpoint* yang tidak menyatakan aturannya sendiri.

Gambar 6.1 memperlihatkan letak setiap pemeriksaan itu ketika kasir mengirim formulir pesanan baru. Permintaan melewati *middleware* Bab 3, diarahkan ke `OrdersController.Create`, cookie autentikasinya dibaca, dan diperiksa terhadap aturan otorisasi; baru setelah itu *model binding* membangun objek formulir dan validasi memeriksanya, dan baru setelah itu *action* berjalan dan memutuskan, dari `ModelState.IsValid`, apakah menyimpan atau menampilkan formulir lagi.



Gambar 6.1 Jalur pengiriman formulir melalui aplikasi Bab 6: autentikasi dan otorisasi sebelum *controller*, *model binding* dan validasi sebelum *action*, serta empat cara permintaan dapat berakhir.

6.3 Validasi dengan *Data Annotation*

6.3.1 Atribut pada model

Cara paling sederhana menyatakan aturan di ASP.NET Core adalah memasang *data annotation attribute* pada sebuah properti. **Kode 6.1** adalah kelas `Customer` dari Bab 5 yang sudah ditambahi aturan.

Kode 6.1 `Models/Customer.cs` dengan atribut validasi: nama wajib sepanjang 2 sampai 80 karakter dan nomor seluler Indonesia yang wajib.

```
using System.ComponentModel.DataAnnotations;

namespace LaundryMvc.Models;

public class Customer
{
    public int Id { get; set; }

    [Required(ErrorMessage = "Please enter the customer's name.")]
    [StringLength(80, MinimumLength = 2,
        ErrorMessage = "The name must be 2 to 80 characters long.")]
    public string Name { get; set; } = "";

    [Required(ErrorMessage = "Please enter a phone number.")]
    [IndonesianMobile]
    [Display(Name = "Phone (WhatsApp)")]
    public string Phone { get; set; } = "";

    [StringLength(200)]
    public string? Address { get; set; }

    public DateTime CreatedAt { get; set; } = DateTime.Now;

    public List<Order> Orders { get; set; } = new();
}
```

`[Required]` menolak nilai yang tidak ada atau kosong; `[StringLength(80, MinimumLength = 2)]` membatasi panjang; `[Display(Name = "Phone (WhatsApp)")]` memberi label yang dicetak `<label asp-for="Phone">` dan dipakai pesan galat sebagai pengganti `{0}`. Setiap atribut menerima `ErrorMessage` yang ditulis dengan bahasa toko, dan itu selalu lebih baik daripada teks bawaan. **Tabel 6.1** mendaftar atribut yang dipakai di bab ini.

Tabel 6.1 Atribut validasi yang dipakai di bab ini.

Atribut	Memeriksa	Dipakai pada
`[Required]`	nilai ada dan tidak kosong	nama dan telepon pelanggan, deskripsi layanan
`[StringLength(max, MinimumLength = min)]`	panjang teks	nama, alamat, catatan
`[Range(min, max)]`	angka berada di antara dua batas	harga, jumlah, besar pembayaran
`[RegularExpression(pola)]`	teks cocok dengan pola	deskripsi layanan berakhiran `(per kg)`
`[EmailAddress]`	teks tampak seperti alamat surel	formulir masuk
`[EnumDataType(typeof(T))]`	nilai adalah salah satu nilai enum	cara bayar
`[IndonesianMobile]` (buatan kita)	nomor seluler Indonesia	telepon pelanggan

Layanan mendapat aturan untuk deskripsi dan harganya (**Kode 6.2**). Ekspresi reguler `^.{3,}\(per (kg|item|pair)\)$` menerima deskripsi minimal tiga karakter yang diikuti spasi dan tepat salah satu dari `(per kg)`, `(per item)`, atau `(per pair)`; karena aturan harga Bab 3 membaca satuan dari akhiran itu, deskripsi tanpa akhiran tersebut diam-diam akan mengubah layanan per kg menjadi per potong. `[Range(1000, 1_000_000)]` akan menghentikan harga setrika Rp 60 di Subbab 6.1.

Kode 6.2 `Models/Service.cs` dengan aturan untuk deskripsi dan harga.

```
using System.ComponentModel.DataAnnotations;

namespace LaundryMvc.Models;

// Chapter 3 read services from a CSV file as read-only records. From
// Chapter 5 on they are rows the owner edits, so the properties have setters.
public class Service
{
    public string Id { get; set; } = "";           // S0001, S0002, ...

    [Required, StringLength(60)]
    [RegularExpression(@"^.{3,} \(per (kg|item|pair)\)$", ErrorMessage =
        "End the description with (per kg), (per item) or (per pair).")]
    public string Description { get; set; } = "";  // "Ironing only (per kg)"

    [Range(1000, 1_000_000,
        ErrorMessage = "The price must be between Rp 1.000 and Rp 1.000.000.")]
    [Display(Name = "Price (rupiah)")]
    public decimal Price { get; set; }             // rupiah per kg/item/pair

    public bool IsActive { get; set; } = true;

    // Worked out from Description, as in Chapter 3; not stored.
    public string Name => Description.Contains(" (")
        ? Description[..Description.LastIndexOf(" (")] : Description;

    public bool PerKg => Description.EndsWith("(per kg)");

    public string Unit => PerKg ? "kg"
        : Description.EndsWith("(per pair)") ? "pair" : "item";
}
```

6.3.2 Atribut validasi buatan sendiri

Tidak ada atribut bawaan yang tahu rupa nomor seluler Indonesia. `[Phone]` memang ada, tetapi hampir semua yang berisi angka diterimanya. Menulis atribut sendiri hanya memerlukan kelas turunan `ValidationAttribute` dengan satu metode, `IsValid` (**Kode 6.3**).

Kode 6.3 `Models/IndonesianMobileAttribute.cs`: atribut validasi untuk nomor seluler Indonesia, dengan pembantu yang menyimpan setiap nomor dalam satu ejaan.

```
using System.ComponentModel.DataAnnotations;

namespace LaundryMvc.Models;

// Accepts 0812-0000-0101, 0812 0000 0101, +62 812-0000-0101, 6281200000101.
public sealed class IndonesianMobileAttribute : ValidationAttribute
{
    public IndonesianMobileAttribute()
        : base("{0} must be a mobile number such as 0812-3456-7890.")
    {
    }

    public override bool IsValid(object? value)
    {
        if (value is not string text || text.Length == 0)
        {
            return true;                  // an empty value is [Required]'s job
        }
        var digits = Normalize(text);
        return digits.Length is >= 10 and <= 13
            && digits.StartsWith("08")
            && digits.All(char.IsAsciiDigit);
    }

    // "+62 812-0000-0101" -> "081200000101": one spelling in the database.
    public static string Normalize(string text)
    {
        var s = new string(text.Where(ch => !char.IsWhiteSpace(ch) && ch != '-')
                               .ToArray());
        if (s.StartsWith("+62")) return "0" + s[3..];
        if (s.StartsWith("62")) return "0" + s[2..];
        return s;
    }
}
```

Orang menulis nomor yang sama dengan banyak cara: `0812-0000-0101`, `0812 0000 0101`, `+62 812-0000-0101`. `Normalize` membuang spasi dan tanda hubung serta mengubah awalan `+62` atau `62` menjadi `0`; `IsValid` lalu menerima hasilnya bila terdiri atas 10 sampai 13 digit dan diawali `08`. Atribut ini sengaja mengembalikan `true` untuk nilai kosong: wajib atau tidaknya sebuah nilai adalah urusan `[Required]`, dan memisahkan keduanya memungkinkan `[IndonesianMobile]` dipakai lagi pada isian yang tidak wajib. *Controller* memanggil `Normalize` sebelum menyimpan, sehingga basis data memegang satu ejaan per nomor dan pemeriksaan keunikan Bab 5 tidak dapat dikelabui dengan sebuah spasi.

6.3.3 Menjalankan aturan tanpa server web

MVC menjalankan aturan-aturan ini selama *model binding*, tetapi semuanya .NET biasa: kelas `Validator` di `System.ComponentModel.DataAnnotations` dapat menjalankannya di mana saja. `ValidationDemo` (**Kode 6.4**) mengompilasi kelas-kelas model aplikasi web, seperti `LinqDemo` di Bab 5, memvalidasi tujuh objek, lalu melihat apa yang disimpan Identity sebagai pengganti kata sandi (Subbab 6.6.3).

Kode 6.4 `ValidationDemo/Program.cs`: aturan validasi milik aplikasi dan *password hasher* Identity, dijalankan dalam konsol.

```
using System.ComponentModel.DataAnnotations;
using LaundryMvc.Models;
using Microsoft.AspNetCore.Identity;

// Runs the same checks that MVC model validation runs, and prints them.
static void Check(string title, object model)
{
    var errors = new List<ValidationResult>();
    var ok = Validator.TryValidateObject(model, new ValidationContext(model),
                                         errors, validateAllProperties: true);
    Console.WriteLine($"{title}: {(ok ? "valid" : "INVALID")}");
    foreach (var e in errors)
    {
        Console.WriteLine($"   [{string.Join(",", e.MemberNames)}] "
                          + e.ErrorMessage);
    }
}

Check("1. Customer 'A', phone 12345",
      new Customer { Name = "A", Phone = "12345" });
Check("2. Customer 'Rina Kartika', phone +62 812-0000-0104",
      new Customer { Name = "Rina Kartika", Phone = "+62 812-0000-0104" });
Console.WriteLine("   stored as "
                  + IndonesianMobileAttribute.Normalize("+62 812-0000-0104"));
Check("3. Service 'Carpet' at Rp 500, no unit",
      new Service { Description = "Carpet", Price = 500 });
Check("4. Order: no customer, no service, 0 kg",
      new OrderForm { Quantity = 0 });
Check("5. Order: pickup without an address",
      new OrderForm { CustomerId = 2, ServiceId = "S0001", Quantity = 4,
                      Pickup = true });
Check("6. Order: pickup to Jl. Mulyosari 30",
      new OrderForm { CustomerId = 2, ServiceId = "S0001", Quantity = 4,
                      Pickup = true, DeliveryAddress = "Jl. Mulyosari 30" });
Check("7. Payment of Rp 500",
      new PaymentForm { Amount = 500, Method = PaymentMethod.Cash });
```

Kode 6.4 (Lanjutan) *ValidationDemo/Program.cs: aturan validasi milik aplikasi dan password hasher Identity, dijalankan dalam konsol.*

```
// What Identity stores instead of a password.
var hasher = new PasswordHasher<IdentityUser>();
var user = new IdentityUser { UserName = "kasir1@profitina-laundry.example" };
var hash = hasher.HashPassword(user, "Kasir#2026");
var bytes = Convert.FromBase64String(hash);
int ReadInt(int at) => (bytes[at] << 24) | (bytes[at + 1] << 16)
                     | (bytes[at + 2] << 8) | bytes[at + 3];
var saltLength = ReadInt(9);
Console.WriteLine("8. PasswordHasher<IdentityUser>");
Console.WriteLine($"   stored text      {hash.Length} characters, "
                  + $"starts {hash[..12]}...");
Console.WriteLine($"   format marker    0x{bytes[0]:X2}");
Console.WriteLine($"   PRF              {(KeyDerivationPrf)ReadInt(1)}");
Console.WriteLine($"   iterations       {ReadInt(5):N0}");
Console.WriteLine($"   salt, key bytes  {saltLength}, "
                  + $"{bytes.Length - 13 - saltLength}");
Console.WriteLine("   right password   "
    + hasher.VerifyHashedPassword(user, hash, "Kasir#2026"));
Console.WriteLine("   wrong password   "
    + hasher.VerifyHashedPassword(user, hash, "kasir#2026"));
Console.WriteLine("   same password, second hash equal? "
    + (hasher.HashPassword(user, "Kasir#2026") == hash));

enum KeyDerivationPrf { HMACSHA1, HMACSHA256, HMACSHA512 }
```

Inilah keluaran asli `dotnet run`:

```
1. Customer 'A', phone 12345: INVALID
   [Name] The name must be 2 to 80 characters long.
   [Phone] Phone (WhatsApp) must be a mobile number such as 0812-3456-7890.
2. Customer 'Rina Kartika', phone +62 812-0000-0104: valid
   stored as 081200000104
3. Service 'Carpet' at Rp 500, no unit: INVALID
   [Description] End the description with (per kg), (per item) or (per pair).
   [Price] The price must be between Rp 1.000 and Rp 1.000.000.
4. Order: no customer, no service, 0 kg: INVALID
   [CustomerId] Choose a customer.
   [ServiceId] Choose a service.
   [Quantity] The quantity must be between 0.1 and 100.
5. Order: pickup without an address: INVALID
   [DeliveryAddress] Pickup and delivery needs a delivery address.
6. Order: pickup to Jl. Mulyosari 30: valid
7. Payment of Rp 500: INVALID
   [Amount] A payment must be between Rp 1.000 and Rp 10.000.000.
8. PasswordHasher<IdentityUser>
   stored text      84 characters, starts AQAAAAIAAYag...
   format marker    0x01
   PRF              HMACSHA512
   iterations       100,000
   salt, key bytes  16, 32
   right password   Success
   wrong password   Failed
   same password, second hash equal? False
```

Kasus 1 gagal dua kali, dengan pesan yang ditulis di atributnya, dan nama `[Display]` muncul di pesan telepon. Kasus 2 lolos dan memperlihatkan bahwa `+62 812-0000-0104` disimpan sebagai `081200000104`. Kasus 3 adalah jenis kecelakaan daftar harga pada Subbab 6.1: tidak ada satuan di deskripsi dan harganya di bawah minimum. Kasus 4 sampai 7 menyangkut kelas-kelas formulir di subbab berikutnya. Perhatikan pada kasus 4 bahwa pelanggan dan layanan yang kosong tertangkap oleh `[Range]` dan `[Required]`, bukan oleh pencarian di basis data: aturan yang tidak memerlukan basis data berjalan sebelum *controller* menyentuh basis data.

6.3.4 Apa yang dilakukan MVC dengan aturan-aturan itu

Selama *model binding*, MVC menjalankan pemeriksaan yang sama dan menyimpan setiap kegagalan di `ModelState` di bawah nama propertinya. *Action* menanyakan `ModelState.IsValid`; *view* menampilkan setiap pesan dengan `<span asp-validation-for="Phone">` di samping isiannya dan pesan yang bukan milik satu isian dengan `<div asp-validation-summary="ModelOnly">`. Itulah sebabnya *view* Razor di Bab 5 sudah memiliki tag-tag ini: tag itu sedang menunggu aturan.

Aturan yang memerlukan basis data, seperti “tidak ada pelanggan lain yang memakai nomor ini” atau “layanan itu masih ada dan masih ditawarkan”, tidak dapat dijadikan atribut pada kelas model tanpa menyeret DbContext ke dalam model. Aturan semacam itu tetap di *controller*, yang menambahkan pesannya sendiri ke `ModelState` dengan `AddModelError`, seperti `CheckPhoneIsFreeAsync` di Bab 5. Dari sudut pandang *view*, tidak ada bedanya.

6.4 Kelas Formulir Alih-alih Entitas

6.4.1 Mengapa kelas tersendiri

Di Bab 5 formulir pesanan baru diikat langsung ke entitas `Order`, dan `[Bind("CustomerId,ServiceId,Quantity,Pickup,DeliveryAddress,Note")]` mendaftar properti yang boleh diisi permintaan. Cara itu bekerja, tetapi memiliki dua kelemahan. Daftar nama itu berupa untai teks yang tidak dapat diperiksa kompilator, sehingga properti yang berganti nama diam-diam terlepas dari daftar. Dan entitas adalah tempat yang keliru untuk aturan milik satu formulir: "pengantaran memerlukan alamat" adalah aturan formulir kasir, bukan aturan setiap pesanan yang akan pernah ada.

Form class (kelas formulir, sering disebut *view model* atau *input model*) memecahkan keduanya. Kelas ini memiliki persis properti yang boleh dikirim formulir, sehingga *overposting* mustahil sejak rancangannya: tidak ada `Total`, `Status`, atau `ReceivedAt` yang dapat ditimpa. Dan kelas ini membawa aturan formulirnya. **Kode 6.5** adalah kelas formulir untuk pesanan baru.

Kode 6.5 `Models/OrderForm.cs`: persis apa yang boleh dikirim formulir pesanan baru, beserta aturannya, termasuk satu aturan yang melibatkan dua isian.

```
using System.ComponentModel.DataAnnotations;

namespace LaundryMvc.Models;

// What the counter form may send: nothing about Total, Status or dates.
public class OrderForm : IValidatableObject
{
    [Range(1, int.MaxValue, ErrorMessage = "Choose a customer.")]
    [Display(Name = "Customer")]
    public int CustomerId { get; set; }

    [Required(ErrorMessage = "Choose a service.")]
    [Display(Name = "Service")]
    public string? ServiceId { get; set; }

    [Range(typeof(decimal), "0.1", "100",
        ParseLimitsInInvariantCulture = true,
        ErrorMessage = "The quantity must be between 0.1 and 100.")]
    [Display(Name = "Quantity (kg, items or pairs)")]
    public decimal Quantity { get; set; } = 1;

    [Display(Name = "Pickup and delivery")]
    public bool Pickup { get; set; }

    [StringLength(200)]
    [Display(Name = "Delivery address")]
    public string? DeliveryAddress { get; set; }

    [StringLength(200)]
    public string? Note { get; set; }

    // A rule about two fields together: delivery needs an address.
    public IEnumerable<ValidationResult> Validate(ValidationContext context)
    {
        if (Pickup && string.IsNullOrWhiteSpace(DeliveryAddress))
        {
            yield return new ValidationResult(
                "Pickup and delivery needs a delivery address.",
                [nameof(DeliveryAddress)]);
        }
    }
}
```

6.4.2 Aturan tentang dua isian

"Antar-jemput memerlukan alamat antar" melibatkan dua properti, sehingga tidak dapat menjadi atribut pada salah satunya. Kelas yang mengimplementasikan `IValidatableObject` memiliki metode `Validate` tempat aturan apa pun dapat ditulis; metode ini mengembalikan `ValidationResult` untuk setiap kegagalan, lengkap dengan nama properti pemilik pesannya. `yield return` menghasilkan hasil-hasil itu satu per satu, sehingga metode dengan beberapa aturan cukup memiliki beberapa blok `if`.

Keluaran `ValidationDemo` memperlihatkan dua rincian yang patut diingat. Pada kasus 5 alamatnya kosong dan pesannya dilekatkan pada `DeliveryAddress`, sehingga formulir menampilkannya di samping isian itu. Pada kasus 4, dengan jumlah nol dan tanpa pelanggan, metode `Validate` sama sekali tidak dijalankan: `Validator` memeriksa atribut lebih dulu dan menjalankan `IValidatableObject.Validate` hanya bila semuanya lolos. Ingat urutan ini ketika Anda heran mengapa sebuah pesan belum muncul: pesan itu akan muncul setelah kesalahan yang lebih sederhana diperbaiki.

`[Range(typeof(decimal), "0.1", "100", ParseLimitsInInvariantCulture = true)]` juga perlu dijelaskan. Argumen atribut harus konstanta, dan nilai `decimal` bukan konstanta di C#, sehingga batasnya ditulis sebagai teks lalu dikonversi. `ParseLimitsInInvariantCulture` membuat konversi itu memakai titik sebagai pemisah desimal di setiap komputer; tanpanya, server yang diatur ke bahasa Indonesia akan membaca `"0.1"` dengan aturan koma dan mendapat batas yang berbeda.

Formulir pembayaran (**Kode 6.6**) lebih kecil: jumlah antara Rp 1.000 dan Rp 10.000.000 dan cara bayar yang harus merupakan salah satu nilai enum. `[EnumDataType]` penting karena *model binding* dengan senang hati memasukkan angka apa pun ke enum: `Method=7` akan menjadi `(PaymentMethod)7` lalu tersimpan.

Kode 6.6 `Models/PaymentForm.cs`: formulir pembayaran dan aturannya.

```
using System.ComponentModel.DataAnnotations;

namespace LaundryMvc.Models;

public class PaymentForm
{
    [Range(1000, 10_000_000,
        ErrorMessage = "A payment must be between Rp 1.000 and Rp 10.000.000.")]
    public int Amount { get; set; }

    [EnumDataType(typeof(PaymentMethod),
        ErrorMessage = "Choose cash, QRIS or transfer.")]
    public PaymentMethod Method { get; set; }
}
```

6.4.3 *Controller* dengan kelas formulir

Kode 6.7 adalah `OrdersController` setelah perubahan. `Create` menerima `OrderForm`; ketika baris pertamanya berjalan, atribut dan `Validate` sudah diperiksa. *Controller* menambahkan dua aturan yang memerlukan basis data, dan hanya bila semuanya sah ia membangun entitas `Order` dari formulir, isian demi isian. Penyalinan eksplisit ini adalah harga dari kelas formulir, dan layak dibayar: setiap nilai yang sampai ke basis data melewati sebuah baris yang dapat Anda baca.

Kode 6.7 `Controllers/OrdersController.cs` dengan `OrderForm` dan `PaymentForm`.

```
using LaundryMvc.Data;
using LaundryMvc.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;

namespace LaundryMvc.Controllers;

public class OrdersController(LaundryDbContext db, PriceCalculator calculator)
    : Controller
{
    // GET /Orders or /Orders?status=Queued
    public async Task<IActionResult> Index(OrderStatus? status)
    {
        var query = db.Orders
            .Include(o => o.Customer)
            .Include(o => o.Service)
            .Include(o => o.Payments)
            .AsNoTracking();
        if (status is not null)
        {
            query = query.Where(o => o.Status == status);
        }
        ViewData["Status"] = status;
        return View(await query.OrderByDescending(o => o.ReceivedAt)
                               .ToListAsync());
    }

    public async Task<IActionResult> Details(int id)
    {
        var order = await db.Orders
            .Include(o => o.Customer)
            .Include(o => o.Service)
            .Include(o => o.Payments)
            .AsNoTracking()
            .FirstOrDefaultAsync(o => o.Id == id);
        return order is null ? NotFound() : View(order);
    }

    public async Task<IActionResult> Create(int? customerId)
    {
        await FillListsAsync();
        return View(new OrderForm { CustomerId = customerId ?? 0 });
    }
```

Kode 6.7 (Lanjutan) *Controllers/OrdersController.cs dengan OrderForm dan PaymentForm.*

```
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Create(OrderForm form)
    {
        // The attributes on OrderForm have been checked already; these two
        // rules need the database, so the controller checks them.
        var service = await db.Services.FindAsync(form.ServiceId ?? "");
        if (form.ServiceId is not null && (service is null || !service.IsActive))
        {
            ModelState.AddModelError(nameof(form.ServiceId),
                "This service is not offered any more.");
        }
        if (form.CustomerId > 0
            && !await db.Customers.AnyAsync(c => c.Id == form.CustomerId))
        {
            ModelState.AddModelError(nameof(form.CustomerId),
                "This customer does not exist.");
        }
        if (!ModelState.IsValid || service is null)
        {
            await FillListsAsync();
            return View(form);
        }

        // The price rules of Chapter 3: minimum 3 kg, whole items, pickup fee.
        var quote = calculator.Calculate(service, form.Quantity, form.Pickup);
        var order = new Order
        {
            CustomerId = form.CustomerId,
            ServiceId = service.Id,
            Quantity = form.Quantity,
            Pickup = form.Pickup,
            DeliveryAddress = form.Pickup ? form.DeliveryAddress : null,
            Note = form.Note,
            Total = (int)decimal.Round(quote.Total,
                                       MidpointRounding.AwayFromZero),
            Status = OrderStatus.Queued,
            ReceivedAt = DateTime.Now
        };
        db.Orders.Add(order);
        await db.SaveChangesAsync();
        TempData["Message"] = $"Order #{order.Id} created: {Money.Rp(order.Total)}."
            + (quote.Notes.Count > 0 ? " " + string.Join(" ", quote.Notes) : "");
        return RedirectToAction(nameof(Details), new { id = order.Id });
    }
```

Kode 6.7 (Lanjutan) *Controllers/OrdersController.cs dengan OrderForm dan PaymentForm.*

```
    // Queued -> Finished -> Departed, each step with its time stamp.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Advance(int id)
    {
        var order = await db.Orders.FindAsync(id);
        if (order is null) return NotFound();
        switch (order.Status)
        {
            case OrderStatus.Queued:
                order.Status = OrderStatus.Finished;
                order.FinishedAt = DateTime.Now;
                break;
            case OrderStatus.Finished:
                order.Status = OrderStatus.Departed;
                order.DepartedAt = DateTime.Now;
                break;
            default:
                TempData["Error"] = "This order has already left the shop.";
                return RedirectToAction(nameof(Details), new { id });
        }
        await db.SaveChangesAsync();
        TempData["Message"] = $"Order #{id} is now {order.Status}.";
        return RedirectToAction(nameof(Details), new { id });
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> AddPayment(int id, PaymentForm payment)
    {
        if (!await db.Orders.AnyAsync(o => o.Id == id)) return NotFound();
        if (!ModelState.IsValid)
        {
            TempData["Error"] = ModelState.Values.SelectMany(v => v.Errors)
                                             .First().ErrorMessage;
            return RedirectToAction(nameof(Details), new { id });
        }
        db.Payments.Add(new Payment
        {
            OrderId = id, Amount = payment.Amount, Method = payment.Method,
            PaidAt = DateTime.Now
        });
        await db.SaveChangesAsync();
        TempData["Message"] = $"Payment of {Money.Rp(payment.Amount)} recorded.";
        return RedirectToAction(nameof(Details), new { id });
    }
```

Kode 6.7 (Lanjutan) *Controllers/OrdersController.cs dengan OrderForm dan PaymentForm.*

```
    async Task FillListsAsync()
    {
        ViewBag.Customers = new SelectList(
            await db.Customers.OrderBy(c => c.Name).ToListAsync(),
            nameof(Customer.Id), nameof(Customer.Name));
        ViewBag.Services = new SelectList(
            await db.Services.Where(s => s.IsActive).OrderBy(s => s.Id)
                             .ToListAsync(),
            nameof(Service.Id), nameof(Service.Description));
    }
}
```

Perhatikan `DeliveryAddress = form.Pickup ? form.DeliveryAddress : null`. Kasir bisa mengetik alamat, berubah pikiran, lalu menghapus centang antar-jemput; alamat itu kemudian

tidak disimpan, karena sudah tidak bermakna. `AddPayment` menerima `PaymentForm`; bila tidak sah, pesan pertamanya ditampilkan di halaman pesanan melalui `TempData["Error"]`, karena formulir pembayaran adalah bagian dari halaman itu, bukan halaman tersendiri.

View-nya (**Kode 6.8**) adalah formulir Bab 5 dengan `@model OrderForm`, label yang berasal dari nama `[Display]`, dan tempat pesan di bawah alamat antar.

Kode 6.8 `Views/Orders/Create.cshtml` yang terikat pada `OrderForm`.

```
@model OrderForm
@{
    ViewData["Title"] = "New order";
}
<h1>New order</h1>
<form asp-action="Create" method="post" class="form">
    <div asp-validation-summary="ModelOnly" class="errors"></div>
    <label asp-for="CustomerId"></label>
    <select asp-for="CustomerId" asp-items="ViewBag.Customers">
        <option value="0">- choose -</option>
    </select>
    <span asp-validation-for="CustomerId" class="error"></span>

    <label asp-for="ServiceId"></label>
    <select asp-for="ServiceId" asp-items="ViewBag.Services">
        <option value="">- choose -</option>
    </select>
    <span asp-validation-for="ServiceId" class="error"></span>

    <label asp-for="Quantity"></label>
    <input asp-for="Quantity" type="number" step="0.01" min="0.01" />
    <span asp-validation-for="Quantity" class="error"></span>

    <label class="check"><input asp-for="Pickup" /> Pickup and delivery</label>
    <label asp-for="DeliveryAddress"></label>
    <input asp-for="DeliveryAddress" />
    <span asp-validation-for="DeliveryAddress" class="error"></span>

    <label asp-for="Note"></label>
    <input asp-for="Note" />
    <span asp-validation-for="Note" class="error"></span>

    <button type="submit">Create order</button>
    <a asp-action="Index">Cancel</a>
</form>
```

6.5 Pemeriksaan di Peramban Juga

Peramban sudah melakukan sebagian pemeriksaan dengan atribut HTML yang ditulis *tag helper*: `type="number"`, `min="0.01"`, dan `step="0.01"` pada isian jumlah menahan huruf atau angka negatif sebelum formulir dikirim. ASP.NET Core dapat berbuat lebih. Untuk setiap atribut validasi, *tag helper* juga menulis atribut `data-val-*` ke HTML; misalnya, properti `[Required]` mendapat `data-val="true" data-val-required="Please enter a phone number."`. Pustaka JavaScript kecil *jQuery Validation* dan *jQuery Validation Unobtrusive* membaca atribut-atribut itu dan menampilkan pesan yang sama selagi pengguna mengetik, tanpa bolak-balik ke server. Templat proyek `dotnet new mvc` menyertakan kedua pustaka itu di `wwwroot/lib` dan *partial view* `_ValidationScriptsPartial.cshtml` yang memuatnya; halaman formulir menyalakannya dengan `@section Scripts { <partial name="_ValidationScriptsPartial" /> }` [1].

Proyek Profitina Laundry sengaja tidak menyertakannya, seperti di Bab 3, agar halaman bebas dari kerangka kerja dan agar pokok Subbab 6.2 terlihat: setiap aturan di bab ini diperiksa di

server, dan halaman dengan pemeriksaan di sisi klien hanya menghemat waktu pengguna sesaat. Aturan yang memerlukan basis data, dan aturan `IValidatableObject`, bagaimanapun hanya diperiksa di server.

6.6 Identity: Siapa di Balik Meja Kasir?

6.6.1 Apa yang disediakan Identity

Menulis fitur masuk sendiri tampak mudah padahal tidak: kata sandi tidak boleh disimpan, hanya *hash* bergaram yang lambat dari kata sandi itu; percobaan gagal harus dihitung dan akun dikunci sementara; cookie harus ditandatangani dan dienkripsi agar tidak dapat dipalsukan; kata sandi yang diganti harus membatalkan cookie lama. ASP.NET Core Identity melakukan semua ini dan telah melakukannya selama bertahun-tahun [2]. Sebagian besar Identity adalah bagian dari *shared framework* yang sudah dimiliki setiap aplikasi ASP.NET Core; bagian yang menyimpan pengguna dan peran dengan EF Core adalah satu paket NuGet lagi, `Microsoft.AspNetCore.Identity.EntityFrameworkCore`, yang ditambahkan seperti paket EF Core di Bab 5:

```
dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore \
    --version 10.0.12
```

DbContext kemudian diturunkan dari `IdentityDbContext<IdentityUser>` alih-alih `DbContext`, yang menambahkan tabel-tabel Identity (`AspNetUsers`, `AspNetRoles`, `AspNetUserRoles`, dan beberapa lagi) ke model. **Kode 6.9** memperlihatkan baris-baris yang berubah; pemanggilan `base.OnModelCreating(model)` harus diletakkan paling awal, karena pemanggilan itulah yang mengatur tabel-tabel tersebut.

Kode 6.9 `Data/LaundryDbContext.cs` yang diturunkan dari `IdentityDbContext`, sehingga pengguna dan peran tinggal di basis data yang sama dengan pesanan.

```
using LaundryMvc.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace LaundryMvc.Data;

public class LaundryDbContext(DbContextOptions<LaundryDbContext> options)
    : IdentityDbContext<IdentityUser>(options)
{
    public DbSet<Service> Services => Set<Service>();
    public DbSet<Outlet> Outlets => Set<Outlet>();
    public DbSet<Customer> Customers => Set<Customer>();
    public DbSet<Order> Orders => Set<Order>();
    public DbSet<Payment> Payments => Set<Payment>();

    protected override void OnModelCreating(ModelBuilder model)
    {
        base.OnModelCreating(model);      // the AspNet* tables of Identity

        model.Entity<Service>(e =>
        {
            e.Property(s => s.Id).HasMaxLength(5);
            e.Property(s => s.Description).HasMaxLength(60);
            e.Property(s => s.Price).HasPrecision(10, 2);
        });

        model.Entity<Outlet>(e =>
        {
            e.Property(o => o.Id).HasMaxLength(3);
            e.Property(o => o.Name).HasMaxLength(60);
            e.Property(o => o.Address).HasMaxLength(120);
            e.Property(o => o.Phone).HasMaxLength(20);
        });

        model.Entity<Customer>(e =>
        {
            e.Property(c => c.Name).HasMaxLength(80);
            e.Property(c => c.Phone).HasMaxLength(20);
            e.Property(c => c.Address).HasMaxLength(200);
            e.HasIndex(c => c.Phone).IsUnique();
        });
```

Kode 6.9 (Lanjutan) *Data/LaundryDbContext.cs yang diturunkan dari IdentityDbContext, sehingga pengguna dan peran tinggal di basis data yang sama dengan pesanan.*

```
        model.Entity<Order>(e =>
        {
            e.Property(o => o.Quantity).HasPrecision(6, 2);
            e.Property(o => o.Status).HasConversion<string>().HasMaxLength(12);
            e.Property(o => o.DeliveryAddress).HasMaxLength(200);
            e.Property(o => o.Note).HasMaxLength(200);
            e.HasIndex(o => o.Status);
            e.HasOne(o => o.Customer).WithMany(c => c.Orders)
             .HasForeignKey(o => o.CustomerId)
             .OnDelete(DeleteBehavior.Restrict);
            e.HasOne(o => o.Service).WithMany()
             .HasForeignKey(o => o.ServiceId)
             .OnDelete(DeleteBehavior.Restrict);
        });

        model.Entity<Payment>(e =>
        {
            e.Property(p => p.Method).HasConversion<string>().HasMaxLength(10);
            e.HasOne(p => p.Order).WithMany(o => o.Payments)
             .HasForeignKey(p => p.OrderId)
             .OnDelete(DeleteBehavior.Cascade);
        });
    }
}
```

Migrasi baru (`dotnet ef migrations add AddIdentity`, atau `InitialCreate` pada proyek baru) membuat tabel-tabelnya. Menyimpan pengguna di berkas SQLite yang sama dengan pesanan berarti satu berkas untuk dicadangkan, dan satu tempat tempat data pelanggan dan akun staf dilindungi bersama.

6.6.2 Program.cs dengan Identity, otorisasi, dan pengaturan penerapan

Kode 6.10 adalah `Program.cs` yang baru. Bacalah mengikuti urutan komentarnya.

Kode 6.10 `Program.cs`: Identity, kebijakan otorisasi cadangan, dan pengaturan yang diperlukan untuk penerapan.

```
using System.Diagnostics;
using LaundryMvc.Data;
using LaundryMvc.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.DataProtection;
using Microsoft.AspNetCore.HttpOverrides;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// 1. Services.
builder.Services.AddControllersWithViews();
builder.Services.AddSingleton<PriceCalculator>();

var dataFolder = Path.Combine(builder.Environment.ContentRootPath, "data");
builder.Services.AddDbContext<LaundryDbContext>(options =>
    options.UseSqlite(builder.Configuration.GetConnectionString("Laundry"))
           .UseSeeding((db, _) =>
               SeedData.Apply((LaundryDbContext)db, dataFolder))
           .UseAsyncSeeding((db, _, ct) =>
               SeedData.ApplyAsync((LaundryDbContext)db, dataFolder, ct)));
builder.Services.AddScoped<ILaundryCatalog, EfCatalog>();

// Chapter 6: users, hashed passwords, roles and lockout, stored by EF Core.
builder.Services
    .AddIdentity<IdentityUser, IdentityRole>(options =>
    {
        options.Password.RequiredLength = 8;
        options.User.RequireUniqueEmail = true;
        options.Lockout.MaxFailedAccessAttempts = 5;
        options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(15);
    })
    .AddEntityFrameworkStores<LaundryDbContext>();

builder.Services.ConfigureApplicationCookie(options =>
{
    options.LoginPath = "/Account/Login";
    options.AccessDeniedPath = "/Account/AccessDenied";
    options.ExpireTimeSpan = TimeSpan.FromHours(10);   // one working day
    options.SlidingExpiration = true;
});
```

Kode 6.10 (Lanjutan) *Program.cs: Identity, kebijakan otorisasi cadangan, dan pengaturan yang diperlukan untuk penerapan.*

```
// Every page needs a signed-in user unless it says [AllowAnonymous].
builder.Services.AddAuthorizationBuilder()
    .SetFallbackPolicy(new AuthorizationPolicyBuilder()
        .RequireAuthenticatedUser()
        .Build());

// Deployment helpers: a health endpoint, cookie keys that survive restarts,
// and the real client address and scheme when running behind a proxy.
builder.Services.AddHealthChecks();
builder.Services.AddDataProtection()
    .PersistKeysToFileSystem(new DirectoryInfo(
        builder.Configuration["DataProtection:KeysPath"]
        ?? Path.Combine(builder.Environment.ContentRootPath, "keys")));
builder.Services.Configure<ForwardedHeadersOptions>(options =>
    options.ForwardedHeaders = ForwardedHeaders.XForwardedFor
                             | ForwardedHeaders.XForwardedProto);

var app = builder.Build();

await IdentitySeeder.SeedAsync(app.Services);

// 2. Pipeline.
app.UseForwardedHeaders();
app.Use(async (context, next) =>
{
    var clock = Stopwatch.StartNew();
    await next(context);
    app.Logger.LogInformation("{Method} {Path}{Query} -> {Status} in {Ms} ms",
        context.Request.Method, context.Request.Path,
        context.Request.QueryString, context.Response.StatusCode,
        clock.ElapsedMilliseconds);
});

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/error/500");
    app.UseHsts();
}
app.UseStatusCodePagesWithReExecute("/error/{0}");

app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();
```

Kode 6.10 (Lanjutan) *Program.cs: Identity, kebijakan otorisasi cadangan, dan pengaturan yang diperlukan untuk penerapan.*

```
app.MapStaticAssets().AllowAnonymous();
app.MapHealthChecks("/health").AllowAnonymous();
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}")
    .WithStaticAssets();

app.Run();
```

- `AddIdentity<IdentityUser, IdentityRole>` mendaftarkan pengelola pengguna, pengelola peran, dan pengelola masuk, dan `AddEntityFrameworkStores<LaundryDbContext>()` meminta semuanya menyimpan data melalui DbContext kita. Opsinya mewajibkan kata sandi minimal delapan karakter, satu akun per alamat surel, dan penguncian 15 menit setelah lima kali gagal.

- `ConfigureApplicationCookie` menetapkan ke mana pengguna dikirim untuk masuk dan di mana penolakan ditampilkan, serta membuat cookie bertahan sepuluh jam, satu hari kerja, dan diperpanjang selama pengguna aktif.
- `SetFallbackPolicy(…RequireAuthenticatedUser())` adalah baris terpenting di bab ini. Baris ini membalik bawaannya: setiap *endpoint* kini mewajibkan pengguna yang sudah masuk kecuali menyatakan lain. *Controller* baru yang ditambahkan seseorang tahun depan tetap terlindungi meskipun ia lupa memasang atribut. Halaman publik Bab 3 memilih keluar secara eksplisit, seperti diperlihatkan subbab berikutnya.
- Di *pipeline*, `UseAuthentication()` membaca cookie dan mengisi `User`, dan `UseAuthorization()` menerapkan aturan. Keduanya datang setelah `UseRouting()`, karena otorisasi perlu tahu *endpoint* mana yang dipilih dan atribut apa yang dimilikinya, dan sebelum *endpoint* dijalankan.
- `MapStaticAssets().AllowAnonymous()` menjaga lembar gaya tetap dapat dijangkau pengunjung yang belum masuk; tanpanya, kebijakan cadangan ikut melindungi `site.css`, dan halaman masuk akan tampil tanpa gaya.

Baris-baris sisanya (*forwarded headers*, *data protection*, *health checks*, HSTS) termasuk urusan penerapan dan dijelaskan di Subbab 6.8.

6.6.3 Apa yang disimpan sebagai pengganti kata sandi

Bagian 8 keluaran `ValidationDemo` memperlihatkan hasil `PasswordHasher<IdentityUser>` milik Identity dari kata sandi `Kasir#2026`: teks sepanjang 84 karakter, bentuk Base64 dari 61 bita. Bita pertama, `0x01`, menandai formatnya; lalu menyusul fungsi pseudo-acak (HMAC-SHA512), jumlah iterasi (100.000), panjang garam (16 bita), garamnya, dan kunci 32 bita yang diturunkan dengan PBKDF2. Memverifikasi kata sandi yang benar menghasilkan `Success`, dan kata sandi yang berbeda satu huruf kapital menghasilkan `Failed`. Baris terakhir paling mengajarkan: meng-*hash* kata sandi yang sama untuk kedua kalinya menghasilkan teks yang *berbeda*, karena setiap *hash* mendapat garam acak yang baru. Dua pengguna dengan kata sandi sama karenanya memiliki *hash* berbeda, dan tabel *hash* yang dihitung sebelumnya tidak berguna bagi penyerang.

Iterasi sebanyak 100.000 itulah yang sengaja membuat *hash* lambat: memeriksa satu tebakan memerlukan sepersekian milidetik yang terasa, yang tidak berarti apa-apa bagi kasir yang masuk sekali sehari tetapi sangat berarti bagi orang yang mencoba miliaran tebakan. *OWASP Password Storage Cheat Sheet* merekomendasikan PBKDF2-HMAC-SHA512 hanya dengan jumlah iterasi yang tinggi, dan lebih memilih Argon2id bila tersedia [3]; *hasher* Identity dapat diganti melalui *dependency injection* bila kebijakan toko mewajibkan algoritme lain. Apa pun algoritmenya, aturan bab ini tetap: aplikasi menyimpan dan membandingkan *hash*, tidak pernah kata sandi.

6.6.4 Peran dan akun pertama

Toko memiliki dua peran, dinamai sekali sebagai konstanta (**Kode 6.11**) sehingga salah ketik pada nama peran menjadi galat kompilasi, bukan pintu yang diam-diam tidak pernah terbuka.

Kode 6.11 `Models/AppRoles.cs`: dua peran di toko.

```
namespace LaundryMvc.Models;

public static class AppRoles
{
    public const string Owner = "Owner";      // prices, deleting customers
    public const string Cashier = "Cashier";  // orders and payments
}
```

Harus ada seseorang sebelum siapa pun dapat masuk. `IdentitySeeder` (**Kode 6.12**) berjalan sekali saat aplikasi dimulai, tepat setelah `builder.Build()`: ia menerapkan migrasi yang tertunda (yang juga menjalankan pengisian data Bab 5), membuat dua peran bila belum ada, dan membuat setiap akun yang tercantum di bawah `SeedUsers` pada konfigurasi yang belum ada. `UserManager.CreateAsync(user, password)` memeriksa kata sandi terhadap aturan `AddIdentity` dan menyimpan *hash*-nya; bila kata sandinya terlalu pendek, aplikasi berhenti saat dimulai dengan pesan yang jelas, alih-alih membuat akun yang tidak dapat dipakai siapa pun.

Kode 6.12 `Data/IdentitySeeder.cs`: migrasi, peran, dan akun pertama saat aplikasi dimulai.

```
using LaundryMvc.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;

namespace LaundryMvc.Data;

// Brings the database up to date and makes sure the roles and the accounts
// listed under "SeedUsers" in the configuration exist.
public static class IdentitySeeder
{
    public static async Task SeedAsync(IServiceProvider services)
    {
        using var scope = services.CreateScope();
        var sp = scope.ServiceProvider;

        var db = sp.GetRequiredService<LaundryDbContext>();
        await db.Database.MigrateAsync();     // also runs UseAsyncSeeding

        var roles = sp.GetRequiredService<RoleManager<IdentityRole>>();
        foreach (var role in new[] { AppRoles.Owner, AppRoles.Cashier })
        {
            if (!await roles.RoleExistsAsync(role))
            {
                await roles.CreateAsync(new IdentityRole(role));
            }
        }

        var users = sp.GetRequiredService<UserManager<IdentityUser>>();
        var config = sp.GetRequiredService<IConfiguration>();
        foreach (var account in config.GetSection("SeedUsers").GetChildren())
        {
            var email = account["Email"];
            var password = account["Password"];
            var role = account["Role"];
            if (email is null || password is null || role is null) continue;
            if (await users.FindByEmailAsync(email) is not null) continue;
```

Kode 6.12 (Lanjutan) *Data/IdentitySeeder.cs: migrasi, peran, dan akun pertama saat aplikasi dimulai.*

```
            var user = new IdentityUser
            {
                UserName = email, Email = email, EmailConfirmed = true
            };
            var result = await users.CreateAsync(user, password);  // hashes it
            if (!result.Succeeded)
            {
                throw new InvalidOperationException(string.Join("; ",
                    result.Errors.Select(e => e.Description)));
            }
            await users.AddToRoleAsync(user, role);
        }
    }
}
```

Akun demo tinggal di `appsettings.Development.json` (**Kode 6.13**), berkas yang hanya dibaca ketika lingkungannya *Development*, yaitu yang dipakai `dotnet run` melalui `launchSettings.json`. Di server produksi berkas itu diabaikan, sehingga kata sandi demo tidak pernah dapat menjadi kata sandi sungguhan. Akun pemilik sungguhan yang pertama dibuat di sana dari variabel lingkungan, sekali saja (Subbab 6.8.4).

Kode 6.13 `appsettings.Development.json`: akun demo hanya untuk komputer pengembang.

```
{
  "SeedUsers": [
    { "Email": "owner@profitina-laundry.example", "Password": "Owner#2026",
      "Role": "Owner" },
    { "Email": "kasir1@profitina-laundry.example", "Password": "Kasir#2026",
      "Role": "Cashier" }
  ]
}
```

6.7 Masuk dan Menjaga Pintu Tetap Tertutup

6.7.1 Halaman masuk

`AccountController` (**Kode 6.14**) memiliki empat *action*: formulir masuk, pengirimannya, keluar, dan halaman penolakan. Seluruh kelas ditandai `[AllowAnonymous]`, karena halaman masuk yang mewajibkan masuk adalah pintu terkunci dengan kunci di dalamnya.

Kode 6.14 `Controllers/AccountController.cs`: masuk, keluar, dan halaman penolakan.

```
using LaundryMvc.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace LaundryMvc.Controllers;

[AllowAnonymous]
public class AccountController(SignInManager<IdentityUser> signIn) : Controller
{
    public IActionResult Login(string? returnUrl) =>
        View(new LoginForm { ReturnUrl = returnUrl });

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Login(LoginForm form)
    {
        if (!ModelState.IsValid) return View(form);

        var result = await signIn.PasswordSignInAsync(form.Email, form.Password,
            form.RememberMe, lockoutOnFailure: true);
        if (result.Succeeded)
        {
            // Only ever return to a page of this site (no open redirect).
            return Url.IsLocalUrl(form.ReturnUrl)
                ? LocalRedirect(form.ReturnUrl)
                : RedirectToAction("Index", "Counter");
        }
        ModelState.AddModelError(string.Empty, result.IsLockedOut
            ? "Too many failed attempts. Please wait 15 minutes."
            : "The e-mail or the password is wrong.");
        return View(form);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Logout()
    {
        await signIn.SignOutAsync();
        return RedirectToAction("Index", "Home");
    }

    public IActionResult AccessDenied() => View();
}
```

`PasswordSignInAsync(email, password, rememberMe, lockoutOnFailure: true)` mencari pengguna, memverifikasi kata sandi terhadap *hash* yang tersimpan, menghitung kegagalan untuk penguncian, dan bila berhasil menulis cookie autentikasi. Dengan `RememberMe`, cookie bertahan setelah peramban ditutup; tanpanya, cookie hanya bertahan selama sesi peramban. Pesan galatnya sengaja sama untuk surel yang tidak dikenal dan kata sandi yang salah: memberi tahu penyerang alamat surel mana yang ada hanya akan membantunya.

Dua baris melindungi dari serangan yang lebih halus. Ketika kebijakan cadangan mengirim pengguna anonim ke halaman masuk, ia menambahkan `?ReturnUrl=/Prices` agar pengguna kembali ke sana sesudahnya. Penyerang dapat mengirim tautan dengan `ReturnUrl=https://evil.example/`, berharap halaman masuk milik toko sendiri meneruskan kasir ke tiruannya; `Url.IsLocalUrl` hanya menerima alamat di dalam situs ini. Dan keluar adalah `POST` dengan token *anti-forgery*, sehingga situs lain tidak dapat mengeluarkan kasir dengan sebuah tag gambar. **Kode 6.15** adalah *view* masuk; `autocomplete="current-password"` memungkinkan pengelola kata sandi membantu.

Kode 6.15 `Views/Account/Login.cshtml`: formulir masuk staf.

```
@model LoginForm
@{
    ViewData["Title"] = "Staff sign-in";
}
<h1>Staff sign-in</h1>
<form asp-action="Login" method="post" class="form">
    <div asp-validation-summary="ModelOnly" class="errors"></div>
    <input type="hidden" asp-for="ReturnUrl" />

    <label asp-for="Email"></label>
    <input asp-for="Email" autocomplete="username" />
    <span asp-validation-for="Email" class="error"></span>

    <label asp-for="Password"></label>
    <input asp-for="Password" autocomplete="current-password" />
    <span asp-validation-for="Password" class="error"></span>

    <label class="check"><input asp-for="RememberMe" /> Keep me signed in</label>
    <button type="submit">Sign in</button>
</form>
```

6.7.2 Siapa boleh melakukan apa

Dengan kebijakan cadangan terpasang, aturan-aturannya ditulis sebagai atribut. **Tabel 6.2** merangkumnya.

Tabel 6.2 Siapa boleh memakai halaman mana setelah bab ini.

Halaman	Pengunjung anonim	Kasir	Pemilik
Beranda, `/services`, penawaran harga, `/outlets`, halaman galat	ya	ya	ya
`/health`, lembar gaya	ya	ya	ya
Counter, Orders, pembayaran	halaman masuk	ya	ya
Customers: daftar, tambah, ubah	halaman masuk	ya	ya
Customers: hapus	halaman masuk	ditolak	ya
Prices	halaman masuk	ditolak	ya

Tiga *controller* Bab 3 masing-masing mendapat satu baris, `[AllowAnonymous]`, di atas kelasnya: wajah publik toko tetap publik. `PricesController` mendapat `[Authorize(Roles = AppRoles.Owner)]` pada kelasnya, sehingga setiap *action* di dalamnya mewajibkan peran Owner. Di `CustomersController`, hanya dua *action* hapus yang mendapat atribut itu, karena kasir memang perlu menambah dan mengubah pelanggan. Kasir yang mengetik `/Prices` di bilah alamat sudah masuk tetapi bukan pemilik; otorisasi gagal, dan penangan cookie mengalihkan ke `/Account/AccessDenied`, halaman pada **Kode 6.16**.

Kode 6.16 `Views/Account/AccessDenied.cshtml`: yang dilihat pengguna yang sudah masuk ketika sebuah halaman bukan untuk perannya.

```
@{
    ViewData["Title"] = "Not allowed";
}
<h1>Not allowed</h1>
<p>You are signed in as @User.Identity?.Name, but this page is for the
   shop owner. Ask the owner to make this change.</p>
<p><a asp-controller="Counter" asp-action="Index">Back to the counter</a></p>
```

Menyembunyikan tautan bukanlah perlindungan, tetapi merupakan sopan santun. *Layout* menampilkan tautan staf hanya kepada pengguna yang sudah masuk dan tautan *Prices* hanya

kepada pemilik, dengan `User.Identity?.IsAuthenticated` dan `User.IsInRole(AppRoles.Owner)`. Atribut pada *controller*-lah yang sebenarnya melindungi halaman; `if` di *layout* hanya menghindarkan kasir melihat tautan yang akan berujung penolakan.

6.7.3 Kebijakan: aturan yang bernama

Peran menjawab pertanyaan "apakah pengguna ini pemilik?". Toko sungguhan segera memiliki aturan yang bukan peran: *hanya staf gerai Keputih yang boleh melihat pesanan Keputih*, *pengembalian uang di atas Rp 100.000 memerlukan pemilik*, *kasir hanya boleh mencatat pembayaran selama toko buka*. ASP.NET Core menyatakan aturan semacam itu sebagai *policy* (kebijakan), aturan bernama yang didaftarkan sekali di `Program.cs` dan dirujuk menurut namanya [4]. Aturan peran di bab ini sendiri dapat ditulis sebagai kebijakan:

```
builder.Services.AddAuthorizationBuilder().AddPolicy("OwnerOnly", p => p.RequireRole(AppRoles.Owner));
```

lalu sebuah *controller* cukup menyatakan `[Authorize(Policy = "OwnerOnly")]`. Keuntungannya, aturan itu tinggal di satu tempat. Ketika kelak pemilik memutuskan bahwa kasir senior yang tepercaya juga boleh mengubah harga, kebijakannya menjadi `p.RequireRole(AppRoles.Owner, "SeniorCashier")` dan tidak satu *controller* pun berubah. Kebijakan juga dapat mewajibkan *claim*, sepotong informasi yang disimpan bersama pengguna, misalnya `p.RequireClaim("outlet", "O02")`; Identity menyimpan *claim* di tabel `AspNetUserClaims`, dan `UserManager.AddClaimAsync` menambahkannya. Untuk aturan yang memerlukan data dari permintaan atau basis data, seperti jam buka, sebuah kebijakan dapat diberi *requirement* buatan sendiri dan kelas *handler* yang memutuskannya; hal itu di luar cakupan bab ini, tetapi atribut pada *controller* akan tampak persis sama.

Kebijakan cadangan di Subbab 6.6.2 tidak lain adalah kebijakan yang dipakai ketika sebuah *endpoint* tidak menyebut kebijakan apa pun. Bersama-sama, kebijakan cadangan, `[AllowAnonymous]`, dan kebijakan bernama mencakup semua yang dibutuhkan toko kecil: tertutup secara bawaan, terbuka di tempat yang Anda tentukan, dan aturan khusus dengan nama yang menjelaskan dirinya sendiri.

6.8 Penerapan: Dari Laptop ke Server

6.8.1 Lingkungan dan pengaturan

ASP.NET Core membaca konfigurasinya berlapis: `appsettings.json` lebih dulu, lalu `appsettings.{Environment}.json`, lalu variabel lingkungan, lalu argumen baris perintah, setiap lapis menimpa lapis sebelumnya [5]. Lingkungan dinamai oleh variabel `ASPNETCORE_ENVIRONMENT`: *Development* di laptop (diatur di `launchSettings.json`), *Production* di server (bawaan bila variabel itu tidak diatur). **Kode 6.17** adalah lapis produksinya.

Kode 6.17 `appsettings.Production.json`: basis data dan kunci cookie di folder tetap, serta log yang lebih tenang.

```
{
  "ConnectionStrings": {
    "Laundry": "Data Source=/var/lib/laundrymvc/laundry.db"
  },
  "DataProtection": {
    "KeysPath": "/var/lib/laundrymvc/keys"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  }
}
```

Lapis ini memindahkan berkas basis data ke `/var/lib/laundrymvc`, folder di luar folder program, sehingga memasang versi baru tidak pernah menyentuh data. Kunci cookie juga disimpan di sana (Subbab 6.8.6) dan log hanya mencatat peringatan, ditambah pesan mulai dan berhenti. Di Windows nilai yang sama dapat diberikan sebagai variabel lingkungan, dengan garis bawah ganda menggantikan titik dua: `ConnectionStrings__Laundry=Data Source=D:\Laundry\laundry.db`.

Pada Production, `Program.cs` juga menyalakan `UseExceptionHandler`, sehingga galat menampilkan halaman ramah dari Bab 3, bukan *stack trace*, dan `UseHsts`, yang memberi tahu peramban agar sejak saat itu hanya memakai HTTPS untuk situs ini.

6.8.2 Menerbitkan

`dotnet publish -c Release -o <folder>` mengompilasi aplikasi dalam mode Release dan menyalin ke satu folder semua yang diperlukan untuk menjalankannya: DLL aplikasi, DLL paket-paket NuGet-nya, *view* yang sudah dikompilasi ke dalam DLL, `wwwroot`, berkas-berkas pengaturan, dan `.runtimeconfig.json` yang menyebut *runtime* .NET yang dipakai [6]. Aplikasi web tidak dapat di-*restore* di tempat buku ini ditulis, jadi inilah `ValidationDemo` yang diterbitkan dengan cara yang sama; baris-barisnya asli, dengan awal jalurnya dipersingkat:

```
  Determining projects to restore...
  Restored …/aspnet/ValidationDemo/ValidationDemo.csproj (in 185 ms).
  ValidationDemo -> …/aspnet/ValidationDemo/bin/Release/net10.0/ValidationDemo.dll
  ValidationDemo -> …/aspnet/ValidationDemo/publish/
```

Folder `publish/` kemudian berisi `ValidationDemo.dll`, `ValidationDemo.deps.json`, `ValidationDemo.pdb`, `ValidationDemo.runtimeconfig.json`, dan peluncur `ValidationDemo`, dan `dotnet ValidationDemo.dll` di folder itu menjalankan programnya. Folder `publish` milik aplikasi web tampak serupa dengan lebih banyak berkas: `LaundryMvc.dll`, DLL EF Core dan SQLite, `runtimes/` dengan pustaka SQLite asli untuk setiap sistem operasi, `wwwroot/`, `data/`, dan berkas `appsettings*.json`.

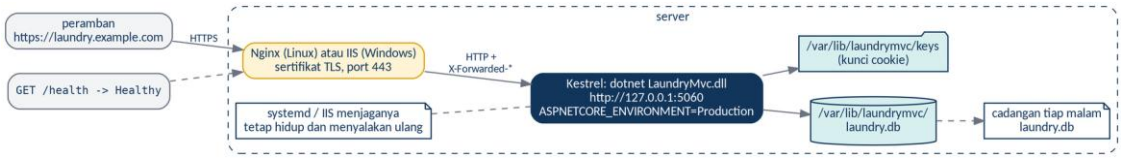
Ini adalah penerbitan *framework-dependent* (bergantung pada kerangka kerja): server harus sudah memasang *runtime* ASP.NET Core 10, dan foldernya tetap kecil. Menambahkan `--self-contained -r linux-x64` (atau `win-x64`) menghasilkan folder *self-contained* (mandiri) dengan *runtime* di dalamnya, lebih besar tetapi tidak bergantung pada apa yang terpasang di server. .NET SDK juga dapat membangun *container image* langsung dengan `dotnet publish /t:PublishContainer`, tanpa menulis Dockerfile, bagi toko yang menjalankan *container* [6].

6.8.3 Migrasi di server

Basis data di server harus dibawa ke migrasi terkini sebelum versi baru melayani permintaan. Proyek ini melakukannya saat aplikasi dimulai: `IdentitySeeder` memanggil `MigrateAsync()`. Untuk satu toko kecil dengan satu instans aplikasi, cara itu sederhana dan aman. Untuk sistem yang lebih besar, Microsoft menganjurkan penerapan migrasi sebagai langkah tersendiri dalam rilis, misalnya dengan *migration bundle*, satu berkas eksekusi yang dihasilkan `dotnet ef migrations bundle` dan menerapkan migrasi ke basis data yang disebut untai koneksinya [7]. Cara mana pun, buatlah salinan `laundry.db` sebelum menerapkan migrasi; migrasi yang menghapus kolom tidak dapat mengembalikan datanya.

6.8.4 Server Linux dengan Nginx dan systemd

Gambar 6.2 memperlihatkan susunan yang lazim di Linux. Kestrel, server web yang terpasang di setiap aplikasi ASP.NET Core, hanya mendengarkan di `127.0.0.1:5060`, sehingga tidak ada yang dari luar server dapat menjangkaunya langsung. Nginx mendengarkan di port 443, memegang sertifikat TLS, dan meneruskan permintaan ke Kestrel. systemd menyalakan aplikasi ketika server hidup dan menyalakannya ulang bila berhenti [8].



Gambar 6.2 Aplikasi laundry di server Linux: Nginx di depan dengan sertifikatnya, Kestrel di belakangnya pada alamat *loopback*, systemd yang menjaganya tetap berjalan, serta folder data berisi basis data, kunci cookie, dan cadangan setiap malam.

Kode 6.18 adalah unit systemd-nya. Unit ini menjalankan aplikasi sebagai pengguna tanpa hak istimewa bernama `laundry`, mengatur lingkungan ke Production dan alamat yang didengarkan, serta menyalakan ulang aplikasi lima detik setelah aplikasi berhenti karena sebab apa pun.

Kode 6.18 `deploy/laundrymvc.service`: unit systemd yang menjaga aplikasi tetap berjalan.

```
# /etc/systemd/system/laundrymvc.service  (Linux server)
[Unit]
Description=Profitina Laundry (ASP.NET Core)
After=network.target

[Service]
WorkingDirectory=/opt/laundrymvc
ExecStart=/usr/bin/dotnet /opt/laundrymvc/LaundryMvc.dll
Restart=always
RestartSec=5
User=laundry
Environment=ASPNETCORE_ENVIRONMENT=Production
Environment=ASPNETCORE_URLS=http://127.0.0.1:5060
Environment=DOTNET_NOLOGO=1

[Install]
WantedBy=multi-user.target
```

Kode 6.19 adalah situs Nginx-nya. Tiga baris `proxy_set_header` meneruskan nama *host* asli, alamat pengunjung, dan skema asli (`https`). `certbot --nginx` kemudian mengambil sertifikat Let's Encrypt gratis dan menambahkan baris `listen 443 ssl`.

Kode 6.19 `deploy/nginx-laundrymvc.conf`: Nginx yang meneruskan ke Kestrel beserta *forwarded headers*.

```
# /etc/nginx/sites-available/laundrymvc  (the TLS lines are added by certbot)
server {
    listen 80;
    server_name laundry.example.com;

    location / {
        proxy_pass         http://127.0.0.1:5060;
        proxy_http_version 1.1;
        proxy_set_header   Host $host;
        proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header   X-Forwarded-Proto $scheme;
    }
}
```

Di belakang *proxy*, Kestrel melihat setiap permintaan datang dari `127.0.0.1` lewat HTTP biasa. `UseForwardedHeaders()`, yang diatur di `Program.cs` untuk `X-Forwarded-For` dan `X-Forwarded-Proto`, mengembalikan alamat pengunjung dan skema yang sebenarnya ke permintaan, dan harus berjalan paling awal di *pipeline* agar semua yang sesudahnya melihat nilai yang benar [9]. Tanpanya, penanda *secure* pada cookie autentikasi, pengalihan, dan log penguncian semuanya mengira situs berjalan di HTTP dari localhost. Secara bawaan *middleware* ini hanya memercayai *forwarded headers* dari alamat *loopback*, persis tempat Nginx berada.

Pemasangannya, dalam urutan yang akan diketik seorang administrator di Ubuntu:

```
sudo apt install aspnetcore-runtime-10.0 nginx sqlite3
sudo useradd --system --home /var/lib/laundrymvc laundry
sudo mkdir -p /opt/laundrymvc /var/lib/laundrymvc
sudo chown laundry /var/lib/laundrymvc
sudo cp -r publish/* /opt/laundrymvc/
sudo cp deploy/laundrymvc.service /etc/systemd/system/
sudo systemctl edit laundrymvc      # add the first owner, see below
sudo systemctl enable --now laundrymvc
```

Langkah `systemctl edit` menambahkan tiga baris ke unit hanya untuk penyalaan pertama, `Environment=SeedUsers__0__Email=…`, `…__Password=…`, dan `…__Role=Owner`; setelah penyalaan pertama membuat akun pemilik, hapus lagi baris-baris itu. Lalu aktifkan situs Nginx, jalankan `certbot --nginx -d laundry.example.com`, dan buka situsnya.

6.8.5 Windows dengan IIS

Toko yang sudah memakai Windows Server dapat menjalankan aplikasi di IIS. Pasang *.NET 10 Hosting Bundle*, yang berisi *runtime* dan ASP.NET Core Module; buat situs yang jalur fisiknya adalah folder hasil penerbitan; atur *application pool*-nya ke *No Managed Code*. Berkas `web.config` yang ditulis `dotnet publish` memberi tahu IIS agar menyerahkan permintaan ke aplikasi melalui modul itu [10]. Pengaturan seperti untai koneksi dimasukkan ke variabel lingkungan *application pool* atau ke `appsettings.Production.json` dengan jalur Windows. IIS sendiri memegang sertifikat dan meneruskan skema aslinya, sehingga konfigurasi *forwarded headers* tidak merugikan apa-apa.

6.8.6 Hal-hal kecil yang membuatnya awet

- **Kunci cookie.** Cookie Identity dienkripsi dengan kunci dari ASP.NET Core Data Protection. Bila kunci hanya disimpan di memori, setiap penyalaan ulang membatalkan setiap cookie dan semua staf terkeluarkan. `PersistKeysToFileSystem` di `Program.cs` menyimpannya di `DataProtection:KeysPath`; folder itu harus dapat ditulisi aplikasi dan tidak boleh dihapus [11].
- ***Endpoint* kesehatan.** `AddHealthChecks()` dan `MapHealthChecks("/health")` menjawab `Healthy` dengan status 200 ketika aplikasi berjalan. Layanan pemantau, atau skrip rilis,

menanyakan `/health` setelah pembaruan dan membunyikan tanda bahaya bila jawabannya tidak datang.

- **Cadangan.** `sqlite3 /var/lib/laundrymvc/laundry.db ".backup /var/backups/laundry-$(date +%F).db"` dalam *cron job* setiap malam membuat salinan yang konsisten bahkan ketika aplikasi sedang berjalan. Salin juga cadangannya ke luar server.
- **Pembaruan.** Terbitkan versi baru, hentikan layanan, cadangkan basis data, salin berkas-berkasnya, nyalakan layanan, periksa `/health`. Enam langkah, selalu dalam urutan itu.

6.8.7 Daftar periksa rilis

Penerapan gagal karena langkah yang terlupa, bukan karena langkah yang sulit. **Tabel 6.3** adalah daftar periksa yang sebaiknya disimpan di dekat server, dalam urutan langkah-langkah itu dikerjakan.

Tabel 6.3 Daftar periksa rilis untuk server laundry.

Langkah	Perintah atau tempat	Mengapa
1. Bangun dan uji di laptop	`dotnet build`, klik mengikuti Subbab 6.9	tidak ada yang rusak sampai ke toko
2. Terbitkan	`dotnet publish -c Release -o publish`	satu folder berisi semuanya
3. Cadangkan basis data	`sqlite3 … ".backup …"`	migrasi tidak dapat dibatalkan beserta datanya
4. Hentikan layanan	`sudo systemctl stop laundrymvc`	tidak ada permintaan yang setengah tertulis
5. Salin berkas	`sudo cp -r publish/* /opt/laundrymvc/`	versi baru
6. Nyalakan layanan	`sudo systemctl start laundrymvc`	migrasi berjalan saat aplikasi dimulai
7. Periksa kesehatan	`curl https://laundry.example.com/health`	`Healthy`, atau kembalikan versi lama
8. Masuk sekali	halaman masuk	cookie, kunci, dan peran masih bekerja

Bila langkah 7 atau 8 gagal, hentikan layanan, salin kembali folder `publish` sebelumnya dan cadangan dari langkah 3, lalu nyalakan lagi: toko kembali seperti sepuluh menit sebelumnya, dan ada waktu untuk mencari kesalahannya.

6.9 Menjalankannya

Di komputer Anda, jalankan perintah-perintah di README bab ini: `dotnet restore`, `dotnet ef migrations add InitialCreate`, dan `dotnet run`, lalu buka `http://localhost:4060`.

1. Beranda, layanan, dan gerai terbuka tanpa masuk. Pilih **Counter**; peramban dikirim ke `/Account/Login?ReturnUrl=%2FCounter`.
2. Masuklah sebagai kasir dengan kata sandi yang salah lima kali; setelah percobaan kelima halaman melaporkan bahwa akunnya terkunci, dan selama 15 menit berikutnya kata sandi yang benar pun ditolak. Masuklah sebagai pemilik saja.
3. Sebagai pemilik, buka **Prices**, sunting *Ironing only*, hapus `(per kg)` dari deskripsinya dan atur harganya menjadi 60. Formulir kembali dengan dua pesan dari kasus 3 pada demo.

4. Tambahkan pelanggan dengan telepon `+62 812-0000-0104`; halaman pelanggan menampilkan `081200000104`. Tambahkan pelanggan lain dengan `0812 0000 1O4`; pesan dari `[IndonesianMobile]` muncul.
5. Buat pesanan dengan antar-jemput tanpa alamat; pesannya muncul di bawah isian alamat.
6. Keluar lalu masuk sebagai kasir. Kepala halaman tidak memiliki tautan Prices; ketik `/Prices` di bilah alamat dan halaman penolakan muncul. Coba hapus seorang pelanggan; ditolak juga.
7. Buka `http://localhost:4060/health` di jendela privat: `Healthy`, tanpa masuk.
8. Hentikan aplikasi lalu nyalakan lagi: Anda masih masuk, karena kunci cookie ada di folder `keys`.

Di tempat buku ini ditulis, NuGet tidak dapat dijangkau, sehingga aplikasi web tidak dapat di-*restore* maupun dijalankan di sana; kode C# dan *view* Razor-nya dikompilasi terhadap tiruan API EF Core dan Identity-EF yang khusus untuk kompilasi, dan sisanya (Identity itu sendiri, otorisasi, *data protection*, *health checks*) berasal dari *shared framework* dan dikompilasi sungguhan. `ValidationDemo` dibangun, dijalankan, dan diterbitkan, dan keluarannya di atas asli. Tur di atas mengikuti kode dan perilaku terdokumentasi setiap fitur.

6.10 Membuatnya Lebih Baik

- **Batasi juga percobaan masuk per alamat.** Penguncian melindungi satu akun; penyerang yang mencoba satu kata sandi pada banyak akun tidak diperlambat olehnya. *Middleware rate limiting* bawaan ASP.NET Core (`AddRateLimiter` dengan pembatas *fixed-window* pada *action* masuk) menambahkan batas per alamat IP.
- ***Passkey*.** ASP.NET Core 10 menambahkan dukungan *passkey* (WebAuthn) pada Identity, sehingga staf dapat masuk dengan pemindai sidik jari ponsel atau laptop alih-alih kata sandi [12]. Untuk meja kasir tempat beberapa orang berbagi satu komputer, kata sandi ditambah penguncian tetap merupakan awal yang praktis.
- **Catat siapa melakukan apa.** Dengan `User.Identity.Name` tersedia di setiap *action*, menambahkan kolom `CreatedBy` ke pesanan dan pembayaran hanyalah migrasi kecil, dan kolom itu menjawab pertanyaan "siapa yang memasukkan pembayaran ini?" yang cepat atau lambat ditanyakan setiap toko.
- **Pakai pengirim surel sungguhan untuk mengatur ulang kata sandi.** Identity dapat membangkitkan token pengaturan ulang kata sandi; mengirimkannya memerlukan implementasi `IEmailSender`, misalnya melalui penyedia surel toko. Sampai saat itu, pemilik mengatur ulang kata sandi kasir secara manual.

6.11 Memakainya Sungguhan

- **Peran Anda.** Toko dengan peran petugas cuci tersendiri menambahkan `Washer` ke `AppRoles`, memberi petugas cuci akses hanya ke daftar pesanan dan tombol *Mark finished*, serta melindungi *action* pembayaran dengan `[Authorize(Roles = AppRoles.Owner + "," + AppRoles.Cashier)]`.
- **Aturan Anda.** Letakkan aturan Anda di tempatnya: aturan tentang satu nilai sebagai atribut pada kelas formulir, aturan tentang beberapa nilai di `Validate`, aturan yang memerlukan basis data di *controller*. Tulis setiap pesan dengan kata-kata yang dipakai staf Anda di meja kasir.
- **Server Anda.** Mesin virtual awan kecil dengan memori 1 GB menjalankan aplikasi ini dengan lancar; begitu pula PC kantor lama dengan Linux di ruang belakang, dengan *tunnel* Cloudflare atau *router* toko yang meneruskan HTTPS kepadanya.

- **Data pribadi, sekali lagi.** Akun staf dan catatan pelanggan adalah data pribadi menurut UU PDP (UU No. 27 Tahun 2022). Kata sandi yang di-*hash*, penguncian, akses menurut peran, dan HTTPS adalah bagian dari "langkah teknis yang patut" yang diminta undang-undang itu; begitu pula cadangan yang juga dilindungi. Berikan akun sendiri kepada setiap orang, jangan pernah akun bersama, agar catatan siapa melakukan apa benar-benar bermakna.

6.12 Profitina dalam Praktik

Platform produksi Profitina telah melalui setiap langkah bab ini, dengan Python alih-alih C#. Sistem akunnya meng-*hash* kata sandi dengan Argon2id, algoritme yang diutamakan OWASP [3], meminta persetujuan eksplisit, membatasi percobaan masuk, dan memungkinkan setiap pengguna mengekspor datanya, mengubah alamat surel dan kata sandinya, serta menghapus akunnya sepenuhnya, yaitu cara platform itu memenuhi UU PDP 27/2022 secara swalayan. Host produksinya adalah mesin Windows 11 yang menjalankan aplikasi sebagai layanan di bawah NSSM (padanan Windows untuk unit systemd di Subbab 6.8.4) di belakang Cloudflare Tunnel yang berperan seperti Nginx dan memegang sertifikat. Dan proses rilisnya dijaga oleh pemeriksaan kesehatan: versi baru disiapkan, dinyalakan, dan baru dilaporkan telah rilis setelah menjawab `/api/health`, gagasan yang sama dengan *endpoint* `/health` di bab ini dan rutinitas pembaruan di Subbab 6.8.6.

6.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Memercayai peramban.** Aturan yang hanya diperiksa di JavaScript bukanlah aturan. Periksa semuanya di server; tambahkan pemeriksaan di sisi klien hanya untuk kenyamanan.
- **Lupa `if (!ModelState.IsValid)`.** Validasi mengisi `ModelState`, tetapi tidak menghentikan *action*. *Action* yang menyimpan tanpa bertanya menyimpan data yang tidak sah.
- **Mengikat formulir ke entitas.** Tanpa `[Bind]` atau kelas formulir, sebuah permintaan dapat mengisi properti apa pun pada entitas, termasuk harga dan status. Pakailah kelas formulir.
- **Menaruh `UseAuthorization()` sebelum `UseRouting()` atau `UseAuthentication()`.** Urutannya penting: *routing* memilih *endpoint*, autentikasi membaca cookie, otorisasi memeriksa aturan *endpoint*. Dengan urutan yang salah, aturan tidak melihat *endpoint* atau tidak melihat pengguna.
- **Melindungi halaman satu per satu.** Lupa memasang `[Authorize]` pada satu *controller* baru membiarkannya terbuka. Kebijakan cadangan melindungi semuanya secara bawaan, dan `[AllowAnonymous]` membuka halaman dengan sengaja.
- **Menyembunyikan tautan alih-alih melindungi halaman.** Tautan yang hilang tidak menghentikan siapa pun mengetik alamatnya. Atributlah yang melindungi; *layout* hanya merapikan.
- **Kata sandi demo di produksi.** Simpan akun awal di `appsettings.Development.json`, jangan pernah di `appsettings.json`, dan buat akun sungguhan sekali saja dari variabel lingkungan.
- **Kehilangan kunci cookie.** Tanpa kunci Data Protection yang disimpan, setiap penyalaan ulang mengeluarkan semua orang, dan dengan beberapa server setiap server menolak cookie dari server lain.
- **Lupa *forwarded headers* di belakang *proxy*.** Pengalihan lalu menunjuk ke `http://`, cookie *secure* berperilaku aneh, dan log menampilkan setiap pengunjung sebagai `127.0.0.1`.
- **Memperbarui tanpa cadangan.** Migrasi yang menghapus atau mengubah kolom tidak dapat dibatalkan oleh `Down` dengan cara yang mengembalikan data. Salin berkas basis data lebih dulu.

6.14 Ringkasan Bab

- Validasi berjalan di server untuk setiap permintaan; atribut *data annotation* menyatakan aturan tentang satu nilai, `IValidatableObject` aturan tentang beberapa nilai, dan *controller* menambahkan aturan yang memerlukan basis data. Semuanya berakhir di `ModelState`, yang ditampilkan oleh *view*.
- Kelas formulir membawa persis isian yang boleh dikirim formulir dan aturan formulir itu, sehingga *overposting* hilang sejak rancangannya.
- Autentikasi menetapkan siapa penggunanya; otorisasi menentukan apa yang boleh dilakukannya. ASP.NET Core Identity menyimpan pengguna di basis data aplikasi, meng-*hash* kata sandi dengan PBKDF2-HMAC-SHA512 bergaram pada 100.000 iterasi, mengunci kegagalan berulang, dan memasukkan pengguna dengan cookie terenkripsi.
- Kebijakan cadangan yang mewajibkan pengguna sudah masuk melindungi setiap halaman secara bawaan; `[AllowAnonymous]` membuka halaman publik, dan `[Authorize(Roles = …)]` membatasi halaman untuk peran tertentu.
- Penerapan berarti lingkungan dan konfigurasi berlapis, `dotnet publish`, migrasi yang diterapkan sebelum melayani, *reverse proxy* (Nginx atau IIS) di depan Kestrel dengan *forwarded headers*, kunci Data Protection yang disimpan, *endpoint* kesehatan, dan cadangan.

6.15 Soal Latihan

1. Untuk setiap masukan, sebutkan aturan yang menolaknya dan pesan yang ditampilkan formulir, atau bahwa masukan itu diterima dan bagaimana disimpan: (a) telepon `08-1200-0001-05`; (b) telepon `+6281200000106`; (c) deskripsi layanan `Carpet cleaning (per m2)` seharga Rp 25.000; (d) pesanan 0,05 kg; (e) pesanan dengan antar-jemput dan alamat `"   "` (tiga spasi).
2. Tulislah atribut validasi `[NotInFuture]` untuk properti `DateTime` yang menolak tanggal setelah hari ini. Pada tanggal pesanan yang mana atribut itu tepat: tanggal sebenarnya sebuah pesanan yang terlambat dicatat diterima, atau tanggal cucian dijanjikan selesai? Tambahkan properti itu ke `OrderForm`.
3. Pemilik ingin kasir dapat mengubah *catatan* pesanan tetapi tidak totalnya. Jelaskan bagaimana kelas formulir membuat hal ini aman, lalu tulis kelas formulir dan *action*-nya.
4. Gambarkan urutan *middleware* di `Program.cs` dari `UseForwardedHeaders` sampai `MapControllerRoute` dan jelaskan, untuk masing-masing, apa yang salah bila dipindahkan ke akhir.
5. Pemilik kedua bergabung dengan usaha ini. Tanpa mengubah kode apa pun, bagaimana Anda memberinya akun Owner di server produksi? Sebutkan langkah-langkahnya.
6. Disk server rusak pada Jumat malam. Daftarlah apa yang Anda perlukan untuk menghidupkan toko kembali di server baru pada Sabtu pagi, dan dari mana masing-masing berasal.

6.16 Penyelesaian Soal Latihan

1. (a) `Normalize` membuang tanda hubung: `081200000105`, 12 digit diawali `08`; diterima dan disimpan sebagai `081200000105`. (b) `+62` menjadi `0`: `081200000106`; diterima dan disimpan demikian. (c) Ekspresi reguler hanya mengizinkan `(per kg)`, `(per item)`, atau `(per pair)`, sehingga `[RegularExpression]` menolaknya dengan “End the description with (per kg), (per item) or (per pair).”; harganya berada dalam rentang. (d) `[Range(typeof(decimal), "0.1", "100")]` menolak 0,05 dengan “The quantity must be between 0.1 and 100.”. (e)

`string.IsNullOrWhiteSpace("   ")` bernilai `true`, sehingga `Validate` mengembalikan "Pickup and delivery needs a delivery address." di bawah isian alamat.

2. Atribut ini membandingkan nilai dengan tanggal hari ini dan, seperti `[IndonesianMobile]`, menyerahkan nilai kosong kepada `[Required]` (**Kode 6.20**).

Kode 6.20 Soal 6.2: `Models/NotInFutureAttribute.cs`, atribut validasi buatan sendiri untuk tanggal.

```
using System.ComponentModel.DataAnnotations;

namespace LaundryMvc.Models;

public sealed class NotInFutureAttribute : ValidationAttribute
{
    public NotInFutureAttribute() : base("{0} cannot be in the future.") { }

    public override bool IsValid(object? value) =>
        value is not DateTime date || date.Date <= DateTime.Today;
}
```

Atribut ini tepat untuk tanggal cucian benar-benar diterima, bagi kasir yang hari ini memasukkan nota kertas kemarin; tanggal janji selesai pada hakikatnya ada di masa depan dan memerlukan aturan kebalikannya. Di `OrderForm` tambahkan `[NotInFuture] [Display(Name = "Received on")] public DateTime? ReceivedAt { get; set; }`, dan di `Create` pakai `ReceivedAt = form.ReceivedAt ?? DateTime.Now`.

3. Kelas formulir yang hanya berisi `Note` (dengan `[StringLength(200)]`) berarti permintaan sama sekali tidak dapat membawa `Total`; *action* memuat pesanan yang dilacak, menyalin catatannya saja, lalu menyimpan. Pemilik tetap dapat mengubah total melalui halaman lain yang dilindungi.

Kode 6.21 Soal 6.3: kelas formulir dengan satu isian dan *action* yang memakainya, untuk ditambahkan ke `OrdersController`.

```
public class OrderNoteForm
{
    [StringLength(200)]
    public string? Note { get; set; }
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> EditNote(int id, OrderNoteForm form)
{
    var order = await db.Orders.FindAsync(id);
    if (order is null) return NotFound();
    if (!ModelState.IsValid) return RedirectToAction(nameof(Details), new { id });
    order.Note = form.Note;
    await db.SaveChangesAsync();
    return RedirectToAction(nameof(Details), new { id });
}
```

Update yang dikirim EF Core hanya berisi kolom `Note`, seperti di Subbab 5.10.4 (**Kode 6.21**).

4. `UseForwardedHeaders` → log pewaktuan → `UseExceptionHandler`/`UseHsts` → `UseStatusCodePagesWithReExecute` → `UseRouting` → `UseAuthentication` → `UseAuthorization` → `MapStaticAssets`/`MapHealthChecks`/`MapControllerRoute`. Bila dipindahkan ke akhir: *forwarded headers* baru diterapkan setelah semuanya telanjur melihat `http` dan `127.0.0.1`; log pewaktuan tidak mengukur apa pun; penangan *exception* tidak menangkap *exception* dari

middleware sebelumnya; halaman kode status melewatkan 404 yang dihasilkan sebelum mereka; *routing* setelah otorisasi membuat otorisasi tanpa *endpoint*, sehingga aturan atribut tidak berlaku; autentikasi setelah otorisasi membuat setiap pengguna tampak anonim, sehingga kebijakan cadangan mengirim semua orang ke halaman masuk. Pemetaan *endpoint* sudah berada di akhir, tempat yang semestinya.

5. Atur tiga variabel lingkungan untuk layanan itu sekali saja, `SeedUsers__0__Email=<surelnya>`, `SeedUsers__0__Password=<kata sandi sementara minimal 8 karakter>`, dan `SeedUsers__0__Role=Owner` (dengan `sudo systemctl edit laundrymvc`), nyalakan ulang layanannya dengan `sudo systemctl restart laundrymvc`, lalu hapus lagi variabel-variabel itu. `IdentitySeeder` membuat akun yang belum ada dan membiarkan akun yang sudah ada, sehingga pemilik pertama tidak tersentuh. Pemilik baru mengganti kata sandi sementaranya setelah masuk pertama kali (halaman ganti kata sandi adalah tambahan kecil dengan `UserManager.ChangePasswordAsync`).

6. (1) Aplikasi yang diterbitkan: bangun ulang dari repositori Git dengan `dotnet publish`, atau simpan folder terbitan terakhir bersama cadangan. (2) Basis data: cadangan malam terakhir `laundry.db`, yang disalin ke luar server. (3) Kunci cookie dari `/var/lib/laundrymvc/keys`: bila ada di cadangan, staf tetap masuk; bila tidak, semua orang masuk lagi, dan itu tidak berbahaya. (4) Konfigurasi: `appsettings.Production.json` ada di repositori; unit systemd dan situs Nginx ada di `deploy/`. (5) Sertifikat: `certbot` menerbitkan yang baru untuk domainnya. (6) Catatan DNS, diarahkan ke alamat server baru. Lalu pasang *runtime* dan Nginx seperti di Subbab 6.8.4, salin berkas-berkas dan basis datanya, nyalakan layanannya, dan periksa `/health`.

Daftar Pustaka

[1] Microsoft, “Model validation in ASP.NET Core MVC and Razor Pages,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/aspnet/core/mvc/models/validation

[2] Microsoft, “Introduction to Identity on ASP.NET Core,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/aspnet/core/security/authentication/identity

[3] OWASP Foundation, “Password Storage Cheat Sheet,” OWASP Cheat Sheet Series, diakses Sep. 2026. [Daring]. Tersedia: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

[4] Microsoft, “Policy-based authorization in ASP.NET Core,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/aspnet/core/security/authorization/policies

[5] Microsoft, “Configuration in ASP.NET Core,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/aspnet/core/fundamentals/configuration/

[6] Microsoft, “.NET application publishing overview,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/dotnet/core/deploying/

[7] Microsoft, “Applying migrations,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/ef/core/managing-schemas/migrations/applying

[8] Microsoft, “Host ASP.NET Core on Linux with Nginx,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/aspnet/core/host-and-deploy/linux-nginx

[9] Microsoft, “Configure ASP.NET Core to work with proxy servers and load balancers,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/aspnet/core/host-and-deploy/proxy-load-balancer

[10] Microsoft, “Host ASP.NET Core on Windows with IIS,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/aspnet/core/host-and-deploy/iis/

[11] Microsoft, “Key storage providers in ASP.NET Core,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/aspnet/core/security/data-protection/implementation/key-storage-providers

[12] Microsoft, “What’s new in ASP.NET Core in .NET 10,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/aspnet/core/release-notes/aspnetcore-10.0

Bab 7

Laravel I: Artisan, Routing, Blade, dan Eloquent

Bab 3, 5, dan 6 membangun Profitina Laundry dengan ASP.NET Core. Bab ini membangun toko yang sama sekali lagi dengan Laravel, kerangka kerja PHP yang paling banyak dipakai, agar Anda dapat meletakkan keduanya berdampingan dan melihat gagasan mana yang milik satu kerangka kerja dan mana yang milik semua kerangka kerja. Di akhir bab Anda akan mampu memasang PHP, Composer, dan Laravel 13 di Windows, macOS, atau Linux; mengenali isi proyek Laravel yang baru; memakai Artisan untuk membuat model, migrasi, dan *controller* serta memeriksa rute; menggambarkan tabel-tabel laundry dalam migrasi dan menjalankannya; menulis model Eloquent dengan relasi, *cast*, dan enum; mengisi basis data dari berkas CSV yang sama dengan Bab 5; mengarahkan permintaan dengan *resource route* dan *route model binding*; memvalidasi formulir di *controller*; serta membangun halaman dengan *layout*, *partial*, dan pembantu formulir Blade. Hasilnya adalah katalog publik Bab 3 dan halaman staf Bab 5 (meja kasir, pelanggan, harga, pesanan, dan pembayaran), dengan data dan aturan harga yang sama, dalam PHP.

7.1 Posisi Kita Sekarang: Toko yang Sama, Bahasa yang Lain

Pemilik Profitina Laundry memiliki keponakan, Raka, yang belajar informatika di perguruan tinggi dan mengenal PHP dari pelajaran pemrograman web. Ketika pemilik menyebut bahwa suatu hari gerai Wiyung mungkin menjalankan salinan halaman stafnya sendiri di *hosting* PHP murah yang sudah dibayarnya, Raka menawarkan diri membangunnya dengan Laravel. Pertanyaan pertamanya tepat: *berapa banyak dari yang sudah dibangun dengan C# yang harus saya ciptakan ulang?*

Jawaban bab ini: sedikit sekali pemikirannya, tetapi seluruh ketikannya. Basis datanya memiliki lima tabel yang sama. Halamannya adalah halaman yang sama. Alur `queued → finished → departed` dan aturan harga (minimum 3 kg untuk layanan per kg, barang dihitung utuh, biaya antar-jemput Rp 10.000 di bawah Rp 100.000) adalah aturan yang sama. Yang berubah adalah tempat tinggal setiap potongan, namanya, dan seberapa banyak yang dikerjakan kerangka kerja untuk Anda lewat konvensi. Mempelajari kerangka kerja kedua sebagian besar berarti mempelajari peta baru untuk wilayah lama, dan bab ini menggambar peta itu.

Bab ini membantu Raka, yang dapat memulai dari rancangan alih-alih dari nol; pemilik, yang mendapat implementasi kedua yang dapat di-*hosting* dengan murah; dan Anda, karena melihat fitur yang sama di dua kerangka kerja adalah cara tercepat memahami apa itu kerangka kerja. Bab 8 kemudian memecahkan tujuh soal yang sama di keduanya.

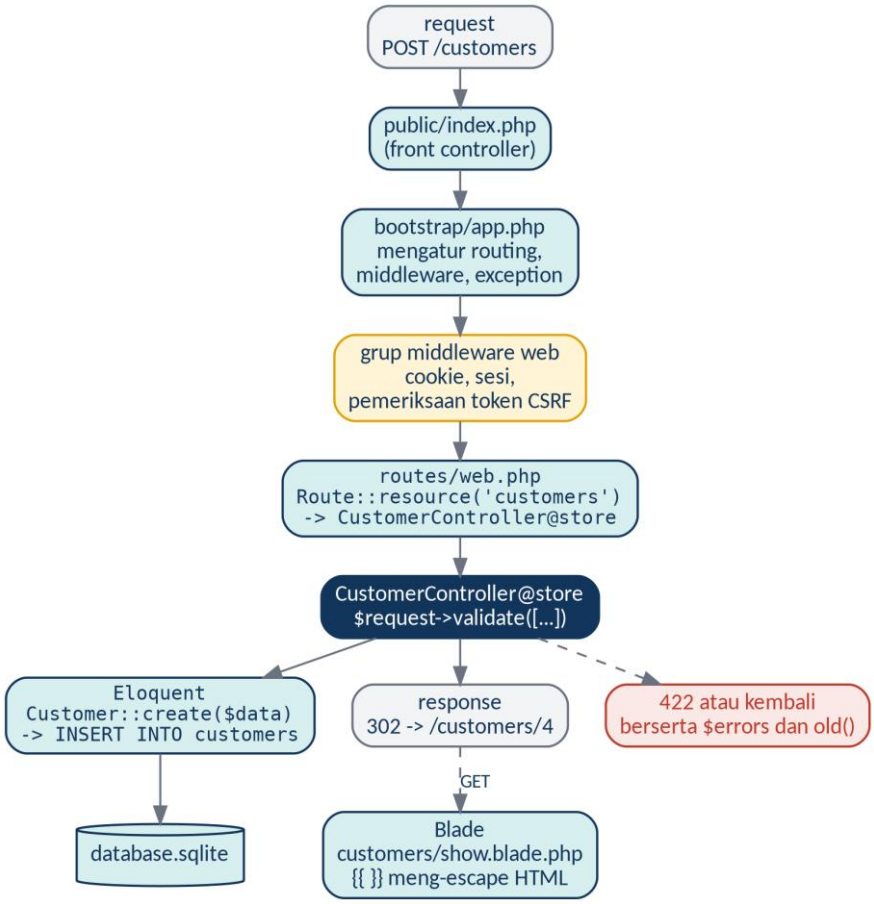
7.2 Gagasannya dengan Kata-Kata Sederhana

Laravel adalah kerangka kerja PHP sumber terbuka yang dibangun di atas pola MVC yang sama dengan ASP.NET Core MVC, dengan kecenderungan kuat pada *convention over configuration* (konvensi di atas konfigurasi): bila Anda menamai sesuatu seperti yang diharapkan Laravel, Laravel menghubungkannya tanpa diberi tahu. Model bernama `Customer` disimpan di tabel

bernama `customers`; kunci asing ke model itu bernama `customer_id`; metode *controller* bernama `store` menerima formulir dari `create`. Laravel 13, dirilis 17 Maret 2026, memerlukan PHP 8.3 atau lebih baru dan mendukung PHP sampai 8.5; versi ini menerima perbaikan *bug* sampai kuartal ketiga 2027 dan perbaikan keamanan sampai 17 Maret 2028 [1].

> **DEFINISI.** *Artisan* adalah alat baris perintah Laravel (`php artisan …`); ia membuat berkas, menjalankan migrasi, dan melayani aplikasi selama pengembangan. *Eloquent* adalah ORM Laravel; setiap kelas model mewakili satu tabel, dan setiap objek model satu baris yang dapat menyimpan dirinya sendiri. *Blade* adalah bahasa templat Laravel; berkasnya berakhiran `.blade.php`, dan `{{ … }}` mencetak nilai dengan *escaping* HTML. *Route* (rute) memetakan metode HTTP dan alamat ke metode *controller*; *resource route* menyatakan tujuh rute CRUD yang lazim untuk satu sumber daya dalam satu baris.

Gambar 7.1 mengikuti satu permintaan melalui aplikasi Laravel: kasir mengirim formulir pelanggan baru. Setiap permintaan masuk melalui `public/index.php`, satu-satunya *front controller*. `bootstrap/app.php` telah mengatur *routing*, *middleware*, dan penanganan *exception*. Grup *middleware* `web` memulai sesi, membaca cookie, dan memeriksa token CSRF formulir. *Router* menemukan rute `POST /customers` di `routes/web.php`, yang dibuat oleh `Route::resource('customers', …)`, lalu memanggil `CustomerController@store`. *Controller* memvalidasi masukan, Eloquent menyisipkan baris, dan responsnya adalah pengalihan ke halaman pelanggan baru, yang kemudian diminta peramban dengan `GET` dan dirender oleh Blade. Bila validasi gagal, Laravel mengalihkan kembali ke formulir beserta pesan galat dan masukan lama.



Gambar 7.1 Satu permintaan melalui aplikasi Laravel: *front controller*, grup *middleware* web, *router*, *controller*, Eloquent, dan Blade.

Bandingkan gambar itu dengan *pipeline* permintaan di Bab 6 (Subbab 6.2). Kotak-kotaknya bernama lain, tetapi urutannya sama: *middleware*, *routing*, *controller*, model, *view*. Perbedaan konseptual terbesar ada pada model. EF Core mengikuti pola *data mapper*: entitas adalah kelas biasa, dan `DbContext` yang terpisah memuat dan menyimpannya. Eloquent mengikuti pola *active record*: objek model mengenal tabelnya dan menyimpan dirinya sendiri, dengan `$customer->save()` atau `Customer::create([...])`. Keduanya ORM; keduanya meletakkan tanggung jawab di tempat yang berbeda.

7.3 Memasang PHP, Composer, dan Laravel

Laravel memerlukan tiga hal: PHP, Composer (pengelola paket PHP, padanan NuGet), dan basis data, yang secara bawaan adalah SQLite dan tidak perlu dipasang. Jalan paling sederhana di ketiga sistem operasi adalah pemasang di `php.new`, yang dianjurkan dokumentasi Laravel: satu perintah memasang PHP, Composer, dan pemasang Laravel [2]. Salin perintahnya dari situs itu, bukan dari buku, karena perintah itu menyebut versi PHP terkini:

- **macOS dan Linux:** satu perintah `curl … | bash`, disalin dari `php.new` atau dari panduan instalasi Laravel.
- **Windows:** satu perintah PowerShell, dijalankan sebagai administrator, disalin dari halaman yang sama.
- **Sistem apa pun, secara manual:** PHP dari `php.net` (Windows) atau dari pengelola paket sistem (`brew install php`, `sudo apt install php-cli php-sqlite3 php-mbstring php-xml php-curl`), lalu Composer dari `getcomposer.org`.

Buka terminal baru dan periksa dengan `php -v` dan `composer -V`. PHP 8.5, dirilis 20 November 2025, adalah versi terbaru, dengan operator *pipe* `|>`, ekstensi URI baru, dan banyak perbaikan kecil [3]; Laravel 13 berjalan di 8.3, 8.4, dan 8.5, sehingga PHP yang sudah disediakan distribusi Anda biasanya cukup. Di VS Code, pasang *PHP Intelephense* untuk PHP dan ekstensi resmi *Laravel* untuk dukungan Blade dan Artisan.

Buat proyeknya:

```
composer create-project laravel/laravel profitina-laundry "^13.0"
cd profitina-laundry
php artisan serve --port=4070
```

Composer mengunduh kerangka kerja beserta dependensinya ke `vendor/`, menulis `.env` dari `.env.example`, membangkitkan kunci aplikasi, membuat `database/database.sqlite`, dan menjalankan migrasi bawaan. `php artisan serve` menyalakan server pengembangan PHP; buka `http://localhost:4070` dan halaman sambutan Laravel muncul. Hentikan dengan Ctrl+C.

Tabel 7.1 adalah peta proyek baru itu, di samping proyek ASP.NET Core dari Bab 3 sampai 6.

Tabel 7.1 Tempat tinggal segala sesuatu dalam proyek Laravel 13, beserta padanannya di ASP.NET Core.

Laravel	Isinya	Di ASP.NET Core
`routes/web.php`	rute halaman web	`MapControllerRoute` dan atribut `[Route]`
`app/Http/Controllers/`	*controller*	`Controllers/`
`app/Models/`	model Eloquent	`Models/` (entitas)
`database/migrations/`	migrasi, ditulis tangan	`Migrations/`, dibangkitkan `dotnet ef`
`database/seeders/`	*seeder*	`SeedData` dengan `UseSeeding`
`resources/views/`	templat Blade	`Views/` (Razor)
`public/`	*front controller* dan berkas statis	`wwwroot/`
`bootstrap/app.php`	*routing*, *middleware*, *exception*	`Program.cs`
`.env`, `config/`	pengaturan	`appsettings*.json`
`composer.json`, `vendor/`	paket	`.csproj`, *cache* NuGet

Folder kode bab ini hanya berisi berkas yang ditambahkan atau diganti bab ini; salin berkas-berkas itu ke atas proyek baru seperti dijelaskan README. Kerangka kerjanya sendiri selalu berasal dari Composer.

7.4 Artisan, Baris Perintah yang Menulis Kode

Hampir setiap berkas di bab ini dapat dimulai dengan Artisan. **Tabel 7.2** mendaftar perintah yang dipakai.

Tabel 7.2 Perintah Artisan yang dipakai di bab ini.

Perintah	Kegunaannya
`php artisan make:model Customer -m`	model beserta migrasinya
`php artisan make:controller CustomerController --resource --model=Customer`	*controller* dengan tujuh metode CRUD
`php artisan make:migration add_rack_to_orders_table --table=orders`	migrasi yang mengubah tabel
`php artisan make:enum Enums/OrderStatus --string`	*backed enum*
`php artisan migrate`	menjalankan migrasi yang belum dijalankan
`php artisan migrate:fresh --seed`	menghapus semua tabel, bermigrasi lagi, dan mengisi data
`php artisan db:seed`	menjalankan `DatabaseSeeder`
`php artisan route:list --except-vendor`	rute-rute aplikasi
`php artisan tinker`	*shell* PHP interaktif di dalam aplikasi
`php artisan serve --port=4070`	server pengembangan

`tinker` layak disebut khusus. Ia adalah konsol tempat Anda dapat mengetik Eloquent secara langsung: `App\Models\Customer::count()`, `App\Models\Order::with('payments')->first()`. Ia padanan Laravel untuk `LinqDemo` di Bab 5, bedanya ia bekerja terhadap basis data sungguhan.

7.5 Migrasi: Tabel yang Ditulis dengan PHP

Di EF Core, migrasi dibangkitkan dari model. Di Laravel kebalikannya: Anda menulis migrasinya, dan model mengikuti tabel. Migrasi adalah kelas dengan metode `up()` dan `down()` yang memakai *schema builder*, `Schema::create` dengan `Blueprint` yang memiliki satu metode per tipe kolom [4]. Laravel menjalankan berkas migrasi menurut urutan namanya, yang diawali tanggal, dan mencatat masing-masing di tabel bernama `migrations`, bersama nomor *batch*.

Tabel pelanggan (**Kode 7.1**) memperlihatkan kosakata dasarnya: `id()` adalah kunci primer yang bertambah otomatis, `string('name', 80)` kolom teks paling panjang 80 karakter, `unique()` indeks unik, `nullable()` kolom yang boleh kosong, dan `timestamps()` dua kolom `created_at` dan `updated_at` yang diisi otomatis oleh Eloquent.

Kode 7.1 `2026_09_01_000001_create_customers_table.php`: tabel pelanggan.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('customers', function (Blueprint $table) {
            $table->id();
            $table->string('name', 80);
            $table->string('phone', 20)->unique();
            $table->string('address', 200)->nullable();
            $table->timestamps();
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('customers');
    }
};
```

Layanan dan gerai mempertahankan kodenya sebagai kunci primer, persis seperti di Bab 5, sehingga `string('id', 5)->primary()` menggantikan `id()` (**Kode 7.2**). Harga bertipe `unsignedInteger` dalam rupiah utuh; karena SQLite juga tidak memiliki tipe desimal, rupiah bilangan bulat adalah pilihan benar yang paling sederhana di kedua kerangka kerja.

Kode 7.2 `2026_09_01_000002_create_services_table.php`: layanan dan gerai, dengan kodenya sebagai kunci primer.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('services', function (Blueprint $table) {
            $table->string('id', 5)->primary();          // S0001, S0002, ...
            $table->string('description', 60);
            $table->unsignedInteger('price');            // rupiah per unit
            $table->boolean('is_active')->default(true);
            $table->timestamps();
        });

        Schema::create('outlets', function (Blueprint $table) {
            $table->string('id', 3)->primary();          // O01, O02, O03
            $table->string('name', 60);
            $table->string('address', 120);
            $table->string('phone', 20);
            $table->date('opened');
            $table->timestamps();
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('outlets');
        Schema::dropIfExists('services');
    }
};
```

Tabel pesanan (**Kode 7.3**) memegang relasi-relasinya. `foreignId('customer_id')->constrained()` membuat kolom bilangan bulat besar tak bertanda dan kunci asing ke `customers.id`, dengan nama tabel yang dibaca dari nama kolom menurut konvensi; `restrictOnDelete()` menolak menghapus pelanggan yang memiliki pesanan, aturan yang sama dengan `DeleteBehavior.Restrict` di Bab 5. Karena `service_id` merujuk kunci berupa teks, ia dinyatakan dalam dua langkah: kolomnya, lalu `foreign(...)->references('id')->on('services')`. Statusnya berupa untai dengan bawaan `'queued'` dan sebuah indeks, karena daftar pesanan menyaring menurut status.

Kode 7.3 `2026_09_01_000003_create_orders_table.php`: pesanan dengan kunci asing, status, dan cap waktunya.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('orders', function (Blueprint $table) {
            $table->id();
            $table->foreignId('customer_id')->constrained()->restrictOnDelete();
            $table->string('service_id', 5);
            $table->foreign('service_id')->references('id')->on('services')
                ->restrictOnDelete();
            $table->decimal('quantity', 6, 2);
            $table->boolean('pickup')->default(false);
            $table->string('delivery_address', 200)->nullable();
            $table->unsignedInteger('total');         // rupiah, fixed at creation
            $table->string('status', 12)->default('queued')->index();
            $table->string('note', 200)->nullable();
            $table->timestamp('received_at');
            $table->timestamp('finished_at')->nullable();
            $table->timestamp('departed_at')->nullable();
            $table->timestamps();
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('orders');
    }
};
```

Pembayaran (**Kode 7.4**) memakai `cascadeOnDelete()`: pembayaran tidak bermakna tanpa pesanannya.

Kode 7.4 `2026_09_01_000004_create_payments_table.php`: pembayaran, yang terhapus bersama pesanannya.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('payments', function (Blueprint $table) {
            $table->id();
            $table->foreignId('order_id')->constrained()->cascadeOnDelete();
            $table->unsignedInteger('amount');        // rupiah
            $table->string('method', 10);             // cash, qris, transfer
            $table->timestamp('paid_at');
            $table->timestamps();
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('payments');
    }
};
```

`php artisan migrate` menjalankan setiap migrasi yang belum ada di tabel `migrations`, sebagai satu *batch*. `php artisan migrate:rollback` menjalankan metode `down()` dari *batch* terakhir, dimulai dari yang terbaru. Bila EF Core membandingkan model dengan *snapshot* untuk menentukan isi migrasi baru, Laravel tidak membandingkan apa pun: berkas yang Anda tulis itulah perubahannya. Hal itu membuat migrasi Laravel mudah dibaca dan mudah ditulis untuk basis data mana pun, sekaligus menaruh tanggung jawab menjaga model dan tabel tetap selaras di tangan Anda. `php artisan migrate:status` mendaftar setiap berkas migrasi dengan keterangan *Ran* atau *Pending*, padanan tabel `__EFMigrationsHistory` milik EF Core. `php artisan migrate:fresh` menghapus setiap tabel dan menjalankan semuanya lagi, yang Anda perlukan di komputer sendiri selama rancangan masih berubah, dan tidak pernah di server toko.

7.6 Model Eloquent

7.6.1 Enum untuk status dan cara bayar

PHP memiliki enumerasi sungguhan sejak versi 8.1 [5]. *Backed enum* memiliki nilai tetap untuk setiap *case*, dan nilai itulah yang disimpan di basis data. **Kode 7.5** adalah status pesanan, dengan `label()` untuk halaman dan metode `next()` yang menggambarkan alur kerja dalam satu ekspresi `match`; **Kode 7.6** adalah cara bayar.

Kode 7.5 `app/Enums/OrderStatus.php`: tiga status pesanan dan alur kerja di antaranya.

```php
<?php

namespace App\Enums;

enum OrderStatus: string
{
    case Queued = 'queued';
    case Finished = 'finished';
    case Departed = 'departed';

    public function label(): string
    {
        return ucfirst($this->value);
    }

    // The status that follows this one, or null when the order has left.
    public function next(): ?self
    {
        return match ($this) {
            self::Queued => self::Finished,
            self::Finished => self::Departed,
            self::Departed => null,
        };
    }
}
```

Kode 7.6 `app/Enums/PaymentMethod.php`: tunai, QRIS, dan transfer.

```php
<?php

namespace App\Enums;

enum PaymentMethod: string
{
    case Cash = 'cash';
    case Qris = 'qris';
    case Transfer = 'transfer';

    public function label(): string
    {
        return match ($this) {
            self::Cash => 'Cash',
            self::Qris => 'QRIS',
            self::Transfer => 'Transfer',
        };
    }
}
```

Uang diformat di satu tempat, `App\Support\Money` (**Kode 7.7**), yang mencetak teks yang sama dengan `Money.Rp` pada versi C#: `Rp 36.000`, dengan titik di antara ribuan.

Kode 7.7 `app/Support/Money.php`: rupiah yang diformat ala Indonesia.

```php
<?php

namespace App\Support;

class Money
{
    // 36000 -> "Rp 36.000", the same text as Money.Rp in the C# version.
    public static function rp(int $amount): string
    {
        return 'Rp ' . number_format($amount, 0, ',', '.');
    }
}
```

Enum dan pembantu uang adalah PHP biasa, sehingga dapat dijalankan tanpa Laravel. `tools/enum_demo.php` (**Kode 7.8**) menelusuri alur kerja dan memperlihatkan apa yang akan dilakukan *cast* enum Eloquent terhadap teks yang tersimpan di basis data.

Kode 7.8 `tools/enum_demo.php`: enum dan pembantu uang, dijalankan dengan PHP biasa.

```php
<?php
// Runs with plain PHP (no Laravel needed): php tools/enum_demo.php

require __DIR__ . '/../app/Enums/OrderStatus.php';
require __DIR__ . '/../app/Enums/PaymentMethod.php';
require __DIR__ . '/../app/Support/Money.php';

use App\Enums\OrderStatus;
use App\Enums\PaymentMethod;
use App\Support\Money;

$status = OrderStatus::Queued;
while ($status !== null) {
    $next = $status->next();
    printf("%-9s -> %s\n", $status->label(), $next?->label() ?? '(end)');
    $status = $next;
}

// What Eloquent's cast does with the text stored in the database:
var_dump(OrderStatus::from('finished'));
var_dump(OrderStatus::tryFrom('washing'));        // unknown text -> null
echo PaymentMethod::Qris->label(), ' is stored as "',
    PaymentMethod::Qris->value, "\"\n";
echo Money::rp(36000), ' ', Money::rp(1250000), "\n";
```

Inilah keluaran aslinya (`php tools/enum_demo.php`):

```
Queued    -> Finished
Finished  -> Departed
Departed  -> (end)
enum(App\Enums\OrderStatus::Finished)
NULL
QRIS is stored as "qris"
Rp 36.000 Rp 1.250.000
```

`OrderStatus::from('finished')` mengubah teks tersimpan menjadi *case* enum, dan `tryFrom('washing')` mengembalikan `null` untuk teks yang bukan status alih-alih melempar *exception*. Daftar pesanan memakai `tryFrom` justru karena alasan itu: `/orders?status=nonsense` menampilkan semua pesanan, bukan halaman galat.

7.6.2 Model-modelnya

Model Eloquent adalah kelas turunan `Model`; tabel, kunci, dan cap waktunya mengikuti konvensi kecuali kelas itu menyatakan lain [6]. **Kode 7.9** adalah model yang paling sederhana. `#[Fillable(['name', 'phone', 'address'])]` mendaftar atribut yang boleh diisi secara *mass assignment* (penugasan massal), yaitu diisi dari sebuah larik seperti pada `Customer::create($data)`. Selain itu semuanya ditolak, dan itulah perlindungan Laravel terhadap serangan *overposting* di Bab 5. Sintaks atribut ini baru di Laravel 13; properti lama `protected $fillable = [...]` tetap bekerja dan bermakna sama [7]. Metode `orders()` menyatakan relasinya: seorang pelanggan *has many* (memiliki banyak) pesanan, ditemukan melalui kolom `customer_id` menurut konvensi [8].

Kode 7.9 `app/Models/Customer.php`: seorang pelanggan dan relasi ke pesanan-pesanannya.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Attributes\Fillable;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasMany;

#[Fillable(['name', 'phone', 'address'])]
class Customer extends Model
{
    public function orders(): HasMany
    {
        return $this->hasMany(Order::class);
    }
}
```

Model layanan (**Kode 7.10**) memiliki tiga rincian. `$keyType = 'string'` dan `$incrementing = false` memberi tahu Eloquent bahwa kuncinya adalah kode seperti `S0001` yang dipilih aplikasi, bukan angka yang dihitung basis data. Metode `casts()` mengubah harga menjadi bilangan bulat dan `is_active` dari `0`/`1` milik SQLite menjadi *boolean* sungguhan. Dan tiga *accessor* `name`, `perKg`, dan `unit` menghitung nilai yang sama dengan properti terhitung pada kelas `Service` versi C#, dari deskripsinya. *Accessor* adalah metode yang mengembalikan `Attribute`; Eloquent memanggilnya ketika halaman membaca `$service->name`, `$service->per_kg`, atau `$service->unit` (metode `perKg` menjadi atribut `per_kg`, menurut konvensi).

Kode 7.10 `app/Models/Service.php`: kunci berupa untai, *cast*, dan tiga atribut terhitung.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Attributes\Fillable;
use Illuminate\Database\Eloquent\Casts\Attribute;
use Illuminate\Database\Eloquent\Model;

#[Fillable(['id', 'description', 'price', 'is_active'])]
class Service extends Model
{
    // The key is a code such as "S0001", not a number the database counts.
    protected $keyType = 'string';

    public $incrementing = false;

    protected function casts(): array
    {
        return [
            'price' => 'integer',             // whole rupiah
            'is_active' => 'boolean',
        ];
    }

    // "Wash, dry & fold (per kg)" -> "Wash, dry & fold"
    protected function name(): Attribute
    {
        return Attribute::get(fn () => str_contains($this->description, ' (')
            ? substr($this->description, 0, strrpos($this->description, ' ('))
            : $this->description);
    }

    protected function perKg(): Attribute
    {
        return Attribute::get(
            fn () => str_ends_with($this->description, '(per kg)'));
    }
```

Kode 7.10 (Lanjutan) *app/Models/Service.php: kunci berupa untai, cast, dan tiga atribut terhitung.*

```php
    protected function unit(): Attribute
    {
        return Attribute::get(fn () => match (true) {
            $this->per_kg => 'kg',
            str_ends_with($this->description, '(per pair)') => 'pair',
            default => 'item',
        });
    }
}
```

Model pesanan (**Kode 7.11**) mengubah kolom-kolomnya menjadi tipe PHP yang tepat: jumlah menjadi untai desimal dengan dua digit, `pickup` menjadi *boolean*, status menjadi enum `OrderStatus`, dan tiga cap waktu menjadi objek tanggal `Carbon` yang dapat diformat *view*. `belongsTo` menyatakan sisi lain relasi satu-ke-banyak. `paid()` dan `balance()` menghitung rumus sisa tagihan Bab 5 dari pembayaran yang telah dimuat.

Kode 7.11 `app/Models/Order.php`: *cast*, relasi, dan sisa tagihan sebuah pesanan.

```php
<?php

namespace App\Models;

use App\Enums\OrderStatus;
use Illuminate\Database\Eloquent\Attributes\Fillable;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;

#[Fillable(['customer_id', 'service_id', 'quantity', 'pickup',
    'delivery_address', 'total', 'status', 'note', 'received_at',
    'finished_at', 'departed_at'])]
class Order extends Model
{
    protected function casts(): array
    {
        return [
            'quantity' => 'decimal:2',
            'pickup' => 'boolean',
            'total' => 'integer',
            'status' => OrderStatus::class,
            'received_at' => 'datetime',
            'finished_at' => 'datetime',
            'departed_at' => 'datetime',
        ];
    }

    public function customer(): BelongsTo
    {
        return $this->belongsTo(Customer::class);
    }

    public function service(): BelongsTo
    {
        return $this->belongsTo(Service::class);
    }

    public function payments(): HasMany
    {
        return $this->hasMany(Payment::class);
    }
```

Kode 7.11 (Lanjutan) *app/Models/Order.php: cast, relasi, dan sisa tagihan sebuah pesanan.*

```php
    // Sums the loaded payments; load them with with('payments') first.
    public function paid(): int
    {
        return (int) $this->payments->sum('amount');
    }

    public function balance(): int
    {
        return $this->total - $this->paid();
    }
}
```

Sebuah pembayaran (**Kode 7.12**) milik satu pesanan dan meng-*cast* cara bayarnya ke enum `PaymentMethod`.

Kode 7.12 `app/Models/Payment.php`: sebuah pembayaran dan pesanannya.

```php
<?php

namespace App\Models;

use App\Enums\PaymentMethod;
use Illuminate\Database\Eloquent\Attributes\Fillable;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

#[Fillable(['order_id', 'amount', 'method', 'paid_at'])]
class Payment extends Model
{
    protected function casts(): array
    {
        return [
            'amount' => 'integer',
            'method' => PaymentMethod::class,
            'paid_at' => 'datetime',
        ];
    }

    public function order(): BelongsTo
    {
        return $this->belongsTo(Order::class);
    }
}
```

Model gerai (**Kode 7.13**) melengkapi rangkaiannya; kolom `opened`-nya di-*cast* menjadi tanggal.

Kode 7.13 `app/Models/Outlet.php`: tiga gerai dari Bab 3.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Attributes\Fillable;
use Illuminate\Database\Eloquent\Model;

#[Fillable(['id', 'name', 'address', 'phone', 'opened'])]
class Outlet extends Model
{
    protected $keyType = 'string';

    public $incrementing = false;

    protected function casts(): array
    {
        return ['opened' => 'date'];
    }
}
```

Ada dua hal yang tidak ada dibandingkan EF Core, dan keduanya disengaja. Tidak ada daftar kolom: Eloquent membaca kolom apa pun yang dimiliki tabel, sehingga `$customer->phone` bekerja karena migrasinya membuat `phone`. Dan tidak ada objek konteks: `Customer::find(3)` bertanya langsung ke basis data melalui koneksi bawaan. Harga dari kemudahan itu adalah salah ketik seperti `$customer->phnoe` diam-diam mengembalikan `null` alih-alih gagal dikompilasi.

7.6.3 Eloquent dan EF Core berdampingan

Tabel 7.3 menempatkan operasi sehari-hari kedua ORM berdampingan, memakai *query* milik laundry sendiri. Membacanya baris demi baris adalah cara tercepat membawa yang Anda pelajari di Bab 5 ke Laravel.

Tabel 7.3 Operasi sehari-hari di EF Core dan di Eloquent.

Tugas	EF Core (Bab 5)	Eloquent (bab ini)
satu baris menurut kunci	`await db.Customers.FindAsync(3)`	`Customer::find(3)`
satu baris atau 404	`FirstOrDefaultAsync(...)` lalu `NotFound()`	*route model binding*, atau `findOrFail(3)`
saring dan urutkan	`db.Customers.Where(...).OrderBy(c => c.Name)`	`Customer::where(...)->orderBy('name')`
jalankan *query*	`await query.ToListAsync()`	`$query->get()`
baris terkait	`.Include(o => o.Customer)`	`->with('customer')`
baris terkait belakangan	(pemuatan eksplisit) `db.Entry(o).Collection(...).LoadAsync()`	`$order->load('payments')`
hitung, jumlahkan	`CountAsync()`, `SumAsync(p => p.Amount)`	`count()`, `sum('amount')`
sisipkan	`db.Customers.Add(c); await db.SaveChangesAsync()`	`Customer::create($data)`
perbarui	ubah entitas terlacak, lalu `SaveChangesAsync()`	`$customer->update($data)` atau ubah lalu `save()`
hapus	`db.Customers.Remove(c); await db.SaveChangesAsync()`	`$customer->delete()`
ada atau tidak	`AnyAsync(o => o.CustomerId == id)`	`$customer->orders()->exists()`

Tiga perbedaan menonjol. Pertama, Eloquent tidak memiliki langkah "simpan semua perubahan" yang terpisah: `create`, `update`, `save`, dan `delete` masing-masing langsung mengirim SQL-nya, sedangkan EF Core mengumpulkan perubahan di pelacak perubahan dan mengirimkannya bersama dalam satu transaksi ketika `SaveChangesAsync` dipanggil. Bila beberapa baris harus berubah bersama di Laravel, bungkus dengan `DB::transaction(function () { … })`. Kedua, pemanggilan Eloquent bersifat sinkron; PHP melayani setiap permintaan dalam prosesnya sendiri, sehingga tidak ada *thread* yang perlu dibebaskan selama basis data bekerja, dan tidak ada `async` maupun `await`. Ketiga, EF Core memeriksa nama properti saat kompilasi, sedangkan Eloquent mencari nama kolom saat berjalan; kompilator menangkap `c.Phnoe`, tetapi hanya pengujian, atau pengguna, yang menangkap `$c->phnoe`.

7.7 Mengisi Data dari Berkas Bab 3

Seeder-nya (**Kode 7.14**) membuat data yang sama dengan kelas `SeedData` di Bab 5: dua belas layanan dan tiga gerai dari berkas CSV Bab 3 itu sendiri, yang disalin ke `database/data/`, dan tiga pelanggan contoh beserta pesanan dan pembayarannya. `$sari->orders()->create([...])` membuat pesanan dan mengisi `customer_id` dari relasinya; `$order->payments()->create([...])` melakukan hal yang sama untuk pembayaran.

Kode 7.14 `database/seeders/DatabaseSeeder.php`: layanan dan gerai dari berkas CSV Bab 3, lalu pelanggan, pesanan, dan pembayaran contoh.

```php
<?php

namespace Database\Seeders;

use App\Enums\OrderStatus;
use App\Enums\PaymentMethod;
use App\Models\Customer;
use App\Models\Outlet;
use App\Models\Service;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    // The same starting data as the ASP.NET Core version of Chapter 5.
    public function run(): void
    {
        foreach ($this->csv('Service.csv') as [$id, $description, $price]) {
            Service::create(['id' => $id, 'description' => $description,
                'price' => (int) $price]);
        }
        foreach ($this->csv('Outlet.csv') as [$id, $name, $address, $phone,
            $opened]) {
            Outlet::create(compact('id', 'name', 'address', 'phone', 'opened'));
        }

        $today = today();

        $sari = Customer::create(['name' => 'Sari Wulandari',
            'phone' => '081200000101',
            'address' => 'Jl. Kertajaya 12, Surabaya']);
        $order = $sari->orders()->create(['service_id' => 'S0001',
            'quantity' => 4.5, 'total' => 36000,
            'status' => OrderStatus::Finished,
            'received_at' => $today->copy()->subDay()->addHours(8),
            'finished_at' => $today->copy()->addHours(9)]);
        $order->payments()->create(['amount' => 36000,
            'method' => PaymentMethod::Qris,
            'paid_at' => $today->copy()->subDay()->addHours(8)]);
```

Kode 7.14 (Lanjutan) *database/seeders/DatabaseSeeder.php: layanan dan gerai dari berkas CSV Bab 3, lalu pelanggan, pesanan, dan pembayaran contoh.*

```
        $budi = Customer::create(['name' => 'Budi Santoso',
            'phone' => '081200000102']);
        $order = $budi->orders()->create(['service_id' => 'S0002',
            'quantity' => 3, 'total' => 36000,
            'received_at' => $today->copy()->addMinutes(450)]);
        $order->payments()->create(['amount' => 20000,
            'method' => PaymentMethod::Cash,
            'paid_at' => $today->copy()->addMinutes(450)]);
        $budi->orders()->create(['service_id' => 'S0008', 'quantity' => 2,
            'pickup' => true, 'delivery_address' => 'Jl. Mulyosari 30, Surabaya',
            'total' => 80000, 'received_at' => $today->copy()->addHours(10)]);

        Customer::create(['name' => 'Dewi Lestari',
            'phone' => '081200000103', 'address' => 'Jl. Manyar 7, Surabaya']);
    }

    // The rows of database/data/<file>, without the header line.
    private function csv(string $file): array
    {
        $lines = file(database_path("data/{$file}"), FILE_IGNORE_NEW_LINES);
        $rows = array_map(fn ($line) => str_getcsv($line, ',', '"', ''), $lines);

        return array_slice($rows, 1);
    }
}
```

`str_getcsv($line, ',', '"', '')` sengaja memberikan keempat argumennya: PHP 8.4 menandai usang kebiasaan mengandalkan nilai bawaan argumen keempat, karakter *escape*, dan memberikan `''` mematikan cara *escape* lama yang tidak baku itu. `database_path()` adalah pembantu Laravel yang mengembalikan jalur absolut folder `database`, sehingga *seeder* bekerja dari folder mana pun Artisan dijalankan. `php artisan migrate:fresh --seed` membangun basis data dari nol dengan satu perintah.

7.8 *Routing*

Kode 7.15 adalah `routes/web.php` selengkapnya. Bandingkan dengan *routing* di Bab 3: di sana, rute konvensional dan atribut pada *controller*; di sini, setiap rute aplikasi dalam satu berkas.

Kode 7.15 `routes/web.php`: halaman publik, halaman staf, dan dua *action* tambahan pada pesanan.

```php
<?php

use App\Http\Controllers\CounterController;
use App\Http\Controllers\CustomerController;
use App\Http\Controllers\HomeController;
use App\Http\Controllers\OrderController;
use App\Http\Controllers\OutletController;
use App\Http\Controllers\PaymentController;
use App\Http\Controllers\PriceController;
use App\Http\Controllers\ServiceController;
use Illuminate\Support\Facades\Route;

// The public pages (Chapter 3 built them in ASP.NET Core).
Route::get('/', HomeController::class)->name('home');
Route::get('services', [ServiceController::class, 'index'])
    ->name('services.index');
Route::get('services/{service}', [ServiceController::class, 'show'])
    ->name('services.show');
Route::get('outlets', [OutletController::class, 'index'])
    ->name('outlets.index');

// The staff pages (Chapter 5 built them in ASP.NET Core).
Route::get('counter', CounterController::class)->name('counter');
Route::resource('customers', CustomerController::class);
Route::resource('prices', PriceController::class)
    ->parameters(['prices' => 'service'])
    ->except(['show', 'destroy']);
Route::resource('orders', OrderController::class)
    ->only(['index', 'create', 'store', 'show']);
Route::post('orders/{order}/advance', [OrderController::class, 'advance'])
    ->name('orders.advance');
Route::post('orders/{order}/payments', [PaymentController::class, 'store'])
    ->name('orders.payments.store');
```

`Route::get('services/{service}', …)` memiliki parameter rute bernama `service`, dan metode *controller*-nya menyatakan `public function show(Service $service)`. Karena nama dan tipenya cocok, Laravel melakukan *route model binding*: mencari layanan yang kuncinya sama dengan nilai di alamat dan memberikan modelnya, atau menjawab 404 bila tidak ada [9]. Ia padanan `FirstOrDefaultAsync(...)` yang diikuti `NotFound()` di setiap *action* ASP.NET Core, dikerjakan sekali oleh kerangka kerja.

`Route::resource('customers', CustomerController::class)` menyatakan tujuh rute sekaligus; **Tabel 7.4** mendaftarnya [10]. *Resource route* memakai metode HTTP sebagaimana mestinya: alamat yang sama `/customers/3` menampilkan pelanggan dengan `GET`, memperbaruinya dengan `PUT`, dan menghapusnya dengan `DELETE`. Karena formulir HTML hanya dapat mengirim `GET` dan `POST`, `@method('PUT')` milik Blade menambahkan isian tersembunyi `_method=PUT` yang dibaca Laravel.

Tabel 7.4 Tujuh rute dari `Route::resource('customers', …)`.

Metode dan alamat	Metode *controller*	Nama rute	Kegunaan
`GET /customers`	`index`	`customers.index`	daftar
`GET /customers/create`	`create`	`customers.create`	formulir kosong
`POST /customers`	`store`	`customers.store`	menyimpan pelanggan baru
`GET /customers/{customer}`	`show`	`customers.show`	satu pelanggan
`GET /customers/{customer}/edit`	`edit`	`customers.edit`	formulir terisi
`PUT /customers/{customer}`	`update`	`customers.update`	menyimpan perubahan
`DELETE /customers/{customer}`	`destroy`	`customers.destroy`	menghapus

`->except([...])` dan `->only([...])` memangkas rangkaian itu: harga tidak memiliki `show` maupun `destroy` (layanan dimatikan, tidak pernah dihapus, seperti di Bab 5), dan pesanan tidak memiliki halaman ubah maupun hapus. `->parameters(['prices' => 'service'])` mengganti nama parameter rute harga dari `{price}` menjadi `{service}`, agar *route model binding* menemukan sebuah `Service`. Dua rute tambahan untuk memajukan pesanan dan mencatat pembayaran mendapat nama sendiri dengan `->name(...)`. Setiap halaman membangun tautannya dengan `route('customers.show', $customer)` alih-alih menulis `/customers/3`, sehingga sebuah alamat dapat berubah di `web.php` tanpa merusak satu tautan pun.

7.9 *Controller*

7.9.1 Pelanggan: validasi dan CRUD

Kode 7.16 adalah *controller* pelanggan. Bacalah di samping `CustomersController` dari Bab 5: delapan *action* C# menjadi tujuh metode PHP, karena satu `destroy` milik Laravel menggantikan halaman konfirmasi ditambah penghapusannya.

Kode 7.16 `app/Http/Controllers/CustomerController.php`: CRUD lengkap untuk pelanggan dengan Eloquent.

```php
<?php

namespace App\Http\Controllers;

use App\Models\Customer;
use Illuminate\Http\Request;
use Illuminate\Validation\Rule;

class CustomerController extends Controller
{
    // GET /customers?search=sari
    public function index(Request $request)
    {
        $search = trim((string) $request->query('search', ''));
        $customers = Customer::query()
            ->when($search !== '', fn ($query) => $query
                ->where('name', 'like', "%{$search}%")
                ->orWhere('phone', 'like', "%{$search}%"))
            ->orderBy('name')
            ->get();

        return view('customers.index', compact('customers', 'search'));
    }

    // GET /customers/3   (route model binding finds Customer 3, or 404)
    public function show(Customer $customer)
    {
        $customer->load(['orders' => fn ($q) => $q->latest('received_at'),
            'orders.service']);

        return view('customers.show', compact('customer'));
    }

    public function create()
    {
        return view('customers.create', ['customer' => new Customer()]);
    }

    public function store(Request $request)
    {
        $customer = Customer::create($request->validate($this->rules()));

        return redirect()->route('customers.show', $customer)
            ->with('message', "Customer {$customer->name} added.");
    }
```

Kode 7.16 (Lanjutan) *app/Http/Controllers/CustomerController.php: CRUD lengkap untuk pelanggan dengan Eloquent.*

```
    public function edit(Customer $customer)
    {
        return view('customers.edit', compact('customer'));
    }

    public function update(Request $request, Customer $customer)
    {
        $customer->update($request->validate($this->rules($customer)));

        return redirect()->route('customers.show', $customer)
            ->with('message', "Customer {$customer->name} saved.");
    }

    public function destroy(Customer $customer)
    {
        if ($customer->orders()->exists()) {
            return back()->with('error',
                'A customer with orders cannot be deleted.');
        }
        $customer->delete();

        return redirect()->route('customers.index')
            ->with('message', "Customer {$customer->name} deleted.");
    }

    private function rules(?Customer $customer = null): array
    {
        return [
            'name' => ['required', 'string', 'min:2', 'max:80'],
            'phone' => ['required', 'string', 'max:20',
                Rule::unique('customers', 'phone')->ignore($customer)],
            'address' => ['nullable', 'string', 'max:200'],
        ];
    }
}
```

- `index` membangun *query* dengan `Customer::query()`, menambahkan pencarian dengan `when($search !== '', …)`, yang menerapkan *closure* hanya bila syaratnya benar, dan diakhiri `get()`, yang mengirim SQL. `where('name', 'like', "%{$search}%")` dikirim dengan teks pencarian sebagai parameter terikat, sehingga sama amannya dari *SQL injection* seperti `Contains` milik EF Core. Berbeda dengan `instr`, `LIKE` milik SQLite mengabaikan perbedaan huruf besar dan kecil untuk huruf ASCII, sehingga `sari` menemukan *Sari Wulandari*.
- `show` menerima pelanggan melalui *route model binding* dan memuat pesanan-pesanannya, yang terbaru lebih dulu, beserta layanan setiap pesanan, dengan `load`, padanan `Include` dan `ThenInclude`.
- `store` dan `update` memanggil `$request->validate($rules)`. Bila sebuah aturan gagal, Laravel menghentikan metodenya, mengalihkan kembali ke formulir, dan menaruh pesan galat serta nilai yang dikirim ke sesi, tempat Blade menemukannya dengan `@error` dan `old()`. Bila semua aturan lolos, `validate` hanya mengembalikan isian yang sudah divalidasi, yang langsung masuk ke `create` atau `update` [11].
- Aturan telepon `Rule::unique('customers', 'phone')->ignore($customer)` memeriksa basis data untuk pelanggan lain dengan nomor yang sama, dengan mengabaikan pelanggan yang sedang diubah: seluruh metode `CheckPhoneIsFreeAsync` dari Bab 5 dalam satu baris.
- `destroy` menolak menghapus pelanggan yang memiliki pesanan, persis seperti di Bab 5, dan memakai `back()->with('error', …)` untuk menjelaskan alasannya. `with` menaruh pesan *flash* ke sesi hanya untuk permintaan berikutnya, padanan `TempData`.

7.9.2 Pesanan, pembayaran, dan harga

Controller pesanan (**Kode 7.17**) mengikuti `OrdersController` Bab 5 dengan dekat.

Kode 7.17 `app/Http/Controllers/OrderController.php`: daftar pesanan, halaman pesanan, pesanan baru yang dihargai dengan aturan Bab 3, dan alur kerjanya.

```php
<?php

namespace App\Http\Controllers;

use App\Enums\OrderStatus;
use App\Models\Customer;
use App\Models\Order;
use App\Models\Service;
use App\Support\Money;
use Illuminate\Http\Request;
use Illuminate\Validation\Rule;

class OrderController extends Controller
{
    // GET /orders or /orders?status=queued
    public function index(Request $request)
    {
        $status = OrderStatus::tryFrom((string) $request->query('status'));
        $orders = Order::with(['customer', 'service', 'payments'])
            ->when($status, fn ($query) => $query->where('status', $status))
            ->latest('received_at')
            ->get();

        return view('orders.index', compact('orders', 'status'));
    }

    public function show(Order $order)
    {
        $order->load(['customer', 'service',
            'payments' => fn ($q) => $q->orderBy('paid_at')]);

        return view('orders.show', compact('order'));
    }

    public function create(Request $request)
    {
        return view('orders.create', [
            'customers' => Customer::orderBy('name')->get(),
            'services' => Service::where('is_active', true)
                ->orderBy('id')->get(),
            'customerId' => $request->integer('customer_id'),
        ]);
    }
```

Kode 7.17 (Lanjutan) *app/Http/Controllers/OrderController.php: daftar pesanan, halaman pesanan, pesanan baru yang dihargai dengan aturan Bab 3, dan alur kerjanya.*

```
    public function store(Request $request)
    {
        $data = $request->validate([
            'customer_id' => ['required', 'integer', 'exists:customers,id'],
            'service_id' => ['required', 'string',
                Rule::exists('services', 'id')->where('is_active', true)],
            'quantity' => ['required', 'numeric', 'min:0.1', 'max:100'],
            'delivery_address' => ['nullable', 'required_if_accepted:pickup',
                'string', 'max:200'],
            'note' => ['nullable', 'string', 'max:200'],
        ]);
        $service = Service::findOrFail($data['service_id']);
        $pickup = $request->boolean('pickup');

        // The price rules of Chapter 3: minimum 3 kg, whole items, pickup fee.
        $quantity = (float) $data['quantity'];
        $billed = $service->per_kg ? max($quantity, 3) : ceil($quantity);
        $subtotal = (int) round($billed * $service->price);
        $total = $subtotal + ($pickup && $subtotal < 100000 ? 10000 : 0);

        $order = Order::create([
            ...$data,
            'pickup' => $pickup,
            'total' => $total,
            'status' => OrderStatus::Queued,
            'received_at' => now(),
        ]);

        return redirect()->route('orders.show', $order)->with('message',
            "Order #{$order->id} created: " . Money::rp($total) . '.');
    }

    // POST /orders/5/advance: queued -> finished -> departed
    public function advance(Order $order)
    {
        $next = $order->status->next();
        if ($next === null) {
            return back()->with('error', 'This order has already left the shop.');
        }
        $order->status = $next;
        if ($next === OrderStatus::Finished) {
            $order->finished_at = now();
        } else {
            $order->departed_at = now();
        }
        $order->save();
```

Kode 7.17 (Lanjutan) *app/Http/Controllers/OrderController.php: daftar pesanan, halaman pesanan, pesanan baru yang dihargai dengan aturan Bab 3, dan alur kerjanya.*

```

        return back()->with('message',
            "Order #{$order->id} is now {$next->label()}.");
    }
}
```

- `index` memakai *eager loading* (pemuatan awal): `Order::with(['customer', 'service', 'payments'])` memuat pesanan lalu, dengan tiga *query* lagi, semua pelanggan, layanan, dan pembayarannya, dan menghubungkannya di memori. Tanpa `with`, setiap `$o->customer->name` di *view* akan mengirim *query*-nya sendiri: satu untuk daftar ditambah tiga per pesanan, *N+1 query problem* yang diukur Bab 8 [8].

- `store` memvalidasi dengan aturan yang membaca basis data: `exists:customers,id` dan `Rule::exists('services', 'id')->where('is_active', true)` memeriksa bahwa pelanggannya ada dan layanannya ada *serta* ditawarkan. `required_if_accepted:pickup` membuat alamat antar wajib hanya ketika kotak antar-jemput dicentang, aturan yang di Bab 6 memerlukan `IValidatableObject`.
- Harga dihitung dengan aturan Bab 3 dalam empat baris: jumlah yang ditagih (minimal 3 untuk layanan per kg, selain itu dibulatkan ke atas menjadi barang utuh), subtotal, dan biaya antar-jemput di bawah Rp 100.000. Baris-baris ini memakai *float* PHP; Soal 8.1 di Bab 8 memperlihatkan dengan tepat kapan hal itu berbahaya dan menggantinya dengan kelas yang memberi hasil sama dengan `PriceCalculator` versi C# dalam setiap kasus.
- `advance` menanyakan status berikutnya kepada enum, sehingga alur kerja tetap di satu tempat, mengisi cap waktu yang tepat, dan memanggil `save()`, yang mengirim `UPDATE` hanya untuk kolom yang berubah, persis seperti pelacak perubahan EF Core.

Controller pembayaran (**Kode 7.18**) memvalidasi jumlah dan cara bayar, dengan `Rule::enum(PaymentMethod::class)` yang hanya menerima `cash`, `qris`, atau `transfer`, lalu membuat pembayaran melalui relasi pesanan.

Kode 7.18 `app/Http/Controllers/PaymentController.php`: mencatat pembayaran untuk sebuah pesanan.

```php
<?php

namespace App\Http\Controllers;

use App\Enums\PaymentMethod;
use App\Models\Order;
use App\Support\Money;
use Illuminate\Http\Request;
use Illuminate\Validation\Rule;

class PaymentController extends Controller
{
    // POST /orders/5/payments
    public function store(Request $request, Order $order)
    {
        $data = $request->validate([
            'amount' => ['required', 'integer', 'min:1'],
            'method' => ['required', Rule::enum(PaymentMethod::class)],
        ]);
        $order->payments()->create([...$data, 'paid_at' => now()]);

        return back()->with('message',
            'Payment of ' . Money::rp((int) $data['amount']) . ' recorded.');
    }
}
```

Controller harga (**Kode 7.19**) mencerminkan `PricesController` Bab 5, termasuk kode layanan bebas berikutnya, dan ekspresi reguler yang mewajibkan `(per kg)`, `(per item)`, atau `(per pair)` di akhir deskripsi, aturan dari Bab 6. Kotak centang yang tidak dicentang sama sekali tidak dikirim peramban, sehingga `$request->boolean('is_active')` secara eksplisit mengubah "tidak ada" menjadi `false`.

Kode 7.19 `app/Http/Controllers/PriceController.php`: daftar harga milik pemilik.

```php
<?php

namespace App\Http\Controllers;

use App\Models\Service;
use Illuminate\Http\Request;

// The owner's price list: the same Service rows that /services shows.
class PriceController extends Controller
{
    public function index()
    {
        return view('prices.index', ['services' => Service::orderBy('id')->get()]);
    }

    public function create()
    {
        return view('prices.create', ['service' => new Service([
            'is_active' => true])]);
    }

    public function store(Request $request)
    {
        $data = $this->validated($request);
        $last = Service::max('id') ?? 'S0000';          // S0012 -> S0013
        $data['id'] = sprintf('S%04d', (int) substr($last, 1) + 1);
        $service = Service::create($data);

        return redirect()->route('prices.index')
            ->with('message', "{$service->id} {$service->name} added.");
    }

    public function edit(Service $service)
    {
        return view('prices.edit', compact('service'));
    }

    // A new price counts for new orders only: every order keeps its total.
    public function update(Request $request, Service $service)
    {
        $service->update($this->validated($request));

        return redirect()->route('prices.index')
            ->with('message', "{$service->id} {$service->name} saved.");
    }
```

Kode 7.19 (Lanjutan) *app/Http/Controllers/PriceController.php: daftar harga milik pemilik.*

```php
    private function validated(Request $request): array
    {
        $data = $request->validate([
            'description' => ['required', 'string', 'max:60',
                'regex:/^.{3,} \(per (kg|item|pair)\)$/'],
            'price' => ['required', 'integer', 'between:1000,1000000'],
        ], [
            'description.regex' =>
                'End the description with (per kg), (per item) or (per pair).',
        ]);
        $data['is_active'] = $request->boolean('is_active');   // checkbox

        return $data;
    }
}
```

Controller sisanya pendek. `CounterController` (**Kode 7.20**) menghitung empat angka halaman kasir, dengan `count()` dan `sum('amount')`; ia adalah *controller invokable* dengan satu metode `__invoke`, yang disebut rute tanpa nama metode. `HomeController`, `ServiceController`, dan `OutletController` melayani halaman publik Bab 3 dan ada di folder kode.

Kode 7.20 `app/Http/Controllers/CounterController.php`: empat angka untuk meja kasir.

```php
<?php

namespace App\Http\Controllers;

use App\Enums\OrderStatus;
use App\Models\Order;
use App\Models\Payment;

// GET /counter: what the cashier needs to know at a glance.
class CounterController extends Controller
{
    public function __invoke()
    {
        $today = today();
        $tomorrow = today()->addDay();

        return view('counter', [
            'queued' => Order::where('status', OrderStatus::Queued)->count(),
            'readyForPickup' => Order::where('status', OrderStatus::Finished)
                ->count(),
            'receivedToday' => Order::where('received_at', '>=', $today)
                ->where('received_at', '<', $tomorrow)->count(),
            'paidToday' => (int) Payment::where('paid_at', '>=', $today)
                ->where('paid_at', '<', $tomorrow)->sum('amount'),
        ]);
    }
}
```

7.9.3 Melihat SQL-nya

Di Bab 5 terminal menampilkan setiap perintah SQL yang dikirim EF Core. Laravel dapat melakukan hal yang sama dengan tiga cara. Di `tinker`, `Customer::where('name', 'like', '%an%')->toRawSql()` mengembalikan SQL dengan nilai-nilainya sudah terisi, tanpa menjalankannya. Untuk seluruh halaman, `DB::listen(fn ($query) => logger($query->sql, $query->bindings));` di `AppServiceProvider::boot()` menulis setiap perintah beserta nilai terikatnya ke `storage/logs/laravel.log`. Dan paket gratis Laravel Debugbar, yang dipasang hanya untuk pengembangan dengan `composer require barryvdh/laravel-debugbar --dev`, menampilkan bilah di bagian bawah setiap halaman berisi jumlah *query*, waktunya, dan mana yang berulang.

Menghitung *query* dengan cara ini layak dijadikan kebiasaan. Buka daftar pesanan, lihat jumlahnya, dan bandingkan dengan yang Anda harapkan: empat untuk daftar di bab ini (pesanan, lalu pelanggan, layanan, dan pembayaran). Bila angkanya bertambah seiring jumlah pesanan, ada `with` yang terlupa. Bab 8 mengubah hitungan inilah menjadi sebuah gambar dan membandingkannya di kedua kerangka kerja.

7.10 Templat Blade

7.10.1 *Layout*

Setiap halaman memperluas satu *layout* (**Kode 7.21**). `@yield('content')` menandai tempat isi halaman, dan `@yield('title')` tempat judulnya; sebuah halaman mengisinya dengan `@section`. `{{ asset('css/site.css') }}` membangun alamat lembar gaya di `public/css`, lembar gaya yang sama dengan versi ASP.NET Core. `@class(['here' => request()->routeIs('orders.*')])` menambahkan kelas `here` pada tautan menu bagian yang sedang dibuka, seperti pembantu `Here()` di *layout* Bab 3. `@session('message')` menampilkan pesan *flash* yang diatur oleh `with('message', …)`, dengan nilainya tersedia sebagai `$value`.

Kode 7.21 `resources/views/layouts/app.blade.php`: *layout* yang dipakai bersama oleh setiap halaman.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>@yield('title') - Profitina Laundry</title>
    <link rel="stylesheet" href="{{ asset('css/site.css') }}">
</head>
<body>
<header>
    <a class="brand" href="{{ route('home') }}">Profitina Laundry</a>
    <nav>
        <a href="{{ route('home') }}"
           @class(['here' => request()->routeIs('home')])>Home</a>
        <a href="{{ route('services.index') }}"
           @class(['here' => request()->routeIs('services.*')])>Services</a>
        <a href="{{ route('outlets.index') }}"
           @class(['here' => request()->routeIs('outlets.*')])>Outlets</a>
    </nav>
    <nav class="staff">
        <a href="{{ route('counter') }}"
           @class(['here' => request()->routeIs('counter')])>Counter</a>
        <a href="{{ route('orders.index') }}"
           @class(['here' => request()->routeIs('orders.*')])>Orders</a>
        <a href="{{ route('customers.index') }}"
           @class(['here' => request()->routeIs('customers.*')])>Customers</a>
        <a href="{{ route('prices.index') }}"
           @class(['here' => request()->routeIs('prices.*')])>Prices</a>
    </nav>
</header>
<main>
    @session('message')
        <p class="flash">{{ $value }}</p>
    @endsession
    @session('error')
        <p class="flash error">{{ $value }}</p>
    @endsession
    @yield('content')
</main>
<footer>
    <p>&copy; {{ date('Y') }} Profitina Laundry, Surabaya</p>
</footer>
</body>
</html>
```

7.10.2 *Escaping*, formulir, dan masukan lama

`{{ $customer->name }}` mencetak nama melalui `htmlspecialchars` milik PHP, sehingga pelanggan bernama `<script>alert(1)</script>` tampil sebagai teks alih-alih dijalankan [12]. `{!! … !!}` mencetak tanpa *escaping* dan hanya pantas di sekitar HTML yang dihasilkan aplikasi sendiri. Ini perlindungan yang sama dengan yang diberikan Razor lewat `@`, dan melupakannya adalah kesalahan *cross-site scripting* klasik pada PHP yang ditulis tangan.

Formulir pelanggan (**Kode 7.22**) dipakai bersama oleh halaman buat dan ubah, yang menyertakannya dengan `@include('customers._form')`. `old('name', $customer->name)` menampilkan nilai yang diketik pengguna bila pengiriman terakhir gagal validasi, dan selain itu nilai yang tersimpan; `@error('name') … @enderror` menampilkan pesan untuk isian itu, yang tersedia sebagai `$message`.

Kode 7.22 `resources/views/customers/_form.blade.php`: isian yang dipakai bersama oleh halaman buat dan ubah.

```
<label for="name">Name</label>
<input id="name" name="name" value="{{ old('name', $customer->name) }}">
@error('name') <span class="error">{{ $message }}</span> @enderror

<label for="phone">Phone (WhatsApp)</label>
<input id="phone" name="phone" value="{{ old('phone', $customer->phone) }}">
@error('phone') <span class="error">{{ $message }}</span> @enderror

<label for="address">Address</label>
<input id="address" name="address"
       value="{{ old('address', $customer->address) }}">
@error('address') <span class="error">{{ $message }}</span> @enderror
```

Halaman ubah (**Kode 7.23**) membungkus isian itu dalam formulir. `@csrf` menambahkan token CSRF tersembunyi yang diperiksa grup *middleware* `web` pada setiap `POST`, `PUT`, dan `DELETE`; formulir tanpa token ditolak dengan status 419 [13]. `@method('PUT')` mengubah `POST` menjadi `PUT` yang diharapkan *resource route*.

Kode 7.23 `resources/views/customers/edit.blade.php`: formulir ubah dengan token CSRF dan isian metodenya.

```
@extends('layouts.app')

@section('title', 'Edit customer')

@section('content')
<h1>Edit {{ $customer->name }}</h1>
<form method="post" action="{{ route('customers.update', $customer) }}"
      class="form">
    @csrf
    @method('PUT')
    @include('customers._form')
    <button type="submit">Save</button>
    <a href="{{ route('customers.show', $customer) }}">Cancel</a>
</form>
@endsection
```

Halaman pelanggan (**Kode 7.24**) mendaftar pesanan dengan `@forelse … @empty … @endforelse`, perulangan dengan cabang untuk daftar kosong, dan menawarkan tombol hapus hanya ketika pelanggan tidak memiliki pesanan. `@use('App\Support\Money')` mengimpor pembantu uang ke templat.

Kode 7.24 `resources/views/customers/show.blade.php`: satu pelanggan, pesanan-pesanannya, dan tombol hapus.

```
@extends('layouts.app')
@use('App\Support\Money')

@section('title', $customer->name)

@section('content')
<h1>{{ $customer->name }}</h1>
<p>{{ $customer->phone }} · {{ $customer->address ?? 'no address' }}
   · customer since {{ $customer->created_at->format('j M Y') }}</p>
<p>
    <a class="button"
       href="{{ route('orders.create', ['customer_id' => $customer->id]) }}">
        New order</a>
    <a href="{{ route('customers.edit', $customer) }}">Edit</a>
</p>
<h2>Orders</h2>
<table class="list">
    <thead>
        <tr><th>#</th><th>Received</th><th>Service</th><th class="num">Total</th>
            <th>Status</th></tr>
    </thead>
    <tbody>
    @forelse ($customer->orders as $o)
        <tr>
            <td><a href="{{ route('orders.show', $o) }}">{{ $o->id }}</a></td>
            <td>{{ $o->received_at->format('d/m H:i') }}</td>
            <td>{{ $o->service->name }}</td>
            <td class="num">{{ Money::rp($o->total) }}</td>
            <td>{{ $o->status->label() }}</td>
        </tr>
    @empty
        <tr><td colspan="5">No orders yet.</td></tr>
    @endforelse
    </tbody>
</table>
@if ($customer->orders->isEmpty())
    <form method="post" action="{{ route('customers.destroy', $customer) }}"
          onsubmit="return confirm('Delete this customer?')">
        @csrf
        @method('DELETE')
        <button type="submit" class="danger">Delete customer</button>
    </form>
@endif
@endsection
```

7.10.3 Halaman pesanan

Halaman pesanan (**Kode 7.25**) adalah *view* paling lengkap di bab ini. Halaman itu menampilkan fakta-fakta pesanan, tombol untuk langkah alur berikutnya, pembayaran, dan formulir pembayaran selama masih ada tagihan. Cara bayar diambil dari `PaymentMethod::cases()`, sehingga cara baru di enum muncul di halaman tanpa menyentuh HTML. `$order->finished_at?->format(...)` memakai operator *nullsafe* PHP: selama pesanan belum selesai, nilainya `null` dan tidak ada yang dicetak.

Kode 7.25 `resources/views/orders/show.blade.php`: satu pesanan dengan tombol alur kerja, pembayaran, dan formulir pembayarannya.

```
@extends('layouts.app')
@use('App\Enums\OrderStatus')
@use('App\Enums\PaymentMethod')
@use('App\Support\Money')

@section('title', "Order #{$order->id}")

@section('content')
<h1>Order #{{ $order->id }} · {{ $order->status->label() }}</h1>
<dl class="facts">
    <dt>Customer</dt>
    <dd><a href="{{ route('customers.show', $order->customer) }}">
        {{ $order->customer->name }}</a> ({{ $order->customer->phone }})</dd>
    <dt>Service</dt>
    <dd>{{ $order->service->name }}, {{ $order->quantity }}
        {{ $order->service->unit }}</dd>
    <dt>Delivery</dt>
    <dd>{{ $order->pickup ? "to {$order->delivery_address}"
                          : 'collected at the shop' }}</dd>
    <dt>Received</dt><dd>{{ $order->received_at->format('d/m/Y H:i') }}</dd>
    <dt>Finished</dt><dd>{{ $order->finished_at?->format('d/m/Y H:i') }}</dd>
    <dt>Departed</dt><dd>{{ $order->departed_at?->format('d/m/Y H:i') }}</dd>
    <dt>Note</dt><dd>{{ $order->note }}</dd>
    <dt>Total</dt><dd>{{ Money::rp($order->total) }}</dd>
    <dt>Paid</dt><dd>{{ Money::rp($order->paid()) }}</dd>
    <dt>Balance</dt><dd><strong>{{ Money::rp($order->balance()) }}</strong></dd>
</dl>

@if ($order->status !== OrderStatus::Departed)
    <form method="post" action="{{ route('orders.advance', $order) }}">
        @csrf
        <button type="submit">
            {{ $order->status === OrderStatus::Queued ? 'Mark finished'
                                                      : 'Mark picked up' }}
        </button>
    </form>
@endif
```

Kode 7.25 (Lanjutan) *resources/views/orders/show.blade.php: satu pesanan dengan tombol alur kerja, pembayaran, dan formulir pembayarannya.*

```
<h2>Payments</h2>
<table class="list">
    <thead><tr><th>Paid at</th><th>Method</th><th class="num">Amount</th></tr>
    </thead>
    <tbody>
    @foreach ($order->payments as $p)
        <tr><td>{{ $p->paid_at->format('d/m H:i') }}</td>
            <td>{{ $p->method->label() }}</td>
            <td class="num">{{ Money::rp($p->amount) }}</td></tr>
    @endforeach
    </tbody>
</table>

@if ($order->balance() > 0)
    <form method="post" action="{{ route('orders.payments.store', $order) }}"
          class="inline">
        @csrf
        <input name="amount" type="number" min="1"
               value="{{ old('amount', $order->balance()) }}">
        <select name="method">
            @foreach (PaymentMethod::cases() as $m)
                <option value="{{ $m->value }}">{{ $m->label() }}</option>
            @endforeach
        </select>
        <button type="submit">Record payment</button>
    </form>
    @error('amount') <p class="error">{{ $message }}</p> @enderror
    @error('method') <p class="error">{{ $message }}</p> @enderror
@endif
@endsection
```

View lainnya (daftar-daftar, formulir pesanan, halaman harga, halaman publik) mengikuti pola yang sama dan ada di folder kode. Meletakkan masing-masing di samping padanan Razor-nya dari Bab 3 dan 5 adalah latihan yang baik untuk membaca kedua bahasa templat: `@foreach` dan `@if` ada di keduanya, `asp-for` menjadi `name` ditambah `old()`, `asp-validation-for` menjadi `@error`, `<partial>` menjadi `@include`, dan `asp-route-id` menjadi `route(..., $model)`.

7.10.4 Razor dan Blade berdampingan

Kedua bahasa templat memecahkan masalah yang sama dengan tanda baca yang berbeda. **Tabel 7.5** memasangkan konstruksi yang dipakai di *view* laundry.

Tabel 7.5 Konstruksi Razor dan Blade untuk tugas yang sama.

Tugas	Razor (Bab 3 sampai 6)	Blade (bab ini)
cetak nilai, di-*escape*	`@Model.Name`	`{{ $customer->name }}`
cetak HTML tanpa *escape*	`@Html.Raw(html)`	`{!! $html !!}`
perulangan	`@foreach (var c in Model) { … }`	`@foreach ($customers as $c) … @endforeach`
percabangan	`@if (…) { … } else { … }`	`@if (…) … @else … @endif`
layout	`Layout = "_Layout"`, `@RenderBody()`	`@extends('layouts.app')`, `@yield('content')`
partial view	`<partial name="_Form" model="Model" />`	`@include('customers._form')`
tautan ke *action*	`<a asp-action="Details" asp-route-id="@c.Id">`	`<a href="{{ route('customers.show', $c) }}">`
isian formulir dengan nilai	`<input asp-for="Name" />`	`<input name="name" value="{{ old('name', $customer->name) }}">`
galat isian	`<span asp-validation-for="Name"></span>`	`@error('name') {{ $message }} @enderror`
token *anti-forgery*	otomatis dalam `<form method="post">`	`@csrf`
pesan *flash*	`TempData["Message"]`	`session('message')`, `@session`

Perbedaan terdalam ada pada cara keduanya mengenal model. *View* Razor menyatakan `@model Customer`, dan kompilator memeriksa setiap `Model.Name` terhadap kelasnya; kesalahan menggagalkan *build*, dan itulah sebabnya pemeriksaan tipe yang dijelaskan di Subbab 5.12 dapat mencakup *view*. *View* Blade menerima variabel lepas dari `view('customers.show', compact('customer'))`, dan `$customer->nmae` yang salah eja tidak mencetak apa-apa sampai seseorang melihat halamannya. *Tag helper* seperti `asp-for` juga menuliskan nama, nilai, tipe, dan atribut validasi isian untuk Anda, sedangkan Blade menyerahkan HTML-nya kepada Anda dan memberi `old()` serta `@error` untuk mengisinya. Tidak ada yang lebih baik dalam segala hal: Blade lebih dekat dengan HTML yang sudah Anda kenal, dan Razor menangkap lebih banyak kesalahan sebelum halaman sempat berjalan.

7.11 Menjalankannya

Di tempat buku ini ditulis, Packagist (registri paket Composer) tidak dapat dijangkau, sehingga tidak ada proyek Laravel yang dapat dibuat atau dijalankan di sana. Setiap berkas PHP di bab ini diperiksa dengan `php -l` (PHP 8.4), berkas Blade diperiksa oleh skrip agar direktifnya berpasangan, dan `tools/enum_demo.php` dijalankan; keluarannya di Subbab 7.6.1 asli. Di komputer Anda:

```
composer create-project laravel/laravel profitina-laundry "^13.0"
cd profitina-laundry
cp -R ../Chapter07/laravel/. .        # Windows: Copy-Item -Recurse -Force
php artisan migrate:fresh --seed
php artisan route:list --except-vendor
php artisan serve --port=4070
```

Atur `'timezone' => 'Asia/Jakarta'` di `config/app.php` lebih dulu, agar "hari ini" di halaman kasir dimulai tengah malam waktu Surabaya, bukan pukul 07.00 (tengah malam UTC). `route:list` mencetak rute-rute **Kode 7.15** beserta nama dan metode *controller*-nya; periksa bahwa semua *resource route* ada sebelum membuka peramban. Lalu ikuti tur yang sama dengan Subbab 5.12:

1. `http://localhost:4070` menampilkan tiga layanan populer; `/services` dan `/outlets` menampilkan data Bab 3 dari basis data.
2. **Counter** menampilkan dua pesanan antre, satu siap diambil, dan dua diterima hari ini.
3. **Customers**: cari `sari` dengan huruf kecil; berbeda dengan versi Bab 5, *Sari Wulandari* ditemukan, karena `LIKE` mengabaikan huruf besar-kecil.
4. Tambahkan pelanggan dengan nomor telepon yang sudah ada; formulir kembali dengan *The phone has already been taken.*, pesan bawaan Laravel untuk `unique`, dan nilai yang diketik masih ada di isiannya.
5. **Orders → New order**: 2 kg *Wash, dry & fold* untuk Dewi Lestari; pesannya menyebut Rp 24.000. Centang antar-jemput tanpa alamat; pesan *The delivery address field is required when pickup is accepted.* muncul.
6. Di halaman pesanan, catat pembayaran dan majukan pesanan dua kali.
7. **Prices**: ubah sebuah harga; `/services` menampilkan harga barunya.

Buka juga `/up`: setiap aplikasi Laravel 11 atau yang lebih baru menjawab di sana dengan halaman kesehatan kecil, padanan *endpoint* `/health` yang ditambahkan dengan tangan di Bab 6. Halaman staf bab ini belum memiliki fitur masuk; autentikasi Laravel adalah pokok bahasan Bab 10. Sampai saat itu, jalankan aplikasi hanya di komputer Anda sendiri.

7.12 Membuatnya Lebih Baik

- **Tangkap *query* N+1 sejak dini.** `Model::preventLazyLoading(! app()->isProduction());` di `AppServiceProvider::boot()` membuat Laravel melempar *exception* saat pengembangan setiap kali relasi dimuat secara malas (*lazy*) di dalam perulangan. `with` yang terlupa lalu gagal dengan keras di laptop Anda, bukan lambat di toko.
- **Pindahkan validasi ke *form request*.** Aturan di `store` dan `update` bertambah dengan setiap kebutuhan. Kelas *form request* (`php artisan make:request StoreOrderRequest`) memegang aturan, pesan, bahkan otorisasi satu formulir, dan *controller* hanya menerima masukan yang sudah divalidasi. Bab 9 melakukan hal ini.
- **Tulis pesan dalam bahasa toko.** `php artisan lang:publish` menyalin pesan validasi Laravel ke `lang/en/validation.php`, tempat pesan-pesan itu dapat ditulis ulang; menambahkan `lang/id/validation.php` dan mengatur `APP_LOCALE=id` menampilkannya dalam bahasa Indonesia.
- **Bagi daftar panjang menjadi halaman.** `->paginate(20)` alih-alih `->get()` mengembalikan satu halaman beserta tautan ke halaman lain; Bab 8 (Soal 8.4) menambahkannya di kedua kerangka kerja.
- **Pakai komponen Blade untuk *markup* yang berulang.** Isian formulir dengan label dan pesan galatnya muncul puluhan kali; komponen anonim `<x-field name="phone" label="Phone (WhatsApp)" :value="$customer->phone" />` di `resources/views/components/field.blade.php` cukup menuliskannya sekali.

7.13 Memakainya Sungguhan

- **MySQL alih-alih SQLite.** Ubah beberapa baris di `.env`: `DB_CONNECTION=mysql`, `DB_HOST`, `DB_DATABASE`, `DB_USERNAME`, dan `DB_PASSWORD`, lalu `php artisan migrate --seed`. Migrasinya ditulis dengan *schema builder* Laravel, bukan SQL, sehingga berjalan tanpa perubahan di SQLite, MySQL, MariaDB, PostgreSQL, dan SQL Server. Itulah satu keuntungan menulis migrasi dengan tangan dalam bahasa yang portabel alih-alih membangkitkannya untuk satu penyedia.
- ***Hosting* PHP bersama.** Sebagian besar paket *hosting* di Indonesia mendukung PHP 8.3 atau lebih baru dan Composer. Unggah proyeknya, arahkan *document root* domain ke `public/` (jangan pernah ke folder proyek, yang akan membuka `.env`), atur `APP_ENV=production` dan `APP_DEBUG=false`, lalu jalankan `php artisan migrate --force` dan `php artisan optimize`.
- **Layanan Anda sendiri.** Ganti berkas CSV di `database/data/` sebelum pengisian data pertama, atau pakai halaman Prices sesudahnya, persis seperti di Bab 5.
- **Cadangan.** Dengan SQLite seluruh toko adalah `database/database.sqlite`. Penjadwal Laravel dapat menyalinnya setiap malam: sebuah *closure* terjadwal di `routes/console.php` yang setiap hari menjalankan `copy(database_path('database.sqlite'), storage_path('backups/laundry-'.date('Y-m-d').'.sqlite'))`, ditambah satu entri *cron* di server, `* * * * * php /path/to/artisan schedule:run`, yang membiarkan Laravel memutuskan kapan setiap tugas terjadwal jatuh tempo. Di MySQL, pakai `mysqldump` dari *cron*, dan dalam setiap kasus salin cadangannya ke tempat kedua.
- **Dua kerangka kerja, satu toko.** Tidak ada yang memaksa toko memilih. Versi ASP.NET Core dapat berjalan di gerai utama dan versi Laravel di Wiyung, masing-masing dengan basis datanya sendiri, atau keduanya dapat berbagi satu basis data MySQL bila migrasinya dijaga identik, yang lebih sulit daripada kedengarannya; jawaban yang lazim adalah satu sistem pencatat utama dan API kecil di antara keduanya, pokok bahasan Bab 10 dan 14.
- **Data pribadi.** Kewajiban UU PDP yang sama dengan Bab 5 dan 6 berlaku: kumpulkan hanya yang diperlukan toko, lindungi berkas basis data dan cadangannya, dan jauhkan `.env`, yang berisi kunci aplikasi, dari *version control*. Halaman staf tidak boleh daring sebelum fitur masuk di Bab 10 melindunginya.

7.14 Profitina dalam Praktik

Situs publik Profitina sendiri dibangun dengan PHP: situs pemasarannya di profitina.com berjalan di WordPress dengan tema dwibahasa (Indonesia dan Inggris) buatan sendiri, dan menerbitkan berkas `/llms.txt` yang memberi tahu mesin jawab AI tentang isi situs itu. Aplikasi di app.profitina.com, sebaliknya, ditulis dengan Python dan FastAPI. Menjalankan dua tumpukan teknologi berdampingan itu lazim, dan untuk alasan yang sama dengan bab ini: setiap bagian memakai alat yang cocok untuknya, sementara gagasan *routing*, templat, ORM, dan migrasi tetap sama di keduanya. Pengembang yang sudah membangun Profitina Laundry dengan ASP.NET Core dan Laravel dapat membaca bagian mana pun dari sistem semacam itu sejak hari pertama.

7.15 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- ***Mass assignment* tanpa `#[Fillable]`.** `Customer::create($request->all())` pada model tanpa daftar *fillable* melempar `MassAssignmentException`; lebih buruk lagi, menjadikan semuanya *fillable* dengan `$guarded = []` membiarkan permintaan mengisi kolom apa pun. Daftarlah atribut yang *fillable* dan berikan hanya hasil `$request->validate(...)`.

- **Lupa `@csrf`.** Formulir tanpa token ditolak dengan *419 Page Expired*. Setiap formulir `POST`, `PUT`, dan `DELETE` memerlukannya.
- **Memakai `{!! !!}` untuk data pengguna.** Direktif itu mencetak tanpa *escaping* dan membuka halaman bagi *cross-site scripting*. Pakailah `{{ }}` untuk semua yang berasal dari pengguna.
- ***Query* N+1.** Perulangan yang membaca `$order->customer->name` tanpa `with('customer')` mengirim satu *query* per pesanan. Muat relasi dengan `with` atau `load`, dan nyalakan `preventLazyLoading` saat pengembangan.
- **Menjalankan `migrate:fresh` di server.** Perintah itu menghapus setiap tabel. Di server toko pakailah `php artisan migrate --force`, yang hanya menjalankan migrasi baru.
- **Mengarahkan server web ke folder proyek.** Hanya `public/` yang boleh dijangkau dari web; selebihnya, termasuk `.env` dan `database/database.sqlite`, tidak boleh.
- **Membiarkan `APP_DEBUG=true` di produksi.** Halaman galat lalu menampilkan kode, pengaturan, dan kadang kata sandi. Produksi selalu berjalan dengan `APP_DEBUG=false`.
- **Zona waktu yang salah.** Zona waktu bawaan Laravel adalah UTC; tanpa `Asia/Jakarta`, "hari ini" dimulai pukul tujuh pagi di Surabaya dan pesanan yang diterima sebelum pukul tujuh terhitung kemarin.

7.16 Ringkasan Bab

- Laravel 13 (Maret 2026) berjalan di PHP 8.3 sampai 8.5; Composer memasangnya, Artisan membuat berkas dan menjalankan migrasi, dan proyek baru memakai SQLite secara bawaan.
- Migrasi ditulis dengan tangan memakai *schema builder* dan dijalankan menurut urutan nama berkasnya; `foreignId(...)->constrained()` membuat kunci asing, dan `restrictOnDelete` serta `cascadeOnDelete` menentukan akibat menghapus induk.
- Model Eloquent adalah *active record*: setiap model mengenal tabelnya, dan objeknya menyimpan dirinya sendiri. `#[Fillable]` membatasi *mass assignment*, `casts()` mengubah kolom menjadi tipe PHP dan enum, *accessor* menghitung atribut, dan metode seperti `hasMany` dan `belongsTo` menyatakan relasi.
- `routes/web.php` memegang setiap rute; `Route::resource` menyatakan tujuh rute CRUD, dan *route model binding* mengubah `{customer}` menjadi sebuah `Customer` atau 404.
- *Controller* memvalidasi dengan `$request->validate(...)`, yang mengalihkan kembali beserta galat dan masukan lama bila sebuah aturan gagal, dan memakai `with` untuk *eager loading*.
- Blade meng-*escape* semua yang dicetak dengan `{{ }}`, membangun *layout* dengan `@extends`, `@section`, dan `@yield`, berbagi isian dengan `@include`, dan melindungi formulir dengan `@csrf`.
- Versi Laravel dan versi ASP.NET Core berbagi rancangan basis data, halaman, dan aturan; keduanya berbeda dalam tempat tinggal segala sesuatu dan seberapa banyak yang dikerjakan konvensi.

7.17 Soal Latihan

1. Tuliskan SQL yang dikirim `Customer::query()->when($search !== '', …)->orderBy('name')->get()` untuk pencarian `an`, dan daftarlah pelanggan dari data contoh yang dikembalikannya. Apa bedanya hasil itu dengan versi Bab 5 untuk pencarian `AN`?
2. Dengan memakai **Tabel 7.4**, sebutkan metode, alamat, dan metode *controller* untuk setiap *action* berikut: menampilkan formulir ubah untuk pelanggan 2, menyimpan formulir itu, menghapus pelanggan 4, dan menampilkan daftar yang disaring dengan pencarian `budi`.

3. Tambahkan halaman penawaran harga Bab 3 ke Laravel: rute `GET /services/{service}/quote?quantity=4.5&pickup=1` yang menampilkan jumlah yang ditagih, subtotal, biaya, dan total untuk layanan tersebut. Tulislah rute dan metode *controller*-nya.
4. *View* Blade `orders/index` memakai `$o->balance()`, yang menjumlahkan `$o->payments`. Apa yang terjadi pada jumlah *query* bila *controller* lupa `'payments'` di `with([...])`, untuk 20 pesanan? Bagaimana bila `preventLazyLoading` dinyalakan?
5. Tulislah migrasi yang menambahkan kolom `rack` yang boleh kosong, paling panjang 8 karakter, ke `orders`, setelah `note`, dengan `down()` yang menghapusnya. Perintah Artisan mana yang membuat berkasnya, dan mana yang menjalankannya?
6. Bandingkan dua cara menolak penghapusan pelanggan yang memiliki pesanan: pemeriksaan di metode `destroy` dan `restrictOnDelete()` di migrasi. Masing-masing melindungi dari apa, dan mengapa keduanya dipertahankan?

7.18 Penyelesaian Soal Latihan

1. `select * from "customers" where "name" like ? or "phone" like ? order by "name" asc`, dengan kedua nilai terikat `%an%`. Hasilnya Budi Santoso dan Sari Wulandari (Dewi Lestari tidak mengandung `an`). Untuk `AN`, `LIKE` milik SQLite mengabaikan huruf besar-kecil untuk huruf ASCII, sehingga hasilnya sama; versi Bab 5 memakai `instr`, yang membedakan huruf besar-kecil, dan tidak menemukan siapa pun untuk `AN`.

2. `GET /customers/2/edit` → `edit`; `PUT /customers/2` (sebuah `POST` dengan `_method=PUT`) → `update`; `DELETE /customers/4` (sebuah `POST` dengan `_method=DELETE`) → `destroy`; `GET /customers?search=budi` → `index`.

3. Rute dan metodenya di bawah ini (**Kode 7.26**); *view*-nya mencetak lima nilai tersebut. *Route model binding* menyediakan layanannya, dan validasi menjaga *query string* tetap jujur.

Kode 7.26 Soal 7.3: rute dan metode *controller* untuk penawaran harga (tambahkan `use Illuminate\Http\Request;` ke *controller*).

```
// routes/web.php
Route::get('services/{service}/quote', [ServiceController::class, 'quote'])
    ->name('services.quote');

// app/Http/Controllers/ServiceController.php
public function quote(Request $request, Service $service)
{
    $data = $request->validate(['quantity' => ['required', 'numeric',
        'min:0.1', 'max:100']]);
    $quantity = (float) $data['quantity'];
    $billed = $service->per_kg ? max($quantity, 3) : ceil($quantity);
    $subtotal = (int) round($billed * $service->price);
    $fee = $request->boolean('pickup') && $subtotal < 100000 ? 10000 : 0;

    return view('services.quote', compact('service', 'quantity', 'billed',
        'subtotal', 'fee') + ['total' => $subtotal + $fee]);
}
```

Untuk `S0001` dengan 2 kg dan antar-jemput: ditagih 3, subtotal Rp 24.000, biaya Rp 10.000, total Rp 34.000.

4. Tanpa `'payments'`, setiap `$o->balance()` memicu pemuatan malas pembayaran pesanan itu: 20 *query* tambahan, seluruhnya 1 + 2 + 20 = 23 alih-alih 4. Dengan `preventLazyLoading` menyala, `$o->payments` yang pertama melempar `LazyLoadingViolationException` yang menyebut `App\Models\Order::payments`, dan kesalahannya langsung ditemukan.

5. `php artisan make:migration add_rack_to_orders_table --table=orders` membuat berkasnya; `php artisan migrate` menjalankannya. Isinya:

Kode 7.27 Soal 7.5: metode `up` dan `down` dari migrasi yang menambahkan kolom rak.

```
public function up(): void
{
    Schema::table('orders', function (Blueprint $table) {
        $table->string('rack', 8)->nullable()->after('note');
    });
}

public function down(): void
{
    Schema::table('orders', function (Blueprint $table) {
        $table->dropColumn('rack');
    });
}
```

`after('note')` hanya berpengaruh di MySQL dan MariaDB, tempat kolom memiliki urutan yang terlihat; SQLite menambahkan kolom di akhir. Bab 8 memakai migrasi ini pada Soal 8.7 (**Kode 7.27**).

6. Pemeriksaan di `destroy` melindungi pengguna: mengubah keadaan itu menjadi kalimat yang dipahami kasir dan menjaga halaman tetap bekerja. Kunci asing melindungi data: tetap berlaku bahkan bila kelak sepotong kode, sesi `tinker`, atau perintah SQL tulisan tangan mencoba menghapus pelanggan tanpa pemeriksaan itu. Yang pertama adalah sopan santun, yang kedua jaminan, dan toko memerlukan keduanya.

Daftar Pustaka

[1] Laravel, “Release notes, Laravel 13.x,” diakses Sep. 2026. [Daring]. Tersedia: https://laravel.com/docs/13.x/releases

[2] Laravel, “Installation, Laravel 13.x,” diakses Sep. 2026. [Daring]. Tersedia: https://laravel.com/docs/13.x/installation

[3] The PHP Group, “PHP 8.5 released,” 20 Nov. 2025. [Daring]. Tersedia: https://www.php.net/releases/8.5/en.php

[4] Laravel, “Database: migrations, Laravel 13.x,” diakses Sep. 2026. [Daring]. Tersedia: https://laravel.com/docs/13.x/migrations

[5] The PHP Group, “Enumerations,” PHP Manual, diakses Sep. 2026. [Daring]. Tersedia: https://www.php.net/manual/en/language.enumerations.php

[6] Laravel, “Eloquent: getting started, Laravel 13.x,” diakses Sep. 2026. [Daring]. Tersedia: https://laravel.com/docs/13.x/eloquent

[7] Laravel Daily, “PHP attributes in Laravel 13: the ultimate guide,” 2026. [Daring]. Tersedia: https://laraveldaily.com/post/php-attributes-in-laravel-13-the-ultimate-guide-36-new-attributes

[8] Laravel, “Eloquent: relationships, Laravel 13.x,” diakses Sep. 2026. [Daring]. Tersedia: https://laravel.com/docs/13.x/eloquent-relationships

[9] Laravel, “Routing: route model binding, Laravel 13.x,” diakses Sep. 2026. [Daring]. Tersedia: https://laravel.com/docs/13.x/routing#route-model-binding

[10] Laravel, “Controllers: resource controllers, Laravel 13.x,” diakses Sep. 2026. [Daring]. Tersedia: https://laravel.com/docs/13.x/controllers#resource-controllers

[11] Laravel, “Validation, Laravel 13.x,” diakses Sep. 2026. [Daring]. Tersedia: https://laravel.com/docs/13.x/validation

[12] Laravel, "Blade templates, Laravel 13.x," diakses Sep. 2026. [Daring]. Tersedia: https://laravel.com/docs/13.x/blade

[13] Laravel, "CSRF protection, Laravel 13.x," diakses Sep. 2026. [Daring]. Tersedia: https://laravel.com/docs/13.x/csrf

Bab 8

Evaluasi dan Pembahasan II: Rangkaian Fitur yang Sama di ASP.NET Core dan Laravel

Bab ini menengok kembali Bab 5, 6, dan 7 dengan memecahkan tujuh masalah nyata Profitina Laundry dua kali, sekali di aplikasi ASP.NET Core dari Bab 6 dan sekali di aplikasi Laravel dari Bab 7. Setiap soal dikerjakan dengan cara yang sama: soal sebagaimana dinyatakan toko, pemahaman dengan tangan memakai angka-angka toko sendiri, rancangan di atas kertas, penyelesaian ASP.NET Core, penyelesaian Laravel, dan penyelesaian keliru yang menggoda beserta alasan kegagalannya. Soal-soalnya mencakup empat tempat tinggal kode kerangka kerja: kelas domain biasa yang tidak tahu apa-apa tentang web (aturan harga dan alur pesanan), *query* (daftar pesanan, pencarian pelanggan, dan laporan pendapatan), validasi (pembayaran yang terlalu besar), dan skema (kolom baru). Di akhir bab Anda akan mampu menentukan untuk kebutuhan baru di lapisan mana ia berada, menuliskannya di kerangka kerja mana pun, memastikan kedua kerangka kerja memberi jawaban yang sama, dan mengenali kesalahan klasik sebelum sampai ke toko.

8.1 Posisi Kita Sekarang

Bab 5 menghubungkan aplikasi ASP.NET Core dari Bab 3 ke basis data SQLite dengan Entity Framework Core: kelas entitas, `DbContext`, migrasi, pengisian data dari berkas CSV Bab 3, *query* LINQ yang diterjemahkan menjadi SQL, dan halaman CRUD untuk pelanggan, harga, pesanan, dan pembayaran. Bab 6 membuat aplikasi itu layak dipercaya: atribut validasi, kelas formulir dan aturan dua isian; ASP.NET Core Identity dengan kata sandi yang di-*hash*, penguncian, serta peran Owner dan Cashier; kebijakan cadangan yang menutup setiap halaman yang tidak dibuka secara eksplisit; dan penerapan ke server Linux atau Windows. Bab 7 membangun toko yang sama dengan Laravel 13: migrasi yang ditulis tangan, model Eloquent dengan *cast*, enum, dan relasi, *resource route* dengan *route model binding*, validasi di *controller*, dan *view* Blade.

Kedua aplikasi kini mengerjakan hal yang sama dengan data yang sama. Justru dalam keadaan seperti itulah perbedaan terlihat: aturan yang ditulis sedikit berbeda di dua tempat, *query* yang cepat di satu sisi dan lambat di sisi lain, pemeriksaan yang hanya ada di satu sisi. Tujuh soal di bab ini lahir dari keadaan itu. Semuanya adalah permintaan pemilik dalam minggu-minggu setelah kedua versi mulai dipakai.

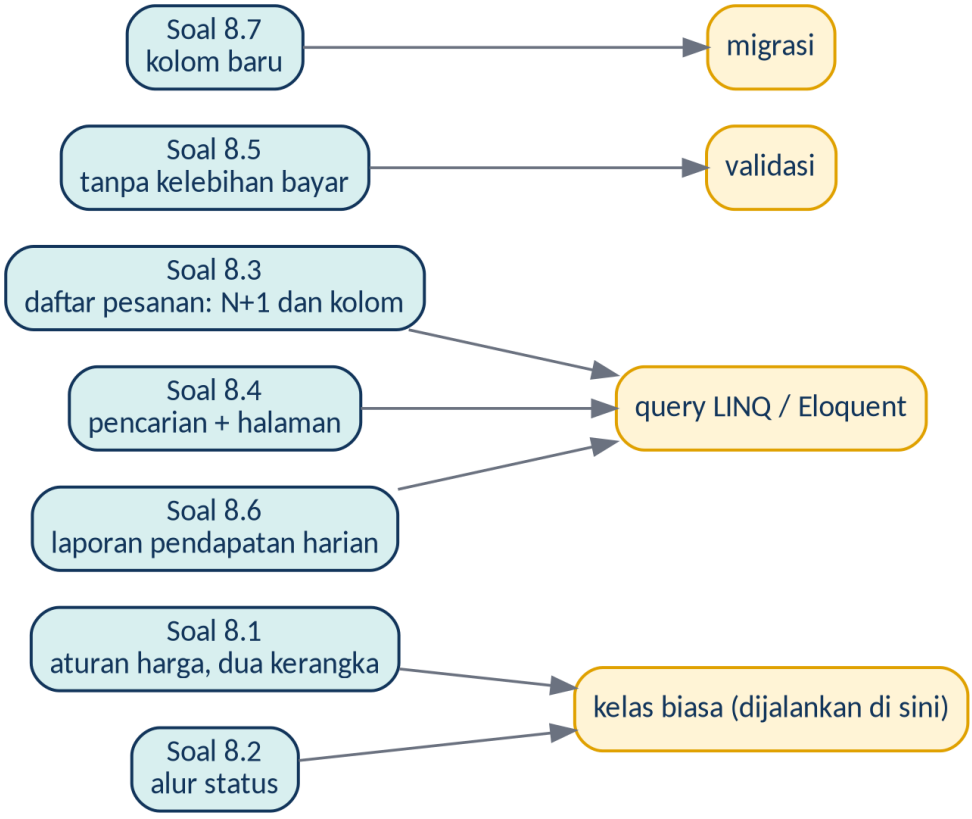
8.2 Cara Mendekati Soal Kerangka Kerja

8.2.1 Enam langkah

Setiap soal di bab ini dipecahkan dengan enam langkah yang sama, dan langkah-langkah itu berlaku untuk fitur apa pun di kerangka kerja apa pun.

1. **Nyatakan dengan kata-kata toko.** "Pelanggan tidak boleh membayar lebih dari tagihannya" adalah kebutuhan; "tambahkan aturan max ke validator" sudah merupakan penyelesaian. Mulailah dari kebutuhannya.
2. **Kerjakan contoh dengan tangan.** Ambil angka nyata dari data contoh dan hitung jawaban yang diharapkan di atas kertas. Jawaban itu menjadi patokan pengujian.
3. **Tentukan lapisannya.** Apakah aturan itu memerlukan web (permintaan, formulir, halaman)? Basis data? Tidak keduanya? Aturan yang tidak memerlukan keduanya masuk ke kelas biasa yang dapat dipanggil kedua kerangka kerja dan dapat dijalankan tanpa server. Aturan tentang satu formulir masuk ke validasi; pertanyaan tentang banyak baris masuk ke *query*; perubahan pada apa yang disimpan masuk ke migrasi.
4. **Tulis di satu kerangka kerja.** Biasanya yang paling Anda kuasai.
5. **Pindahkan, lalu bandingkan.** Tulis di kerangka kerja yang lain dan jalankan keduanya pada contoh di langkah 2. Keduanya harus sepakat sampai ke rupiahnya.
6. **Coba patahkan.** Carilah penyelesaian keliru yang menggoda: versi yang benar pada contoh tetapi gagal pada contoh berikutnya.

Gambar 8.1 memetakan tujuh soal ke lapisan-lapisan pada langkah 3.



Gambar 8.1 Tujuh soal di bab ini dan lapisan tempat masing-masing berada.

8.2.2 Tempat kode berada, dan apa yang dijalankan

Penyelesaiannya berupa aplikasi lengkap: `Chapter08/aspnet/LaundryMvc` adalah aplikasi Bab 6 yang ditambahi setiap penyelesaian, dan `Chapter08/laravel` memuat berkas-berkas Bab 7 yang ditambahi setiap penyelesaian. Dua program kecil memeriksa aturan domain tanpa server web: `Chapter08/aspnet/CoreCheck` dalam C# dan `Chapter08/laravel/tools/core_check.php` dalam PHP biasa. Di tempat buku ini ditulis, NuGet dan Packagist tidak dapat dijangkau, sehingga kedua aplikasi web tidak dapat dijalankan di sana; aplikasi C# dikompilasi terhadap tiruan EF Core yang

khusus untuk kompilasi, dan setiap berkas PHP lolos `php -l`. Kedua program pemeriksa dan demonstrasi PHP tentang kesalahan *float* dijalankan, dan setiap keluaran di bab ini adalah keluaran asli program-program itu.

8.3 Soal 8.1: Satu Aturan Harga, Dua Kerangka Kerja

8.3.1 Soalnya

Aturan harga Bab 3 adalah: layanan per kg ditagih paling sedikit 3 kg; layanan per potong atau per pasang ditagih dalam barang utuh, dibulatkan ke atas; antar-jemput dikenai Rp 10.000, gratis mulai subtotal Rp 100.000. Aplikasi C# menghitungnya di `PriceCalculator` dengan aritmetika `decimal`. Aplikasi Laravel Bab 7 menghitungnya langsung di `OrderController::store` dengan *float* PHP. Pemilik meminta satu hal: *kedua sistem harus selalu menagih harga yang sama, untuk setiap jumlah yang dapat diketik kasir*. Tulislah kelas PHP yang menjaminnya, dan perlihatkan bahwa keduanya sepakat.

8.3.2 Memahaminya dengan tangan

Ambil *Wash, dry & fold* seharga Rp 8.000 per kg. Untuk 4,5 kg tagihannya $4.5 \times 8000 = 36000$. Untuk 2 kg berlaku minimum: $3 \times 8000 = 24000$. Untuk 4,02 kg: $4.02 \times 8000 = 32160$. *Shoes deep clean* seharga Rp 45.000 per pasang untuk 1,5 pasang ditagih sebagai 2 pasang, Rp 90.000. *Bed cover* seharga Rp 35.000 untuk 2 potong dengan antar-jemput: $70000 + 10000 = 80000$. *Dry cleaning, shirt* seharga Rp 20.000 untuk 5 potong dengan antar-jemput: subtotalnya tepat Rp 100.000, jadi antar-jemput gratis dan totalnya Rp 100.000.

Setiap jumlah yang diterima meja kasir memiliki paling banyak dua desimal, sehingga merupakan bilangan bulat perseratusan: 4,5 kg adalah 450 perseratus, 4,02 kg adalah 402. Misalkan h adalah jumlah dalam perseratus dan P harga per satuan dalam rupiah. Jumlah yang ditagih b, juga dalam perseratus, adalah

$$b = \begin{cases} \max(h,\ 300) & \text{per kg} \\ 100 \cdot \lceil h/100 \rceil & \text{per potong atau pasang} \end{cases} \tag{8.1}$$

dan subtotal dalam rupiah, dibulatkan setengah ke atas ke rupiah utuh, adalah

$$S = \left\lfloor \frac{P \cdot b + 50}{100} \right\rfloor \tag{8.2}$$

Untuk 4,02 kg: $b = \max(402, 300) = 402$ menurut Persamaan (8.1), dan $S = \lfloor (8000 \cdot 402 + 50)/100 \rfloor = \lfloor 3216050/100 \rfloor = 32160$ menurut Persamaan (8.2). Setiap langkahnya aritmetika bilangan bulat: perkalian, penjumlahan, dan pembagian bilangan bulat. Totalnya adalah S ditambah Rp 10.000 bila antar-jemput dipilih dan $S < 100000$.

8.3.3 Merancangnya di atas kertas

Aturan ini tidak memerlukan permintaan maupun basis data, sehingga tempatnya di kelas biasa: `App\Domain\PriceCalculator` di PHP, bersebelahan dengan `PriceCalculator` yang sudah ada di C#. Kelas PHP itu menerima jumlah sebagai *untai*, persis seperti datangnya dari formulir, dan mengubahnya menjadi perseratus tanpa pernah menjadi *float*. Program pemeriksa di masing-masing bahasa menjalankan sembilan kasus yang sama dan mencetak baris-baris yang sama, sehingga membandingkan kedua keluaran cukup dengan satu `diff`.

8.3.4 Penyelesaian ASP.NET Core

Sisi C# tidak memerlukan kode baru. **Kode 8.1** adalah `PriceCalculator` dari Bab 3, tidak berubah sejak itu. `decimal` menyimpan 4,02 secara tepat, sehingga `billed * service.Price` tepat, dan aplikasi membulatkan total setengah ke atas dengan `MidpointRounding.AwayFromZero` ketika menyimpannya sebagai rupiah utuh.

Kode 8.1 `Models/PriceCalculator.cs`: aturan harga Bab 3 dalam C#, dengan aritmetika `decimal`.

```
namespace LaundryMvc.Models;

public record Quote(Service Service, decimal Quantity, decimal Billed,
    decimal Subtotal, decimal PickupFee, decimal Total, List<string> Notes);

// The shop's pricing rules live here, not in a controller or a view.
public class PriceCalculator
{
    public const decimal MinimumKg = 3m;
    public const decimal PickupFee = 10_000m;
    public const decimal FreePickupFrom = 100_000m;

    public Quote Calculate(Service service, decimal quantity, bool pickup)
    {
        var notes = new List<string>();
        decimal billed = quantity;
        if (service.PerKg && quantity < MinimumKg)
        {
            billed = MinimumKg;
            notes.Add($"Minimum {MinimumKg} kg, so {quantity} kg is billed "
                + $"as {MinimumKg} kg.");
        }
        else if (!service.PerKg && quantity != Math.Ceiling(quantity))
        {
            billed = Math.Ceiling(quantity);
            notes.Add($"Items are counted whole: {quantity} becomes {billed}.");
        }
        decimal subtotal = billed * service.Price;
        decimal fee = 0m;
        if (pickup)
        {
            fee = subtotal >= FreePickupFrom ? 0m : PickupFee;
            notes.Add(fee == 0m
                ? $"Pickup and delivery are free from {Money.Rp(FreePickupFrom)}."
                : $"Pickup and delivery cost {Money.Rp(PickupFee)}.");
        }
        return new Quote(service, quantity, billed, subtotal, fee,
            subtotal + fee, notes);
    }
}
```

`CoreCheck` (**Kode 8.2**) mengompilasi kelas-kelas model aplikasi dan kelas alur kerja dari Soal 8.2 lalu menjalankannya pada kasus-kasus tetap. Keluaran aslinya menyusul.

Kode 8.2 `CoreCheck/Program.cs`: aturan harga dan alur kerja pada kasus-kasus tetap, dalam C#.

```
using LaundryMvc.Domain;
using LaundryMvc.Models;

// The same cases as core_check.php; both programs must print the same lines.
(string Description, decimal Price, decimal Qty, bool Pickup)[] cases =
[
    ("Wash, dry & fold (per kg)", 8000, 4.5m, false),
    ("Wash, dry & fold (per kg)", 8000, 2m, false),
    ("Wash, dry & fold (per kg)", 8000, 4.02m, false),
    ("Ironing only (per kg)", 6000, 3.35m, false),
    ("Curtain (per kg)", 15000, 4.4m, true),
    ("Bed cover, any size (per item)", 35000, 2m, true),
    ("Dry cleaning - suit 2 pieces (per item)", 65000, 2m, true),
    ("Dry cleaning - shirt/blouse (per item)", 20000, 5m, true),
    ("Shoes deep clean (per pair)", 45000, 1.5m, false),
];
var calculator = new PriceCalculator();
Console.WriteLine("Prices (Problem 8.1)");
foreach (var c in cases)
{
    var service = new Service { Description = c.Description, Price = c.Price };
    var quote = calculator.Calculate(service, c.Qty, c.Pickup);
    var total = (int)decimal.Round(quote.Total, MidpointRounding.AwayFromZero);
    Console.WriteLine($"  {service.Name,-28} {c.Qty,5:0.00} {service.Unit,-4} "
                      + $"{(c.Pickup ? "pickup" : ""),-6} {total,7}");
}

Console.WriteLine("Workflow (Problem 8.2)");
(OrderStatus Status, int Balance)[] steps =
[
    (OrderStatus.Queued, 25000), (OrderStatus.Finished, 25000),
    (OrderStatus.Finished, 0), (OrderStatus.Departed, 0),
];
foreach (var (status, balance) in steps)
{
    try
    {
        var next = OrderWorkflow.Advance(status, balance);
        Console.WriteLine($"  {status,-8} owing {balance,6} -> {next}");
    }
    catch (InvalidOperationException e)
    {
        Console.WriteLine($"  {status,-8} owing {balance,6} -> {e.Message}");
    }
}
```

```
Prices (Problem 8.1)
  Wash, dry & fold              4.50 kg            36000
  Wash, dry & fold              2.00 kg            24000
  Wash, dry & fold              4.02 kg            32160
  Ironing only                  3.35 kg            20100
  Curtain                       4.40 kg   pickup   76000
  Bed cover, any size           2.00 item pickup   80000
  Dry cleaning - suit 2 pieces  2.00 item pickup  130000
  Dry cleaning - shirt/blouse   5.00 item pickup  100000
  Shoes deep clean              1.50 pair          90000
Workflow (Problem 8.2)
  Queued   owing  25000 -> Finished
  Finished owing  25000 -> Collect Rp 25.000 before handing over the laundry.
  Finished owing      0 -> Departed
  Departed owing      0 -> This order has already left the shop.
```

8.3.5 Penyelesaian Laravel

Kode 8.3 menerapkan Persamaan (8.1) dan Persamaan (8.2) apa adanya. `hundredths` membaca jumlah dengan ekspresi reguler, bagian bulat dan paling banyak dua desimal, dan menolak selain itu; `str_pad($m[2] ?? '', 2, '0')` mengubah `"5"` pada `"4.5"` menjadi 50 perseratus dan `"02"` pada `"4.02"` menjadi 2. Sejak itu, hanya `intdiv`, `+`, dan `*` pada bilangan bulat yang dipakai.

Kode 8.3 `app/Domain/PriceCalculator.php`: aturan yang sama dalam PHP, dengan aritmetika bilangan bulat yang tepat.

```php
<?php

namespace App\Domain;

use InvalidArgumentException;

// Problem 8.1: the price rules of Chapter 3, with exact integer arithmetic,
// so that Laravel charges exactly what the ASP.NET Core version charges.
final class PriceCalculator
{
    public const MINIMUM_KG = 3;
    public const PICKUP_FEE = 10000;
    public const FREE_PICKUP_FROM = 100000;

    public static function total(bool $perKg, int $price, string $quantity,
        bool $pickup): int
    {
        $hundredths = self::hundredths($quantity);
        $billed = $perKg
            ? max($hundredths, self::MINIMUM_KG * 100)    // at least 3 kg
            : intdiv($hundredths + 99, 100) * 100;        // whole items
        $subtotal = intdiv($price * $billed + 50, 100);   // half up
        $fee = $pickup && $subtotal < self::FREE_PICKUP_FROM
            ? self::PICKUP_FEE : 0;

        return $subtotal + $fee;
    }

    // "4.5" -> 450, "3.35" -> 335: read as text, never as a float.
    public static function hundredths(string $quantity): int
    {
        if (! preg_match('/^(\d+)(?:\.(\d{1,2}))?$/', trim($quantity), $m)) {
            throw new InvalidArgumentException(
                "Not a quantity with at most two decimals: {$quantity}");
        }

        return (int) $m[1] * 100 + (int) str_pad($m[2] ?? '', 2, '0');
    }
}
```

Program pemeriksanya (**Kode 8.4**) memuat kelas-kelas dengan `require`, karena Laravel tidak diperlukan, lalu menjalankan kasus-kasus yang sama.

Kode 8.4 `tools/core_check.php`: kasus-kasus yang sama dengan `CoreCheck`, dalam PHP biasa.

```php
<?php
// Plain PHP, no Laravel needed: php tools/core_check.php
// The same cases as CoreCheck in C#; both must print the same lines.

foreach (['Enums/OrderStatus', 'Support/Money', 'Domain/PriceCalculator',
    'Domain/OrderWorkflow'] as $file) {
    require __DIR__ . "/../app/{$file}.php";
}

use App\Domain\OrderWorkflow;
use App\Domain\PriceCalculator;
use App\Enums\OrderStatus;

$cases = [
    ['Wash, dry & fold', 'kg', 8000, '4.5', false],
    ['Wash, dry & fold', 'kg', 8000, '2', false],
    ['Wash, dry & fold', 'kg', 8000, '4.02', false],
    ['Ironing only', 'kg', 6000, '3.35', false],
    ['Curtain', 'kg', 15000, '4.4', true],
    ['Bed cover, any size', 'item', 35000, '2', true],
    ['Dry cleaning - suit 2 pieces', 'item', 65000, '2', true],
    ['Dry cleaning - shirt/blouse', 'item', 20000, '5', true],
    ['Shoes deep clean', 'pair', 45000, '1.5', false],
];
echo "Prices (Problem 8.1)\n";
foreach ($cases as [$name, $unit, $price, $qty, $pickup]) {
    $total = PriceCalculator::total($unit === 'kg', $price, $qty, $pickup);
    printf("  %-28s %5.2f %-4s %-6s %7d\n", $name, $qty, $unit,
        $pickup ? 'pickup' : '', $total);
}

echo "Workflow (Problem 8.2)\n";
$steps = [[OrderStatus::Queued, 25000], [OrderStatus::Finished, 25000],
    [OrderStatus::Finished, 0], [OrderStatus::Departed, 0]];
foreach ($steps as [$status, $balance]) {
    try {
        $result = OrderWorkflow::advance($status, $balance)->label();
    } catch (DomainException $e) {
        $result = $e->getMessage();
    }
    printf("  %-8s owing %6d -> %s\n", $status->label(), $balance, $result);
}
```

Keluaran aslinya identik, baris demi baris, dengan keluaran C# di atas; `diff` di antara keduanya tidak mencetak apa pun. *Controller* pesanan pada Soal 8.3 (**Kode 8.12**) memanggil `PriceCalculator::total($service->per_kg, $service->price, (string) $data['quantity'], $pickup)`, dan aturan validasi `decimal:0,2` menjamin jumlah memiliki paling banyak dua desimal sebelum kalkulator melihatnya. Di sisi C#, jaminan yang sama berasal dari aturan baru di `OrderForm.Validate` (Soal 8.7, **Kode 8.23**).

8.3.6 Penyelesaian keliru yang menggoda

PHP yang paling jelas adalah `(int) ($service->price * $quantity)`, dan hasilnya benar untuk sebagian besar jumlah. `tools/wrong_price.php` (**Kode 8.5**) mencetak apa yang terjadi untuk tiga di antaranya.

Kode 8.5 `tools/wrong_price.php`: harga yang dihitung dengan *float*, dengan tiga cara.

```php
<?php
// Tempting wrong solutions to Problem 8.1: arithmetic with floats.
// php tools/wrong_price.php

require __DIR__ . '/../app/Domain/PriceCalculator.php';

use App\Domain\PriceCalculator;

foreach (['4.02', '8.03', '4.5'] as $qty) {
    $float = 8000 * (float) $qty;                  // 8000 * 4.02 as a float
    printf("%s kg x Rp 8000: float %s, (int) %d, round %d, exact %d\n",
        $qty, var_export($float, true), (int) $float, (int) round($float),
        PriceCalculator::total(true, 8000, $qty, false));
}
```

```
4.02 kg x Rp 8000: float 32159.999999999996, (int) 32159, round 32160, exact 32160
8.03 kg x Rp 8000: float 64239.99999999999, (int) 64239, round 64240, exact 64240
4.5 kg x Rp 8000: float 36000.0, (int) 36000, round 36000, exact 36000
```

Bilangan 4,02 tidak memiliki representasi biner yang tepat, sama seperti 1/3 tidak memiliki representasi desimal yang tepat; *float* yang paling dekat dengannya sedikit lebih kecil, dan 8000×4.02 keluar sebagai 32159.999999999996 [1], [2]. *Cast* ke `int` memotong pecahannya dan menagih Rp 32.159: satu rupiah lebih sedikit daripada aplikasi C# untuk pesanan yang sama, pada jumlah yang biasa saja. Dengan `round` hasilnya benar di sini, dan itulah sebabnya kode Bab 7 bekerja pada setiap contoh di bab itu; tetapi aturan yang benar hanya karena langkah pembulatan kebetulan menyembunyikan galatnya itu rapuh, dan perubahan kelak (diskon 10%, biaya tambahan 7,5%) dapat membuat galat tersembunyi itu terlihat lagi. Menghitung dalam bilangan bulat perseratus menghilangkan masalahnya alih-alih menyembunyikannya.

8.4 Soal 8.2: Alur Pesanan di Satu Tempat

8.4.1 Soalnya

Setelah seorang pelanggan pergi membawa cuciannya tanpa membayar Rp 16.000 yang masih menjadi tagihannya, pemilik menetapkan aturan baru: *pesanan tidak boleh ditandai sudah diambil selama masih ada tagihan*. Alur lainnya tetap: antre, lalu selesai, lalu diambil, dan tidak ada apa-apa setelah diambil.

8.4.2 Memahaminya dengan tangan

Pesanan ekspres Budi Santoso di data contoh berharga Rp 36.000, dan sudah dibayar Rp 20.000. Menandainya selesai diperbolehkan. Menandainya sudah diambil harus ditolak dengan pesan yang menyebut berapa yang harus ditagih, Rp 16.000; setelah pembayaran Rp 16.000 hal itu harus diperbolehkan. Pesanan Sari Wulandari sudah selesai dan lunas, sehingga menandainya sudah diambil diperbolehkan. Pesanan yang sudah diambil tidak dapat bergerak sama sekali.

8.4.3 Merancangnya di atas kertas

Keputusannya memerlukan dua fakta, status saat ini dan sisa tagihan, dan menghasilkan status berikutnya atau sebuah alasan. Keputusan itu tidak memerlukan web maupun basis data, sehingga menjadi kelas biasa dengan satu fungsi, `Advance(status, balance)`, yang mengembalikan status berikutnya atau melempar *exception* berisi pesannya. *Controller* tetap memegang yang memang memerlukan web dan basis data: memuat pesanan beserta pembayarannya, mengisi cap waktu, menyimpan, dan menampilkan pesan.

8.4.4 Penyelesaian ASP.NET Core

Kode 8.6 adalah kelas alur kerjanya. Ekspresi `switch` C# memetakan setiap status ke status berikutnya, dan `??` mengubah "tidak ada status berikutnya" menjadi *exception* dalam satu baris.

Kode 8.6 `Domain/OrderWorkflow.cs`: alur kerja dan aturan barunya, sebagai kelas biasa.

```
using LaundryMvc.Models;

namespace LaundryMvc.Domain;

// Problem 8.2: which status may follow which, and when.
public static class OrderWorkflow
{
    public static OrderStatus? Next(OrderStatus status) => status switch
    {
        OrderStatus.Queued => OrderStatus.Finished,
        OrderStatus.Finished => OrderStatus.Departed,
        _ => null
    };

    // The next status, or an exception that explains why there is none.
    public static OrderStatus Advance(OrderStatus status, int balance)
    {
        var next = Next(status)
            ?? throw new InvalidOperationException(
                "This order has already left the shop.");
        if (next == OrderStatus.Departed && balance > 0)
        {
            throw new InvalidOperationException(
                $"Collect {Money.Rp(balance)} before handing over the laundry.");
        }
        return next;
    }
}
```

Action `Advance` di `OrdersController` (**Kode 8.10**, ditampilkan utuh pada Soal 8.3) memuat pesanan *beserta pembayarannya*, memanggil `OrderWorkflow.Advance(order.Status, order.Balance)`, menangkap *exception* dan menampilkan pesannya, dan bila tidak ada *exception* mengisi `FinishedAt` atau `DepartedAt`. Paruh kedua keluaran `CoreCheck` di atas memperlihatkan empat kasus di Subbab 8.4.2 memberi jawaban yang diharapkan.

8.4.5 Penyelesaian Laravel

Kode 8.7 adalah kelas yang sama dalam PHP. Enum status dari Bab 7 sudah tahu status berikutnya, sehingga kelas ini hanya menambahkan aturan sisa tagihan; `?? throw` bekerja di PHP 8 persis seperti di C#.

Kode 8.7 `app/Domain/OrderWorkflow.php`: alur kerja dan aturan barunya dalam PHP.

```php
<?php

namespace App\Domain;

use App\Enums\OrderStatus;
use App\Support\Money;
use DomainException;

// Problem 8.2: which status may follow which, and when.
final class OrderWorkflow
{
    // The next status, or an exception that explains why there is none.
    public static function advance(OrderStatus $status, int $balance): OrderStatus
    {
        $next = $status->next()
            ?? throw new DomainException('This order has already left the shop.');
        if ($next === OrderStatus::Departed && $balance > 0) {
            throw new DomainException('Collect ' . Money::rp($balance)
                . ' before handing over the laundry.');
        }

        return $next;
    }
}
```

Di `OrderController::advance` (**Kode 8.12**) sisa tagihan dihitung di basis data dengan `$order->total - (int) $order->payments()->sum('amount')`: satu `SELECT SUM(...)`, tanpa memuat model pembayaran. Baris-baris terakhir keluaran `core_check.php` sama persis dengan baris-baris C#, termasuk jumlah uang yang diformat di dalam pesannya.

8.4.6 Penyelesaian keliru yang menggoda

Di C#, versi pertama yang wajar memuat pesanan dengan `await db.Orders.FindAsync(id)`, seperti `Advance` di Bab 5, lalu menanyakan `order.Balance`. Kode itu dapat dikompilasi, dan kelirunya tidak akan tertangkap oleh pengujian dengan pesanan yang belum dibayar: `FindAsync` hanya memuat baris pesanan, `Payments` adalah *list* kosong, `Paid` bernilai 0, dan `Balance` sama dengan `Total`. Setiap pesanan, lunas atau belum, kini ditolak saat diambil. Perbaikannya adalah `Include(o => o.Payments)` pada penyelesaian, atau menghitung sisa tagihan di basis data seperti versi Laravel. Pelajarannya sama dengan Subbab 5.6.2: properti navigasi yang tidak dimuat bukanlah galat, melainkan *list* kosong, dan apa pun yang dihitung darinya salah tanpa suara.

8.5 Soal 8.3: Daftar Pesanan Tanpa N+1 dan Tanpa Pemborosan

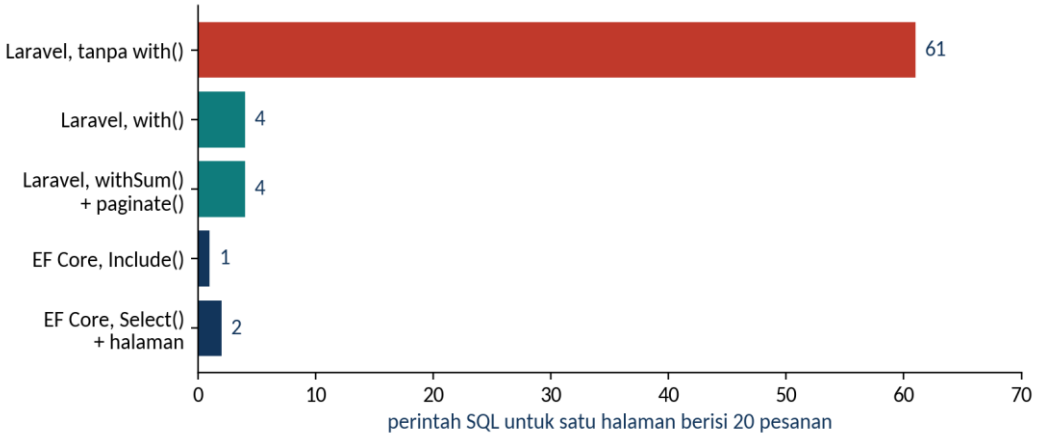
8.5.1 Soalnya

Setelah tiga bulan toko memiliki sekitar 750 pesanan, dan daftar pesanan, yang menampilkan semuanya dengan setiap pembayaran dimuat, menjadi lambat dibuka di laptop tua meja kasir. Pemilik meminta daftar itu terbuka seketika, dengan 20 pesanan per halaman, yang terbaru lebih dulu.

8.5.2 Memahaminya dengan tangan

Hitunglah perintah SQL untuk satu halaman berisi 20 pesanan, yaitu yang harus dijawab basis data. Di Laravel tanpa *eager loading*, *view* menyentuh tiga relasi per pesanan: 1 *query* untuk pesanan ditambah $3 \times 20 = 60$, seluruhnya 61. Dengan `with(['customer', 'service',`

`'payments'])`, seperti di Bab 7: 4. Di EF Core dengan tiga `Include`, seperti di Bab 5: 1 perintah dengan *join*, tetapi mengembalikan setiap kolom dari empat tabel dan setiap baris pembayaran. Satu halaman hanya memerlukan tujuh nilai per pesanan: nomor, tanggal, nama pelanggan, layanan, total, jumlah yang sudah dibayar, dan status. **Gambar 8.2** merangkum hitungannya.



Gambar 8.2 Perintah SQL yang diperlukan untuk menampilkan satu halaman berisi 20 pesanan, menurut cara daftar dimuat; hitungannya mengikuti cara setiap pemanggilan didokumentasikan memuat data.

8.5.3 Merancangnya di atas kertas

Dua gagasan memecahkannya di kedua kerangka kerja. *Paging* (pembagian halaman): minta basis data hanya satu halaman, dan jumlah seluruhnya untuk menggambar penomor halaman. *Projection* (proyeksi): minta hanya kolom yang ditampilkan halaman, dan biarkan basis data menjumlahkan pembayaran alih-alih mengirimkannya. Di EF Core, `Select` ke sebuah *record* kecil mengerjakan pemilihan kolom sekaligus penjumlahan; di Eloquent, `select([...])` memilih kolom pesanan, `with(['customer:id,name', ...])` membatasi kolom terkait, dan `withSum('payments', 'amount')` menambahkan jumlahnya sebagai kolom terhitung [3], [4].

8.5.4 Penyelesaian ASP.NET Core

Halaman menampilkan *record* `OrderRow` (**Kode 8.8**), bukan entitas.

Kode 8.8 `Models/OrderRow.cs`: persis kolom-kolom daftar pesanan.

```
namespace LaundryMvc.Models;

// Problem 8.3: exactly the columns the order list shows, nothing more.
public record OrderRow(int Id, DateTime ReceivedAt, string Customer,
                       string Service, int Total, int Paid, OrderStatus Status)
{
    public int Balance => Total - Paid;
}
```

`PagedList<T>` (**Kode 8.9**) menjalankan dua perintah: `CountAsync()` untuk jumlah seluruhnya, dan *query* dengan `Skip` dan `Take` untuk satu halaman [5]. Kelas ini menjepit nomor halaman, sehingga `?page=99` menampilkan halaman terakhir alih-alih halaman kosong.

Kode 8.9 `Domain/PagedList.cs`: satu halaman dari *query* terurut dan apa yang dibutuhkan penomor halaman.

```
using Microsoft.EntityFrameworkCore;

namespace LaundryMvc.Domain;

public interface IPaged
{
    int Page { get; }
    int PageCount { get; }
    int TotalCount { get; }
}

// Problem 8.4: one page of a query, plus what the pager needs to know.
public class PagedList<T>(List<T> items, int page, int pageCount,
                          int totalCount) : IPaged
{
    public List<T> Items { get; } = items;
    public int Page { get; } = page;
    public int PageCount { get; } = pageCount;
    public int TotalCount { get; } = totalCount;

    // The query must already be ordered, or pages may overlap or skip rows.
    public static async Task<PagedList<T>> CreateAsync(IQueryable<T> query,
                                                       int page, int pageSize)
    {
        var total = await query.CountAsync();
        var pageCount = Math.Max(1, (total + pageSize - 1) / pageSize);
        page = Math.Clamp(page, 1, pageCount);
        var items = await query.Skip((page - 1) * pageSize).Take(pageSize)
                               .ToListAsync();
        return new PagedList<T>(items, page, pageCount, total);
    }
}
```

Kode 8.10 adalah seluruh `OrdersController` bab ini; `Index` ada di bagian atas. `Select` membangun `OrderRow` dari `o.Customer!.Name`, `o.Service!.Description`, dan `o.Payments.Sum(p => p.Amount)`; EF Core menerjemahkan navigasi menjadi *join* dan `Sum` menjadi `SELECT SUM(...)` berkorelasi di dalam perintah yang sama, dan tidak ada entitas yang dilacak, karena hasilnya bukan entitas. *Query* diurutkan menurut waktu lalu menurut nomor, agar dua pesanan yang diterima pada detik yang sama tidak pernah bertukar tempat di antara halaman.

Kode 8.10 `Controllers/OrdersController.cs`: daftar pesanan sebagai proyeksi berhalaman, alur kerja Soal 8.2, dan pemeriksaan pembayaran Soal 8.5.

```
using LaundryMvc.Data;
using LaundryMvc.Domain;
using LaundryMvc.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;

namespace LaundryMvc.Controllers;

public class OrdersController(LaundryDbContext db, PriceCalculator calculator)
    : Controller
{
    // Problem 8.3: one query with only the needed columns, 20 rows a page.
    public async Task<IActionResult> Index(OrderStatus? status, int page = 1)
    {
        var query = db.Orders.AsQueryable();
        if (status is not null)
        {
            query = query.Where(o => o.Status == status);
        }
        var rows = query
            .OrderByDescending(o => o.ReceivedAt).ThenByDescending(o => o.Id)
            .Select(o => new OrderRow(o.Id, o.ReceivedAt, o.Customer!.Name,
                o.Service!.Description, o.Total,
                o.Payments.Sum(p => p.Amount), o.Status));
        ViewData["Status"] = status;
        return View(await PagedList<OrderRow>.CreateAsync(rows, page, 20));
    }

    public async Task<IActionResult> Details(int id)
    {
        var order = await db.Orders
            .Include(o => o.Customer)
            .Include(o => o.Service)
            .Include(o => o.Payments)
            .AsNoTracking()
            .FirstOrDefaultAsync(o => o.Id == id);
        return order is null ? NotFound() : View(order);
    }

    public async Task<IActionResult> Create(int? customerId)
    {
        await FillListsAsync();
        return View(new OrderForm { CustomerId = customerId ?? 0 });
    }
```

Kode 8.10 (Lanjutan) *Controllers/OrdersController.cs: daftar pesanan sebagai proyeksi berhalaman, alur kerja Soal 8.2, dan pemeriksaan pembayaran Soal 8.5.*

```
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Create(OrderForm form)
    {
        var service = await db.Services.FindAsync(form.ServiceId ?? "");
        if (form.ServiceId is not null && (service is null || !service.IsActive))
        {
            ModelState.AddModelError(nameof(form.ServiceId),
                "This service is not offered any more.");
        }
        if (form.CustomerId > 0
            && !await db.Customers.AnyAsync(c => c.Id == form.CustomerId))
        {
            ModelState.AddModelError(nameof(form.CustomerId),
                "This customer does not exist.");
        }
        if (!ModelState.IsValid || service is null)
        {
            await FillListsAsync();
            return View(form);
        }

        var quote = calculator.Calculate(service, form.Quantity, form.Pickup);
        var order = new Order
        {
            CustomerId = form.CustomerId,
            ServiceId = service.Id,
            Quantity = form.Quantity,
            Pickup = form.Pickup,
            DeliveryAddress = form.Pickup ? form.DeliveryAddress : null,
            Note = form.Note,
            Rack = form.Rack,
            Total = (int)decimal.Round(quote.Total,
                                       MidpointRounding.AwayFromZero),
            Status = OrderStatus.Queued,
            ReceivedAt = DateTime.Now
        };
        db.Orders.Add(order);
        await db.SaveChangesAsync();
        TempData["Message"] = $"Order #{order.Id} created: {Money.Rp(order.Total)}."
            + (quote.Notes.Count > 0 ? " " + string.Join(" ", quote.Notes) : "");
        return RedirectToAction(nameof(Details), new { id = order.Id });
    }
```

Kode 8.10 (Lanjutan) *Controllers/OrdersController.cs: daftar pesanan sebagai proyeksi berhalaman, alur kerja Soal 8.2, dan pemeriksaan pembayaran Soal 8.5.*

```
    // Problem 8.2: the workflow decides; the controller stamps the time.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Advance(int id)
    {
        var order = await db.Orders.Include(o => o.Payments)
                                   .FirstOrDefaultAsync(o => o.Id == id);
        if (order is null) return NotFound();
        try
        {
            order.Status = OrderWorkflow.Advance(order.Status, order.Balance);
        }
        catch (InvalidOperationException e)
        {
            TempData["Error"] = e.Message;
            return RedirectToAction(nameof(Details), new { id });
        }
        if (order.Status == OrderStatus.Finished)
        {
            order.FinishedAt = DateTime.Now;
        }
        else
        {
            order.DepartedAt = DateTime.Now;
        }
        await db.SaveChangesAsync();
        TempData["Message"] = $"Order #{id} is now {order.Status}.";
        return RedirectToAction(nameof(Details), new { id });
    }

    // Problem 8.5: a payment may not be more than what is still owed.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> AddPayment(int id, PaymentForm payment)
    {
        var balance = await db.Orders.Where(o => o.Id == id)
            .Select(o => (int?)(o.Total - o.Payments.Sum(p => p.Amount)))
            .FirstOrDefaultAsync();
        if (balance is null) return NotFound();
        if (ModelState.IsValid && payment.Amount > balance)
        {
            ModelState.AddModelError(nameof(payment.Amount),
                $"Only {Money.Rp(balance.Value)} is still owed.");
        }
        if (!ModelState.IsValid)
        {
```

Kode 8.10 (Lanjutan) *Controllers/OrdersController.cs: daftar pesanan sebagai proyeksi berhalaman, alur kerja Soal 8.2, dan pemeriksaan pembayaran Soal 8.5.*

```
            TempData["Error"] = ModelState.Values.SelectMany(v => v.Errors)
                                            .First().ErrorMessage;
            return RedirectToAction(nameof(Details), new { id });
        }
        db.Payments.Add(new Payment
        {
            OrderId = id, Amount = payment.Amount, Method = payment.Method,
            PaidAt = DateTime.Now
        });
        await db.SaveChangesAsync();
        TempData["Message"] = $"Payment of {Money.Rp(payment.Amount)} recorded.";
        return RedirectToAction(nameof(Details), new { id });
    }

    async Task FillListsAsync()
    {
        ViewBag.Customers = new SelectList(
            await db.Customers.OrderBy(c => c.Name).ToListAsync(),
            nameof(Customer.Id), nameof(Customer.Name));
        ViewBag.Services = new SelectList(
            await db.Services.Where(s => s.IsActive).OrderBy(s => s.Id)
                             .ToListAsync(),
            nameof(Service.Id), nameof(Service.Description));
    }
}
```

View-nya (**Kode 8.11**) menerima `PagedList<OrderRow>` dan diakhiri *partial* penomor halaman bersama dari Soal 8.4.

Kode 8.11 `Views/Orders/Index.cshtml`: daftar pesanan dari *record* `OrderRow`, dengan penomor halaman.

```
@model PagedList<OrderRow>
@{
    ViewData["Title"] = "Orders";
    var current = ViewData["Status"] as OrderStatus?;
}
<h1>Orders</h1>
<p class="filters">
    <a asp-action="Index" class="@(current is null ? "on" : "")">All</a>
    @foreach (var s in Enum.GetValues<OrderStatus>())
    {
        <a asp-action="Index" asp-route-status="@s"
           class="@(current == s ? "on" : "")">@s</a>
    }
    <a class="button" asp-action="Create">New order</a>
</p>
<table class="list">
    <thead>
        <tr><th>#</th><th>Received</th><th>Customer</th><th>Service</th>
            <th class="num">Total</th><th class="num">Balance</th>
            <th>Status</th></tr>
    </thead>
    <tbody>
    @foreach (var o in Model.Items)
    {
        <tr>
            <td><a asp-action="Details" asp-route-id="@o.Id">@o.Id</a></td>
            <td>@o.ReceivedAt.ToString("dd/MM HH:mm")</td>
            <td>@o.Customer</td>
            <td>@o.Service</td>
            <td class="num">@Money.Rp(o.Total)</td>
            <td class="num">@Money.Rp(o.Balance)</td>
            <td>@o.Status</td>
        </tr>
    }
    </tbody>
</table>
<partial name="_Pager" model="Model" />
```

8.5.5 Penyelesaian Laravel

Kode 8.12 adalah seluruh `OrderController` bab ini. `index` memilih enam kolom pesanan, hanya memuat `id` dan `name` pelanggan serta `id` dan `description` layanan, menambahkan `payments_sum_amount` dengan `withSum`, dan membagi halaman dengan `paginate(20)`. Relasi yang dimuat lebih awal memerlukan kolom kuncinya (`customer_id`, `service_id` pada pesanan, `id` pada model terkait), atau Eloquent tidak dapat menghubungkan barisnya; itulah sebabnya daftar `select` menyebutnya.

Kode 8.12 `app/Http/Controllers/OrderController.php`: daftar pesanan dengan halaman dan jumlah, harga dari Soal 8.1, dan alur kerja Soal 8.2.

```php
<?php

namespace App\Http\Controllers;

use App\Domain\OrderWorkflow;
use App\Domain\PriceCalculator;
use App\Enums\OrderStatus;
use App\Models\Customer;
use App\Models\Order;
use App\Models\Service;
use App\Support\Money;
use DomainException;
use Illuminate\Http\Request;
use Illuminate\Validation\Rule;

class OrderController extends Controller
{
    // Problem 8.3: four queries in all, only the needed columns, 20 a page.
    public function index(Request $request)
    {
        $status = OrderStatus::tryFrom((string) $request->query('status'));
        $orders = Order::query()
            ->select(['id', 'customer_id', 'service_id', 'total', 'status',
                'received_at'])
            ->with(['customer:id,name', 'service:id,description'])
            ->withSum('payments', 'amount')     // adds payments_sum_amount
            ->when($status, fn ($query) => $query->where('status', $status))
            ->orderByDesc('received_at')->orderByDesc('id')
            ->paginate(20)
            ->withQueryString();

        return view('orders.index', compact('orders', 'status'));
    }

    public function show(Order $order)
    {
        $order->load(['customer', 'service',
            'payments' => fn ($q) => $q->orderBy('paid_at')]);

        return view('orders.show', compact('order'));
    }
```

Kode 8.12 (Lanjutan) *app/Http/Controllers/OrderController.php: daftar pesanan dengan halaman dan jumlah, harga dari Soal 8.1, dan alur kerja Soal 8.2.*

```
    public function create(Request $request)
    {
        return view('orders.create', [
            'customers' => Customer::orderBy('name')->get(),
            'services' => Service::where('is_active', true)
                ->orderBy('id')->get(),
            'customerId' => $request->integer('customer_id'),
        ]);
    }

    public function store(Request $request)
    {
        $data = $request->validate([
            'customer_id' => ['required', 'integer', 'exists:customers,id'],
            'service_id' => ['required', 'string',
                Rule::exists('services', 'id')->where('is_active', true)],
            'quantity' => ['required', 'numeric', 'decimal:0,2', 'min:0.1',
                'max:100'],
            'delivery_address' => ['nullable', 'required_if_accepted:pickup',
                'string', 'max:200'],
            'rack' => ['nullable', 'regex:/^[A-Z]-[0-9]{2}$/'],
            'note' => ['nullable', 'string', 'max:200'],
        ], [
            'rack.regex' => 'A rack looks like B-03: a letter, a dash, two digits.',
        ]);
        $service = Service::findOrFail($data['service_id']);
        $pickup = $request->boolean('pickup');

        $order = Order::create([
            ...$data,
            'pickup' => $pickup,
            'total' => PriceCalculator::total($service->per_kg, $service->price,
                (string) $data['quantity'], $pickup),
            'status' => OrderStatus::Queued,
            'received_at' => now(),
        ]);

        return redirect()->route('orders.show', $order)->with('message',
            "Order #{$order->id} created: " . Money::rp($order->total) . '.');
    }
```

Kode 8.12 (Lanjutan) *app/Http/Controllers/OrderController.php: daftar pesanan dengan halaman dan jumlah, harga dari Soal 8.1, dan alur kerja Soal 8.2.*

```
    // Problem 8.2: the workflow decides; the controller stamps the time.
    public function advance(Order $order)
    {
        $balance = $order->total - (int) $order->payments()->sum('amount');
        try {
            $next = OrderWorkflow::advance($order->status, $balance);
        } catch (DomainException $e) {
            return back()->with('error', $e->getMessage());
        }
        $order->status = $next;
        if ($next === OrderStatus::Finished) {
            $order->finished_at = now();
        } else {
            $order->departed_at = now();
        }
        $order->save();

        return back()->with('message',
            "Order #{$order->id} is now {$next->label()}.");
    }
}
```

Di *view*, sisa tagihan menjadi `$o->total - (int) $o->payments_sum_amount`; *cast* itu penting, karena jumlah dari nol pembayaran adalah `null`, bukan 0. `paginate` menjalankan `COUNT` dan *query* halaman, dan dua pemuatan awal masing-masing menambah satu *query*: seluruhnya 4, berapa pun jumlah pesanannya.

8.5.6 Penyelesaian keliru yang menggoda

Di EF Core, kesalahan yang menggoda adalah memindahkan pekerjaan ke memori: `var all = await db.Orders.Include(...).ToListAsync();` lalu `all.Where(...).Skip(...).Take(20)`. Hasilnya halaman yang benar, dan setiap klik membaca setiap pesanan dan setiap pembayaran toko, persis kelambatan yang dikeluhkan pemilik. Semua yang ada sebelum `ToListAsync()` berjalan di basis data; semua sesudahnya berjalan di C#. Penyaringan, pengurutan, pembagian halaman, dan penjumlahan tempatnya sebelum pemanggilan itu.

Di Laravel, kesalahan yang menggoda adalah membuang `with(...)` karena halamannya "tetap jalan tanpanya". Memang jalan, dengan 61 perintah per halaman alih-alih 4; `preventLazyLoading` dari Bab 7 mengubah kesalahan itu menjadi *exception* selama pengembangan.

8.6 Soal 8.4: Pencarian Pelanggan yang Menemukan Sari

8.6.1 Soalnya

Kasir mengetik `sari` dan aplikasi Bab 5 tidak menemukan siapa pun, sedangkan aplikasi Laravel menemukan Sari Wulandari. Dengan 250 pelanggan, daftarnya juga memerlukan halaman. Pemilik meminta pencarian yang sama di keduanya: abaikan huruf besar-kecil, cocokkan nama atau telepon, 15 pelanggan per halaman, dan pertahankan pencarian ketika berpindah halaman.

8.6.2 Memahaminya dengan tangan

`Contains` di penyedia SQLite EF Core menjadi `instr(Name, 'sari') > 0`, dan `instr` membandingkan bita, sehingga `S` dan `s` berbeda. `LIKE` milik SQLite membandingkan huruf ASCII tanpa memedulikan huruf besar-kecil, sehingga `Name LIKE '%sari%'` menemukan *Sari*

Wulandari. Untuk halamannya: 250 pelanggan dengan 15 per halaman menjadi $\lceil 250/15 \rceil = 17$ halaman, yang terakhir berisi 10 pelanggan. Halaman 2 dari pencarian `an` harus tetap mencari `an`, jadi tautannya harus `?search=an&page=2`, bukan `?page=2`.

8.6.3 Merancangnya di atas kertas

Di EF Core, `EF.Functions.Like(c.Name, pattern)` meminta SQL `LIKE` secara eksplisit [6]. Di Laravel, pencarian Bab 7 sudah memakai `like`. Keduanya mendapat pembagian halaman: `PagedList` di C#, `paginate(15)` di Laravel. Keduanya mendapat penomor halaman yang menyalin *query string* saat ini dan hanya mengubah `page`.

8.6.4 Penyelesaian ASP.NET Core

Kode 8.13 adalah `CustomersController` yang baru. `Index` membangun pola `%sari%` sekali dan mengirimkannya sebagai parameter; ia mengurutkan menurut nama lalu menurut `Id`, karena dua pelanggan bisa bernama sama dan halaman memerlukan urutan yang lengkap.

Kode 8.13 `Controllers/CustomersController.cs`: pencarian yang tidak membedakan huruf besar-kecil, dengan halaman.

```
using LaundryMvc.Data;
using LaundryMvc.Domain;
using LaundryMvc.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace LaundryMvc.Controllers;

public class CustomersController(LaundryDbContext db) : Controller
{
    // Problem 8.4. GET /Customers?search=sari&page=2
    public async Task<IActionResult> Index(string? search, int page = 1)
    {
        var query = db.Customers.AsNoTracking();
        if (!string.IsNullOrWhiteSpace(search))
        {
            // LIKE ignores upper/lower case for ASCII letters in SQLite and
            // in MySQL's usual collations; Contains() would not in SQLite.
            var pattern = $"%{search.Trim()}%";
            query = query.Where(c => EF.Functions.Like(c.Name, pattern)
                                  || EF.Functions.Like(c.Phone, pattern));
        }
        ViewData["Search"] = search;
        return View(await PagedList<Customer>.CreateAsync(
            query.OrderBy(c => c.Name).ThenBy(c => c.Id), page, pageSize: 15));
    }

    // GET /Customers/Details/3
    public async Task<IActionResult> Details(int id)
    {
        var customer = await db.Customers
            .Include(c => c.Orders).ThenInclude(o => o.Service)
            .AsNoTracking()
            .FirstOrDefaultAsync(c => c.Id == id);
        return customer is null ? NotFound() : View(customer);
    }

    public IActionResult Create() => View(new Customer());
```

Kode 8.13 (Lanjutan) *Controllers/CustomersController.cs: pencarian yang tidak membedakan huruf besar-kecil, dengan halaman.*

```
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Create(
        [Bind("Name,Phone,Address")] Customer customer)
    {
        customer.Phone = IndonesianMobileAttribute.Normalize(customer.Phone);
        await CheckPhoneIsFreeAsync(customer.Phone, customer.Id);
        if (!ModelState.IsValid) return View(customer);

        db.Customers.Add(customer);
        await db.SaveChangesAsync();          // INSERT, then customer.Id is set
        TempData["Message"] = $"Customer {customer.Name} added.";
        return RedirectToAction(nameof(Details), new { id = customer.Id });
    }

    public async Task<IActionResult> Edit(int id)
    {
        var customer = await db.Customers.FindAsync(id);
        return customer is null ? NotFound() : View(customer);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(int id,
        [Bind("Id,Name,Phone,Address")] Customer input)
    {
        if (id != input.Id) return BadRequest();
        var customer = await db.Customers.FindAsync(id);
        if (customer is null) return NotFound();

        input.Phone = IndonesianMobileAttribute.Normalize(input.Phone);
        await CheckPhoneIsFreeAsync(input.Phone, id);
        if (!ModelState.IsValid) return View(input);

        customer.Name = input.Name;           // the tracked entity is changed,
        customer.Phone = input.Phone;         // so SaveChanges sends an UPDATE
        customer.Address = input.Address;     // for exactly these columns
        await db.SaveChangesAsync();
        TempData["Message"] = $"Customer {customer.Name} saved.";
        return RedirectToAction(nameof(Details), new { id });
    }
```

Kode 8.13 (Lanjutan) *Controllers/CustomersController.cs: pencarian yang tidak membedakan huruf besar-kecil, dengan halaman.*

```
    [Authorize(Roles = AppRoles.Owner)]
    public async Task<IActionResult> Delete(int id)
    {
        var customer = await db.Customers.Include(c => c.Orders)
            .AsNoTracking().FirstOrDefaultAsync(c => c.Id == id);
        return customer is null ? NotFound() : View(customer);
    }

    [HttpPost, ActionName("Delete")]
    [Authorize(Roles = AppRoles.Owner)]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var customer = await db.Customers.FindAsync(id);
        if (customer is null) return NotFound();
        if (await db.Orders.AnyAsync(o => o.CustomerId == id))
        {
            TempData["Error"] = "A customer with orders cannot be deleted.";
            return RedirectToAction(nameof(Details), new { id });
        }
        db.Customers.Remove(customer);
        await db.SaveChangesAsync();          // DELETE
        TempData["Message"] = $"Customer {customer.Name} deleted.";
        return RedirectToAction(nameof(Index));
    }

    async Task CheckPhoneIsFreeAsync(string phone, int ownId)
    {
        if (await db.Customers.AnyAsync(c => c.Phone == phone && c.Id != ownId))
        {
            ModelState.AddModelError(nameof(Customer.Phone),
                "Another customer already uses this phone number.");
        }
    }
}
```

Penomor halamannya (**Kode 8.14**) adalah *partial view* yang bekerja untuk model `IPaged` apa pun. Fungsi lokalnya, `PageLink`, menyalin setiap nilai *query string* saat ini ke sebuah kamus dan mengganti `page`, lalu `asp-all-route-data` mengubah kamus itu menjadi tautan. Pencarian, dan pada daftar pesanan penyaring status, bertahan di setiap klik.

Kode 8.14 `Views/Shared/_Pager.cshtml`: satu penomor halaman untuk setiap daftar, yang mempertahankan *query string*.

```
@model IPaged
@functions {
    // The current query string (search, status...) with another page number.
    Dictionary<string, string> PageLink(int page)
    {
        var values = Context.Request.Query
            .ToDictionary(q => q.Key, q => q.Value.ToString());
        values["page"] = page.ToString();
        return values;
    }
}
<nav class="pager">
    @if (Model.Page > 1)
    {
        <a asp-all-route-data="PageLink(Model.Page - 1)">&laquo; Previous</a>
    }
    <span>Page @Model.Page of @Model.PageCount · @Model.TotalCount in all</span>
    @if (Model.Page < Model.PageCount)
    {
        <a asp-all-route-data="PageLink(Model.Page + 1)">Next &raquo;</a>
    }
</nav>
```

8.6.5 Penyelesaian Laravel

`index` versi Laravel (**Kode 8.15**) berubah di dua tempat: pencariannya dibungkus `where(fn ($q) => ...)`, dan `get()` menjadi `paginate(15)->withQueryString()`.

Kode 8.15 `app/Http/Controllers/CustomerController.php`: pencarian yang sama dengan halaman di Laravel.

```php
<?php

namespace App\Http\Controllers;

use App\Models\Customer;
use Illuminate\Http\Request;
use Illuminate\Validation\Rule;

class CustomerController extends Controller
{
    // Problem 8.4. GET /customers?search=sari&page=2
    public function index(Request $request)
    {
        $search = trim((string) $request->query('search', ''));
        $customers = Customer::query()
            ->when($search !== '', fn ($query) => $query->where(fn ($q) => $q
                ->where('name', 'like', "%{$search}%")
                ->orWhere('phone', 'like', "%{$search}%")))
            ->orderBy('name')->orderBy('id')
            ->paginate(15)
            ->withQueryString();          // keep ?search= in the page links

        return view('customers.index', compact('customers', 'search'));
    }

    // GET /customers/3   (route model binding finds Customer 3, or 404)
    public function show(Customer $customer)
    {
        $customer->load(['orders' => fn ($q) => $q->latest('received_at'),
            'orders.service']);

        return view('customers.show', compact('customer'));
    }

    public function create()
    {
        return view('customers.create', ['customer' => new Customer()]);
    }

    public function store(Request $request)
    {
        $customer = Customer::create($request->validate($this->rules()));
```

Kode 8.15 (Lanjutan) *app/Http/Controllers/CustomerController.php: pencarian yang sama dengan halaman di Laravel.*

```
        return redirect()->route('customers.show', $customer)
            ->with('message', "Customer {$customer->name} added.");
    }

    public function edit(Customer $customer)
    {
        return view('customers.edit', compact('customer'));
    }

    public function update(Request $request, Customer $customer)
    {
        $customer->update($request->validate($this->rules($customer)));

        return redirect()->route('customers.show', $customer)
            ->with('message', "Customer {$customer->name} saved.");
    }

    public function destroy(Customer $customer)
    {
        if ($customer->orders()->exists()) {
            return back()->with('error',
                'A customer with orders cannot be deleted.');
        }
        $customer->delete();

        return redirect()->route('customers.index')
            ->with('message', "Customer {$customer->name} deleted.");
    }

    private function rules(?Customer $customer = null): array
    {
        return [
            'name' => ['required', 'string', 'min:2', 'max:80'],
            'phone' => ['required', 'string', 'max:20',
                Rule::unique('customers', 'phone')->ignore($customer)],
            'address' => ['nullable', 'string', 'max:200'],
        ];
    }
}
```

`withQueryString()` membuat setiap tautan halaman membawa *query string* saat ini, tugas yang dikerjakan `PageLink` di C#. *Partial* penomor halaman (**Kode 8.16**) memakai metode milik *paginator* sendiri, `previousPageUrl()`, `nextPageUrl()`, `currentPage()`, `lastPage()`, dan `total()`, dan mencetak baris yang sama dengan penomor halaman C# [7].

Kode 8.16 `resources/views/_pager.blade.php`: penomor halaman dalam Blade.

```
{{-- One pager for every list: $page is a LengthAwarePaginator. --}}
<nav class="pager">
    @if ($page->previousPageUrl())
        <a href="{{ $page->previousPageUrl() }}">&laquo; Previous</a>
    @endif
    <span>Page {{ $page->currentPage() }} of {{ $page->lastPage() }}
        · {{ $page->total() }} in all</span>
    @if ($page->nextPageUrl())
        <a href="{{ $page->nextPageUrl() }}">Next &raquo;</a>
    @endif
</nav>
```

8.6.6 Penyelesaian keliru yang menggoda

Mengapa ada `where(fn ($q) => ...)` tambahan di sekeliling dua syarat itu? Tanpanya SQL-nya menjadi `where name like ? or phone like ?`, yang benar selama tidak ada syarat lain yang ditambahkan. Pada hari seseorang menambahkan "hanya pelanggan gerai ini", *query*-nya menjadi `where name like ? or phone like ? and outlet_id = ?`, dan karena `and` mengikat lebih kuat daripada `or`, setiap pelanggan yang *nama*-nya cocok ditampilkan, apa pun gerainya. *Closure* itu menghasilkan tanda kurung: `where (name like ? or phone like ?) and outlet_id = ?`.

Di EF Core, kesalahan yang menggoda adalah membagi halaman tanpa `OrderBy`. SQLite lalu mengembalikan baris dalam urutan apa pun yang dianggapnya mudah, yang dapat berbeda antara *query* untuk halaman 1 dan *query* untuk halaman 2, sehingga seorang pelanggan dapat muncul di kedua halaman atau tidak di mana pun. EF Core memperingatkannya di log: *the query uses a row limiting operator ('Skip'/'Take') without an 'OrderBy' operator*. `PagedList` mendokumentasikan syarat yang sama di komentarnya.

8.7 Soal 8.5: Tanpa Kelebihan Bayar

8.7.1 Soalnya

Seorang kasir mengetik Rp 60.000 alih-alih Rp 16.000 untuk sisa tagihan Budi Santoso, dan halaman pesanan menampilkan sisa tagihan −Rp 44.000. Pemilik meminta: *pembayaran tidak boleh melebihi tagihan yang tersisa, dan pesannya harus menyebut berapa sisanya*.

8.7.2 Memahaminya dengan tangan

Pesanan Budi: total Rp 36.000, dibayar Rp 20.000, sisa Rp 16.000 menurut rumus sisa tagihan di Subbab 5.3. Pembayaran Rp 16.000 diterima dan menyisakan 0. Pembayaran Rp 20.000 ditolak dengan "Only Rp 16.000 is still owed." Pembayaran Rp 500 ditolak oleh batas minimum Rp 1.000 dari Bab 6. Pemeriksaannya harus memakai sisa tagihan pada saat pembayaran, dibaca dari basis data, tidak pernah angka yang dikirim formulir.

8.7.3 Merancangnya di atas kertas

Ini validasi yang memerlukan basis data, sehingga di ASP.NET Core tempatnya di *controller*, setelah pemeriksaan atribut `PaymentForm`; di Laravel ia menjadi aturan validasi yang batasnya dihitung sebelum `validate` dipanggil.

8.7.4 Penyelesaian ASP.NET Core

`AddPayment` di **Kode 8.10** meminta sisa tagihan kepada basis data dalam satu perintah: `Select(o => (int?)(o.Total - o.Payments.Sum(p => p.Amount)))`. *Cast* ke `int?` membuat `FirstOrDefaultAsync` mengembalikan `null` untuk pesanan yang tidak ada, yang menjadi 404. Jumlahnya baru dibandingkan dengan sisa tagihan bila atributnya lolos (`ModelState.IsValid`), agar kasir melihat satu pesan pada satu waktu, yang paling mendasar lebih dulu.

8.7.5 Penyelesaian Laravel

Kode 8.17 menghitung sisa tagihan dengan `sum('amount')` dan menaruhnya di dalam aturan itu sendiri: `"max:{$balance}"`, dengan `max` membandingkan angka karena isian itu juga memiliki aturan `integer` [8]. Pesan khusus untuk `amount.max` memakai `Money::rp`, sehingga kedua aplikasi mencetak kalimat yang sama.

Kode 8.17 `app/Http/Controllers/PaymentController.php`: pembayaran yang dibatasi sisa tagihan.

```php
<?php

namespace App\Http\Controllers;

use App\Enums\PaymentMethod;
use App\Models\Order;
use App\Support\Money;
use Illuminate\Http\Request;
use Illuminate\Validation\Rule;

class PaymentController extends Controller
{
    // Problem 8.5: a payment may not be more than what is still owed.
    public function store(Request $request, Order $order)
    {
        $balance = $order->total - (int) $order->payments()->sum('amount');
        $data = $request->validate([
            'amount' => ['required', 'integer', 'min:1000', "max:{$balance}"],
            'method' => ['required', Rule::enum(PaymentMethod::class)],
        ], [
            'amount.max' => 'Only ' . Money::rp($balance) . ' is still owed.',
        ]);
        $order->payments()->create([...$data, 'paid_at' => now()]);

        return back()->with('message',
            'Payment of ' . Money::rp((int) $data['amount']) . ' recorded.');
    }
}
```

8.7.6 Penyelesaian keliru yang menggoda

Membandingkan jumlah dengan *total* pesanan alih-alih sisa tagihannya menerima Rp 36.000 di atas Rp 20.000 yang sudah dibayar. Membandingkan dengan sisa tagihan yang dikirim sebagai isian tersembunyi formulir membiarkan siapa pun yang menyunting isian itu membayar berapa saja. Dan masih ada masalah yang lebih halus di kedua penyelesaian: dua kasir yang mencatat pembayaran terakhir untuk pesanan yang sama pada detik yang sama dapat sama-sama lolos pemeriksaan. SQLite menjalankan operasi tulis satu per satu, dan meja kasir laundry jarang memiliki dua kasir untuk satu pesanan, sehingga penyelesaian ini menerima risiko itu; sistem yang sibuk akan menjalankan pemeriksaan dan penyisipan dalam satu transaksi dengan kunci, atau memeriksa ulang sisa tagihan setelah menyimpan.

8.8 Soal 8.6: Laporan Pendapatan Harian

8.8.1 Soalnya

Setiap malam pemilik mencocokkan uang tunai di laci, laporan QRIS, dan daftar transfer bank dengan yang dicatat sistem. Ia meminta laporan: *uang yang diterima per hari, dipisah menurut cara bayar, untuk satu minggu yang dipilih*.

8.8.2 Memahaminya dengan tangan

Di data contoh ada dua pembayaran: Rp 36.000 lewat QRIS kemarin pukul 08.00 (Sari Wulandari) dan Rp 20.000 tunai hari ini pukul 07.30 (Budi Santoso). Untuk rentang bawaan, tujuh hari terakhir sampai hari ini, laporannya berisi dua baris, kemarin dengan QRIS Rp 36.000 dan hari ini

dengan tunai Rp 20.000, masing-masing satu pembayaran, serta baris total Rp 56.000 dari 2 pembayaran. Hari sebuah pembayaran adalah tanggalnya menurut waktu setempat; pembayaran pukul 00.30 termasuk hari yang baru.

8.8.3 Merancangnya di atas kertas

Biarkan basis data mengelompokkan dan menjumlahkan: satu baris per hari dan cara bayar. Lalu ubah baris-baris itu, paling banyak beberapa puluh, menjadi satu baris per hari dengan satu kolom per cara bayar, di memori. Rentang tanggalnya adalah *dari hari pertama pukul tengah malam, sampai tetapi tidak termasuk hari setelah hari terakhir*, yang mencakup setiap saat di hari terakhir tanpa harus menulis 23.59.59.

8.8.4 Penyelesaian ASP.NET Core

Satu baris laporan adalah *record* kecil (**Kode 8.18**).

Kode 8.18 `Models/DailyRevenueRow.cs`: satu baris laporan.

```
namespace LaundryMvc.Models;

// Problem 8.6: one line of the daily revenue report.
public record DailyRevenueRow(DateTime Day, int Cash, int Qris, int Transfer,
                              int Payments)
{
    public int Total => Cash + Qris + Transfer;
}
```

`ReportsController` (**Kode 8.19**) hanya untuk pemilik. Ia menerima rentang sebagai nilai `DateOnly?`, yang dibaca *model binding* dari `?from=2026-09-01&to=2026-09-07`, secara bawaan memakai tujuh hari terakhir, dan menukar keduanya bila datang dalam urutan terbalik. `GroupBy(p => new { Day = p.PaidAt.Date, p.Method })` diterjemahkan menjadi `GROUP BY` pada bagian tanggal cap waktu dan kolom cara bayar; `Count()` dan `Sum()` menjadi `COUNT(*)` dan `SUM("Amount")` [9]. `GroupBy` kedua, di memori, membangun baris-baris laporan.

Kode 8.19 `Controllers/ReportsController.cs`: uang yang diterima per hari dan cara bayar, dikelompokkan di basis data.

```
using LaundryMvc.Data;
using LaundryMvc.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace LaundryMvc.Controllers;

// Problem 8.6: money received per day and per payment method.
[Authorize(Roles = AppRoles.Owner)]
public class ReportsController(LaundryDbContext db) : Controller
{
    // GET /Reports/Daily?from=2026-09-01&to=2026-09-07
    public async Task<IActionResult> Daily(DateOnly? from, DateOnly? to)
    {
        var last = to ?? DateOnly.FromDateTime(DateTime.Today);
        var first = from ?? last.AddDays(-6);
        if (first > last) (first, last) = (last, first);
        var start = first.ToDateTime(TimeOnly.MinValue);
        var end = last.AddDays(1).ToDateTime(TimeOnly.MinValue);  // exclusive

        // One row per day and method, computed by the database.
        var groups = await db.Payments
            .Where(p => p.PaidAt >= start && p.PaidAt < end)
            .GroupBy(p => new { Day = p.PaidAt.Date, p.Method })
            .Select(g => new
            {
                g.Key.Day, g.Key.Method,
                Count = g.Count(), Amount = g.Sum(p => p.Amount)
            })
            .ToListAsync();

        // Turned into one line per day, with a column per method.
        var rows = groups
            .GroupBy(g => g.Day)
            .OrderBy(d => d.Key)
            .Select(d => new DailyRevenueRow(d.Key,
                Cash: d.Where(g => g.Method == PaymentMethod.Cash)
                       .Sum(g => g.Amount),
                Qris: d.Where(g => g.Method == PaymentMethod.Qris)
                       .Sum(g => g.Amount),
                Transfer: d.Where(g => g.Method == PaymentMethod.Transfer)
                           .Sum(g => g.Amount),
                Payments: d.Sum(g => g.Count)))
            .ToList();
```

Kode 8.19 (Lanjutan) *Controllers/ReportsController.cs: uang yang diterima per hari dan cara bayar, dikelompokkan di basis data.*

```
        ViewData["From"] = first.ToString("yyyy-MM-dd");
        ViewData["To"] = last.ToString("yyyy-MM-dd");
        return View(rows);
    }
}
```

View-nya (**Kode 8.20**) mencetak baris-baris itu dan satu baris total.

Kode 8.20 `Views/Reports/Daily.cshtml`: laporan pendapatan harian.

```
@model List<DailyRevenueRow>
@{
    ViewData["Title"] = "Daily revenue";
}
<h1>Daily revenue</h1>
<form method="get" class="search">
    <input type="date" name="from" value="@ViewData["From"]" />
    <input type="date" name="to" value="@ViewData["To"]" />
    <button type="submit">Show</button>
</form>
<table class="list">
    <thead>
        <tr><th>Day</th><th class="num">Cash</th><th class="num">QRIS</th>
            <th class="num">Transfer</th><th class="num">Total</th>
            <th class="num">Payments</th></tr>
    </thead>
    <tbody>
    @foreach (var r in Model)
    {
        <tr>
            <td>@r.Day.ToString("ddd dd/MM/yyyy")</td>
            <td class="num">@Money.Rp(r.Cash)</td>
            <td class="num">@Money.Rp(r.Qris)</td>
            <td class="num">@Money.Rp(r.Transfer)</td>
            <td class="num"><strong>@Money.Rp(r.Total)</strong></td>
            <td class="num">@r.Payments</td>
        </tr>
    }
    <tr class="total"><th>All days</th>
        <td class="num">@Money.Rp(Model.Sum(r => r.Cash))</td>
        <td class="num">@Money.Rp(Model.Sum(r => r.Qris))</td>
        <td class="num">@Money.Rp(Model.Sum(r => r.Transfer))</td>
        <td class="num">@Money.Rp(Model.Sum(r => r.Total))</td>
        <td class="num">@Model.Sum(r => r.Payments)</td></tr>
    </tbody>
</table>
```

8.8.5 Penyelesaian Laravel

Kode 8.21 menulis pengelompokannya dengan `selectRaw('date(paid_at) as day, method, count(*) as payments, sum(amount) as amount')` dan `groupByRaw('date(paid_at), method')` [10]. `date(...)` ada di SQLite dan MySQL dengan makna yang sama, sehingga laporan ini berjalan di keduanya. `toBase()` mengembalikan baris biasa alih-alih model `Payment`, karena baris-baris itu bukan pembayaran dan tidak boleh melewati *cast* milik model. Metode koleksi `groupBy('day')`, `sortKeys()`, dan `where('method', 'cash')->sum('amount')` lalu membangun baris-barisnya, seperti yang dilakukan LINQ di C#.

Kode 8.21 `app/Http/Controllers/ReportController.php`: laporan yang sama dengan *query builder* dan koleksi.

```php
<?php

namespace App\Http\Controllers;

use App\Models\Payment;
use Illuminate\Http\Request;
use Illuminate\Support\Carbon;

// Problem 8.6: money received per day and per payment method.
class ReportController extends Controller
{
    // GET /reports/daily?from=2026-09-01&to=2026-09-07
    public function daily(Request $request)
    {
        $request->validate(['from' => ['nullable', 'date'],
            'to' => ['nullable', 'date']]);
        $last = ($request->date('to') ?? today())->startOfDay();
        $first = ($request->date('from') ?? $last->copy()->subDays(6))
            ->startOfDay();
        if ($first->gt($last)) {
            [$first, $last] = [$last, $first];
        }

        // One row per day and method, computed by the database.
        $groups = Payment::query()
            ->where('paid_at', '>=', $first)
            ->where('paid_at', '<', $last->copy()->addDay())
            ->selectRaw('date(paid_at) as day, method, count(*) as payments,'
                . ' sum(amount) as amount')
            ->groupByRaw('date(paid_at), method')
            ->toBase()                    // plain rows, not Payment models
            ->get();

        // Turned into one line per day, with a column per method.
        $rows = $groups->groupBy('day')->sortKeys()
            ->map(function ($day, $date) {
                $cash = (int) $day->where('method', 'cash')->sum('amount');
                $qris = (int) $day->where('method', 'qris')->sum('amount');
                $transfer = (int) $day->where('method', 'transfer')
                    ->sum('amount');
```

Kode 8.21 (Lanjutan) *app/Http/Controllers/ReportController.php: laporan yang sama dengan query builder dan koleksi.*

```php
                return (object) ['day' => Carbon::parse($date),
                    'cash' => $cash, 'qris' => $qris, 'transfer' => $transfer,
                    'total' => $cash + $qris + $transfer,
                    'payments' => (int) $day->sum('payments')];
            })->values();

        return view('reports.daily', ['rows' => $rows,
            'from' => $first->toDateString(), 'to' => $last->toDateString()]);
    }
}
```

Kode 8.22 `resources/views/reports/daily.blade.php`: laporan dalam Blade.

```
@extends('layouts.app')
@use('App\Support\Money')

@section('title', 'Daily revenue')

@section('content')
<h1>Daily revenue</h1>
<form method="get" class="search">
    <input type="date" name="from" value="{{ $from }}">
    <input type="date" name="to" value="{{ $to }}">
    <button type="submit">Show</button>
</form>
<table class="list">
    <thead>
        <tr><th>Day</th><th class="num">Cash</th><th class="num">QRIS</th>
            <th class="num">Transfer</th><th class="num">Total</th>
            <th class="num">Payments</th></tr>
    </thead>
    <tbody>
    @foreach ($rows as $r)
        <tr>
            <td>{{ $r->day->format('D d/m/Y') }}</td>
            <td class="num">{{ Money::rp($r->cash) }}</td>
            <td class="num">{{ Money::rp($r->qris) }}</td>
            <td class="num">{{ Money::rp($r->transfer) }}</td>
            <td class="num"><strong>{{ Money::rp($r->total) }}</strong></td>
            <td class="num">{{ $r->payments }}</td>
        </tr>
    @endforeach
    <tr class="total"><th>All days</th>
        <td class="num">{{ Money::rp($rows->sum('cash')) }}</td>
        <td class="num">{{ Money::rp($rows->sum('qris')) }}</td>
        <td class="num">{{ Money::rp($rows->sum('transfer')) }}</td>
        <td class="num">{{ Money::rp($rows->sum('total')) }}</td>
        <td class="num">{{ $rows->sum('payments') }}</td></tr>
    </tbody>
</table>
@endsection
```

View Blade-nya (**Kode 8.22**) baris demi baris sama dengan *view* Razor pada **Kode 8.20**. Sampai Bab 10 menambahkan fitur masuk ke aplikasi Laravel, laporannya sama terbukanya dengan halaman staf lainnya, satu alasan lagi untuk sementara menjalankan versi itu hanya di komputer lokal.

8.8.6 Penyelesaian keliru yang menggoda

Mengelompokkan menurut `PaidAt` alih-alih `PaidAt.Date` menghasilkan satu baris per pembayaran, karena tidak ada dua pembayaran yang berbagi detik yang sama. Menyimpan jumlah uang sebagai `decimal`, seperti harga layanan, akan membuat `Sum` gagal di SQLite saat berjalan dengan EF Core, yang tidak dapat menerjemahkan penjumlahan desimal untuk penyedia itu (Subbab 5.3); laporan ini bekerja karena Bab 5 menyimpan jumlah uang sebagai rupiah utuh. Dan di Laravel, lupa mengatur zona waktu aplikasi ke `Asia/Jakarta` menggeser setiap batas hari ke pukul 07.00 waktu setempat, sehingga pembayaran pagi sebelum pukul tujuh muncul di hari sebelumnya.

8.9 Soal 8.7: Rak untuk Setiap Kantong yang Selesai

8.9.1 Soalnya

Kantong yang sudah selesai menunggu di rak bernomor, dan pelanggan menunggu sementara petugas cuci mencari kantongnya. Pemilik meminta isian *rack* (rak) pada setiap pesanan, seperti `B-03`: satu huruf kapital, tanda hubung, dan dua digit, tidak wajib, diisi di meja kasir dan ditampilkan di halaman pesanan.

8.9.2 Memahaminya dengan tangan

`B-03` dan `A-10` sah; `b-03`, `B3`, dan `B-003` tidak. Pesanan yang sudah ada tidak memiliki rak, sehingga kolom barunya harus boleh kosong; bila tidak, migrasinya harus mengarang rak untuk 750 pesanan lama.

8.9.3 Merancangnya di atas kertas

Perubahan skema adalah migrasi di kedua kerangka kerja, ditambah properti model, isian formulir dengan aturan validasi, dan satu baris di halaman pesanan. Urutan kerjanya penting: model dan migrasi lebih dulu, lalu formulir, lalu halaman.

8.9.4 Penyelesaian ASP.NET Core

`Order` mendapat `public string? Rack { get; set; }` dan `LaundryDbContext` mendapat `e.Property(o => o.Rack).HasMaxLength(8)`. `OrderForm` (**Kode 8.23**) mendapat properti itu dengan ekspresi reguler, dan metode `Validate`-nya mendapat aturan dua desimal dari Soal 8.1.

Kode 8.23 `Models/OrderForm.cs` dengan rak dan aturan dua desimal.

```
using System.ComponentModel.DataAnnotations;

namespace LaundryMvc.Models;

// What the counter form may send: nothing about Total, Status or dates.
public class OrderForm : IValidatableObject
{
    [Range(1, int.MaxValue, ErrorMessage = "Choose a customer.")]
    [Display(Name = "Customer")]
    public int CustomerId { get; set; }

    [Required(ErrorMessage = "Choose a service.")]
    [Display(Name = "Service")]
    public string? ServiceId { get; set; }

    [Range(typeof(decimal), "0.1", "100",
        ParseLimitsInInvariantCulture = true,
        ErrorMessage = "The quantity must be between 0.1 and 100.")]
    [Display(Name = "Quantity (kg, items or pairs)")]
    public decimal Quantity { get; set; } = 1;

    [Display(Name = "Pickup and delivery")]
    public bool Pickup { get; set; }

    [StringLength(200)]
    [Display(Name = "Delivery address")]
    public string? DeliveryAddress { get; set; }

    [StringLength(200)]
    public string? Note { get; set; }

    [RegularExpression("^[A-Z]-[0-9]{2}$",
        ErrorMessage = "A rack looks like B-03: a letter, a dash, two digits.")]
    public string? Rack { get; set; }

    // A rule about two fields together: delivery needs an address.
    public IEnumerable<ValidationResult> Validate(ValidationContext context)
    {
        if (decimal.Round(Quantity, 2) != Quantity)
        {
            yield return new ValidationResult(
                "Use at most two decimals, for example 4.25.",
                [nameof(Quantity)]);
        }
        if (Pickup && string.IsNullOrWhiteSpace(DeliveryAddress))
        {
```

Kode 8.23 (Lanjutan) *Models/OrderForm.cs dengan rak dan aturan dua desimal.*

```
            yield return new ValidationResult(
                "Pickup and delivery needs a delivery address.",
                [nameof(DeliveryAddress)]);
        }
    }
}
```

Lalu `dotnet ef migrations add AddRackToOrders` membandingkan model dengan *snapshot* dan menulis migrasi yang `Up`-nya memanggil `migrationBuilder.AddColumn<string>(name: "Rack", table: "Orders", type: "TEXT", maxLength: 8, nullable: true)` dan `Down`-nya memanggil `DropColumn` [11]. `dotnet ef database update`, atau `MigrateAsync()` saat aplikasi dimulai dari Bab 6, menerapkannya. `Create` menyalin `form.Rack` ke pesanan, dan halaman rincian mencetaknya.

8.9.5 Penyelesaian Laravel

Kode 8.24 adalah migrasinya, ditulis dengan tangan; `Schema::table` mengubah tabel yang sudah ada, sedangkan `Schema::create` membuat tabel baru [12]. `'rack'` ditambahkan ke daftar `#[Fillable]` model, aturan *store* mendapat `'rack' => ['nullable', 'regex:/^[A-Z]-[0-9]{2}$/']` dengan pesan yang sama seperti C#, dan formulir serta halaman pesanan mendapat satu isian dan satu baris.

Kode 8.24 `2026_09_20_000001_add_rack_to_orders_table.php`: kolom rak, ditambahkan dan dihapus.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

// Problem 8.7: where the finished bag waits, for example rack "B-03".
return new class extends Migration
{
    public function up(): void
    {
        Schema::table('orders', function (Blueprint $table) {
            $table->string('rack', 8)->nullable()->after('note');
        });
    }

    public function down(): void
    {
        Schema::table('orders', function (Blueprint $table) {
            $table->dropColumn('rack');
        });
    }
};
```

8.9.6 Penyelesaian keliru yang menggoda

Jalan pintas yang menggoda adalah menyunting migrasi yang membuat tabel pesanan, menambahkan kolomnya di sana, lalu menjalankan migrasi lagi. Di laptop sendiri, `migrate:fresh` atau menghapus `laundry.db` membuatnya tampak baik-baik saja. Di server toko, tempat migrasi aslinya sudah dijalankan, tidak terjadi apa-apa: Laravel dan EF Core sama-sama melewati migrasi yang tercatat sudah diterapkan, sehingga kolomnya tidak pernah muncul dan pesanan pertama dengan rak gagal. Migrasi yang sudah meninggalkan komputer Anda adalah sejarah; perubahan selalu berupa migrasi baru.

8.10 Soal-Soal Berdampingan

Tabel 8.1 mengumpulkan ketujuh penyelesaian.

Tabel 8.1 Tujuh soal, penyelesaiannya di kedua kerangka kerja, dan pelajaran dari masing-masing.

Soal	ASP.NET Core	Laravel	Pelajaran
8.1 harga	`PriceCalculator`, `decimal`	`PriceCalculator`, bilangan bulat perseratus	aturan yang sama, terbukti setara pada kasus tetap
8.2 alur kerja	`OrderWorkflow.Advance` + `Include(Payments)`	`OrderWorkflow::advance` + `sum('amount')`	kelas biasa untuk aturan; muat apa yang Anda hitung
8.3 daftar pesanan	`Select` ke `OrderRow`, `PagedList`	`select`, `with` pada kolom, `withSum`, `paginate`	bagi halaman dan proyeksikan di basis data
8.4 pencarian	`EF.Functions.Like`, `OrderBy` + `ThenBy`	`where` berkelompok, `withQueryString`	ketahui apa yang dilakukan fungsi untai penyedia Anda
8.5 kelebihan bayar	sisa tagihan dalam satu *query*, galat `ModelState`	aturan `"max:{$balance}"`	batas dari basis data, jangan dari formulir
8.6 laporan	`GroupBy` tanggal dan cara bayar, lalu pivot	`selectRaw` + `groupByRaw`, koleksi	agregasi di SQL, bentuk di memori
8.7 rak	model + `migrations add`	migrasi `Schema::table` tulisan tangan	perubahan skema selalu migrasi baru

Bacalah kolom terakhir dari atas ke bawah, dan tema bab ini menjadi jelas. Kedua kerangka kerja berbeda dalam kosakata, dalam seberapa banyak yang mereka bangkitkan, dan dalam seberapa banyak yang mereka periksa saat kompilasi; keputusan-keputusan yang membuat aplikasi benar sama di keduanya.

8.11 Profitina dalam Praktik

Dua kebiasaan dalam bab ini adalah praktik sehari-hari di Profitina. Yang pertama adalah membuktikan bahwa dua implementasi sepakat. Dasbor riwayat Profitina menampilkan skor mingguan per merek yang dihitung dari tabel yang terus bertambah, dan angka-angkanya diperiksa terhadap perhitungan mandiri atas baris-baris yang sama sebelum sebuah perubahan dirilis, seperti `CoreCheck` dan `core_check.php` yang harus mencetak baris-baris identik. Yang kedua adalah gerbang rilis: seluruh rangkaian uji otomatis, lebih dari seribu pengujian dalam Python ditambah penjaga JavaScript untuk antarmuka, harus lolos sebelum rilis, dan rilis bertahap baru dilaporkan berhasil setelah versi yang diterapkan menjawab *endpoint* kesehatannya. Ketujuh soal di bab ini masing-masing akan menjadi pengujian dalam rangkaian semacam itu: sembilan kasus harga, empat kasus alur kerja, jumlah *query* daftar pesanan, pencarian `sari`, kelebihan bayar yang ditolak, dua baris laporan, dan migrasi yang dijalankan pada salinan basis data produksi.

8.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Memindahkan aturan dengan membaca kodenya, bukan kebutuhannya.** Kode harga Bab 7 adalah pemindahan yang setia dengan *float*; hasilnya cocok hanya karena `round`. Pindahkan aturannya, lalu buktikan kesetaraannya pada kasus-kasus tetap.
- **Menghitung dari relasi yang tidak dimuat.** Di EF Core, koleksi yang tidak dimuat bernilai kosong; di Laravel ia dimuat secara malas, satu *query* per model. Bagaimanapun, ketahui apa yang dimuat sebelum menghitung darinya.

- **Tanpa sengaja memindahkan pekerjaan ke memori.** Di EF Core, `ToList` sebelum `Where`, `Skip`, atau `Sum` membaca seluruh tabel. Pertahankan *query* sebagai `IQueryable` sampai langkah terakhir.
- **Membagi halaman tanpa urutan yang lengkap.** Urutkan menurut kolom unik di akhir, seperti `Id`, atau baris-baris dapat berulang atau hilang di antara halaman.
- **`orWhere` yang tidak dikelompokkan.** Bungkus pilihan-pilihan dalam *closure* agar menjadi satu syarat bertanda kurung.
- **Memercayai angka dari formulir.** Sisa tagihan, harga, dan total dibaca dari basis data di server, tidak pernah diambil dari isian tersembunyi.
- **Mengelompokkan menurut cap waktu alih-alih tanggalnya,** dan melupakan zona waktu aplikasi.
- **Menyunting migrasi yang sudah dijalankan.** Tambahkan migrasi baru.

8.13 Ringkasan Bab

- Soal kerangka kerja dipecahkan dalam enam langkah: nyatakan dengan kata-kata toko, kerjakan contoh dengan tangan, tentukan lapisannya, tulis di satu kerangka kerja, pindahkan dan bandingkan, lalu coba patahkan.
- Aturan yang tidak memerlukan web maupun basis data tempatnya di kelas biasa yang dapat dipanggil kedua kerangka kerja dan diperiksa program konsol; aturan harga dan alur pesanan terbukti identik di C# dan PHP oleh dua program yang mencetak baris-baris yang sama.
- Uang dan jumlah paling aman sebagai bilangan bulat: rupiah utuh di basis data dan bilangan bulat perseratus dalam aritmetika PHP; *float* dapat kehilangan satu rupiah pada jumlah yang biasa saja, seperti 4,02 kg.
- Daftar dibagi halaman dan diproyeksikan di basis data: dua perintah di EF Core, empat di Eloquent, alih-alih 61 untuk halaman yang dimuat secara malas.
- Fungsi untai berbeda antarpenyedia: `Contains` membedakan huruf besar-kecil di SQLite, `LIKE` tidak.
- Batas yang bergantung pada data tersimpan, seperti sisa tagihan pesanan, dihitung di server pada saat permintaan.
- Laporan mengelompokkan dan menjumlahkan di SQL dan dibentuk di memori; hari adalah rentang dari tengah malam ke tengah malam menurut zona waktu toko.
- Perubahan skema adalah migrasi baru, dibangkitkan `dotnet ef` atau ditulis untuk Artisan, tidak pernah suntingan terhadap migrasi lama.

8.14 Soal Latihan

1. Dengan tangan, hitung dengan Persamaan (8.1) dan Persamaan (8.2) total untuk *Curtain* (Rp 15.000 per kg) sebanyak 2,5 kg dengan antar-jemput, dan untuk *Helmet cleaning* (Rp 30.000 per potong) sebanyak 3,2 potong tanpa antar-jemput. Tambahkan kedua kasus ke `CoreCheck` dan `core_check.php` dan periksa bahwa keluarannya tetap sepakat.
2. Pemilik memberlakukan diskon 10% untuk pesanan 10 kg atau lebih pada layanan per kg, dibulatkan ke bawah ke rupiah utuh. Perluas kedua kelas `PriceCalculator` dan sebutkan total setelah diskon untuk 12 kg *Wash, dry & fold*.
3. Alur kerja mendapat status keempat, *Cancelled*, yang hanya diperbolehkan dari *Queued*. Ubah kedua kelas alur kerja. Apa yang juga harus berubah di basis data, di EF Core dan di Laravel?

4. Untuk satu halaman berisi 20 pesanan, berapa perintah SQL yang dikirim daftar pesanan Laravel bila `withSum` diganti `with('payments')` dan *view* menjumlahkan `$o->payments`? Dan bila *view* memakai `$o->payments()->sum('amount')`?
5. Perluas laporan harian dengan parameter `method` yang membatasinya pada satu cara bayar, di kedua kerangka kerja. Validasikan parameternya.
6. Tulislah, di kedua kerangka kerja, *query* untuk laporan baru: sepuluh pelanggan dengan total sisa tagihan terbesar, berisi nama, telepon, dan jumlahnya. Bagian mana yang berjalan di basis data?

8.15 Penyelesaian Soal Latihan

1. *Curtain*: $h = 250$, per kg, sehingga $b = \max(250,300) = 300$ dan $S = \lfloor(15000 \cdot 300 + 50)/100\rfloor = 45000$; antar-jemput menambah Rp 10.000 karena $45000 < 100000$: total Rp 55.000. *Helmet*: $h = 320$, per potong, sehingga $b = 100 \cdot \lceil 3.2\rceil = 400$ dan $S = \lfloor(30000 \cdot 400 + 50)/100\rfloor = 120000$; tanpa antar-jemput: total Rp 120.000. Di `CoreCheck` tambahkan `("Curtain (per kg)", 15000, 2.5m, true)` dan `("Helmet cleaning (per item)", 30000, 3.2m, false)`; di `core_check.php` tambahkan `['Curtain', 'kg', 15000, '2.5', true]` dan `['Helmet cleaning', 'item', 30000, '3.2', false]`. Keduanya mencetak 55000 dan 120000, dan `diff` tetap diam.

2. Setelah subtotal, kurangi `subtotal / 10` untuk layanan per kg bila jumlah yang ditagih paling sedikit 10 kg. Di C#, `if (service.PerKg && billed >= 10) subtotal -= decimal.Floor(subtotal / 10);`. Di PHP, `if ($perKg && $billed >= 1000) { $subtotal -= intdiv($subtotal, 10); }` (1000 perseratus = 10 kg). Untuk 12 kg *Wash, dry & fold*: $S = 12 \times 8000 = 96000$, diskon $\lfloor 96000/10\rfloor = 9600$, total Rp 86.400. Tambahkan kasusnya ke kedua program pemeriksa; diskon adalah jenis perubahan yang menurut Subbab 8.3.6 dapat memperlihatkan galat *float*, dan dengan aritmetika bilangan bulat hal itu tidak mungkin terjadi.

3. Tambahkan `Cancelled` ke enum C# dan `case Cancelled = 'cancelled';` ke enum PHP. Di C#, `OrderWorkflow` mendapat metode `Cancel(OrderStatus status)` yang mengembalikan `Cancelled` dari `Queued` dan melempar *exception* dalam keadaan lain; `Next` tidak berubah, sehingga alur normal tidak pernah mencapai `Cancelled`. Di PHP, metode statis `cancel(OrderStatus $status)` melakukan hal yang sama, dan `next()` di enum mengembalikan `null` untuk `Cancelled`. Basis data tidak memerlukan perubahan struktur, karena keduanya menyimpan status sebagai teks, dengan ruang untuk 12 karakter; tetapi `HasConversion<string>()` di C# berarti kode lama yang membaca nilai baru akan gagal, jadi terapkan kodenya sebelum membatalkan pesanan pertama. Cap waktu `CancelledAt` yang baru, bila diinginkan, adalah migrasi di keduanya.

4. Dengan `with('payments')`: `COUNT`, pesanan, pelanggan, layanan, dan pembayaran dari 20 pesanan itu, 5 perintah, tetapi setiap baris pembayaran dikirimkan. Dengan `$o->payments()->sum('amount')` di *view*: *metode* relasi selalu membangun *query* baru, sehingga 4 perintah ditambah satu `SUM` per pesanan, seluruhnya 24; `preventLazyLoading` tidak menangkap hal ini, karena tidak ada yang dimuat secara malas. `withSum` mengerjakan hal yang sama di dalam *query* pesanan.

5. Di C#, tambahkan `PaymentMethod? method` ke `Daily` dan `if (method is not null) query = query.Where(p => p.Method == method);` sebelum `GroupBy`; *model binding* menolak nama yang tidak dikenal dan membiarkan `method` bernilai null, jadi validasikan dengan memeriksa `ModelState.IsValid` dan menampilkan pesan. Di Laravel, tambahkan `'method' => ['nullable', Rule::enum(PaymentMethod::class)]` ke validasi dan `->when($request-`

`>filled('method'), fn ($q) => $q->where('method', $request->input('method')))` ke *query*. Penomor halaman tidak terpengaruh, karena laporan tidak memiliki halaman.

6. Sisa tagihan per pelanggan adalah jumlah total dikurangi jumlah pembayaran atas pesanan-pesanan pelanggan itu. Di C#: `db.Customers.Select(c => new { c.Name, c.Phone, Owed = c.Orders.Sum(o => o.Total) - c.Orders.SelectMany(o => o.Payments).Sum(p => p.Amount) }).Where(r => r.Owed > 0).OrderByDescending(r => r.Owed).Take(10)`; EF Core menerjemahkan semuanya, termasuk kedua penjumlahan, penyaringan, pengurutan, dan batasnya, menjadi satu perintah SQL. Di Laravel, `withSum('orders', 'total')` menangani penjumlahan pertama, tetapi pembayaran berjarak dua relasi, sehingga kedua penjumlahan ditulis sebagai *subquery*: `$sums = Customer::select('name', 'phone')->selectSub(Order::selectRaw('coalesce(sum(total), 0)')->whereColumn('customer_id', 'customers.id'), 'billed')->selectSub(Payment::join('orders', 'orders.id', '=', 'payments.order_id')->selectRaw('coalesce(sum(amount), 0)')->whereColumn('orders.customer_id', 'customers.id'), 'paid');` lalu `DB::query()->fromSub($sums, 'c')->selectRaw('name, phone, billed - paid as owed')->whereRaw('billed - paid > 0')->orderByDesc('owed')->limit(10)->get()`. Membungkus *query* pertama dengan `fromSub` memungkinkan *query* luar menyaring dan mengurutkan menurut kolom terhitung. Di kedua kerangka kerja semuanya berjalan di basis data dan hanya sepuluh baris yang kembali.

Daftar Pustaka

[1] D. Goldberg, “What every computer scientist should know about floating-point arithmetic,” *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48, 1991.

[2] The PHP Group, “Floating point numbers,” PHP Manual, diakses Sep. 2026. [Daring]. Tersedia: https://www.php.net/manual/en/language.types.float.php

[3] Microsoft, “Efficient querying,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/ef/core/performance/efficient-querying

[4] Laravel, “Eloquent: relationships (eager loading and aggregating related models), Laravel 13.x,” diakses Sep. 2026. [Daring]. Tersedia: https://laravel.com/docs/13.x/eloquent-relationships

[5] Microsoft, “Pagination,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/ef/core/querying/pagination

[6] Microsoft, “Function mappings of the SQLite EF Core provider,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/ef/core/providers/sqlite/functions

[7] Laravel, “Database: pagination, Laravel 13.x,” diakses Sep. 2026. [Daring]. Tersedia: https://laravel.com/docs/13.x/pagination

[8] Laravel, “Validation: available validation rules, Laravel 13.x,” diakses Sep. 2026. [Daring]. Tersedia: https://laravel.com/docs/13.x/validation#available-validation-rules

[9] Microsoft, “Complex query operators,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/ef/core/querying/complex-query-operators

[10] Laravel, “Database: query builder, Laravel 13.x,” diakses Sep. 2026. [Daring]. Tersedia: https://laravel.com/docs/13.x/queries

[11] Microsoft, “Migrations overview,” Microsoft Learn, diakses Sep. 2026. [Daring]. Tersedia: https://learn.microsoft.com/ef/core/managing-schemas/migrations/

[12] Laravel, “Database: migrations, Laravel 13.x,” diakses Sep. 2026. [Daring]. Tersedia: https://laravel.com/docs/13.x/migrations

Bab 9

Laravel II: Migrasi, Seeder, Middleware, dan Form Request

Bab 7 membangun meja kasir Profitina Laundry dengan Laravel, dan Bab 8 memperkuatnya dengan harga yang tepat, alur kerja pesanan, dan penomoran halaman. Aplikasinya sudah berjalan, tetapi masih berupa program untuk satu orang di satu laptop. Bab ini menyiapkannya untuk toko sungguhan dengan beberapa gerai, beberapa staf, dan basis data yang sudah berisi pesanan berbulan-bulan. Di akhir bab ini Anda akan mampu mengubah tabel yang sudah berisi data tanpa kehilangan satu baris pun, serta menjelaskan apa yang dicatat oleh tabel `migrations` dan *batch*-nya; mengisi basis data dengan satu bulan pesanan rekaan yang masuk akal memakai *model factory* dan *seeder*, sambil memisahkan daftar harga yang sungguhan dari data rekaan; menjelaskan apa itu *middleware*, menulis *middleware* "sebelum" dan "sesudah", mendaftarkannya, dan memasangnya pada rute; memindahkan validasi dari *controller* ke kelas *form request*, termasuk aturan yang memerlukan basis data; serta membaca seluruh perjalanan satu permintaan melalui *router*, *middleware*, *form request*, dan *controller*.

9.1 Gerai Ketiga Bu Wati

Profitina Laundry dimulai pada 2019 sebagai satu toko di Jl. Kenari Raya, Rungkut, Surabaya. Bu Wati, pemiliknya, membuka gerai kedua di dekat kampus di Keputih pada 2022 dan gerai ketiga di Wiyung pada Januari 2026. Daftar harganya sama di semua gerai, dan pelanggan berpindah-pindah gerai: seorang mahasiswa menitipkan sekantong cucian di Keputih pada hari Senin lalu menanyakan lewat telepon pada hari Rabu apakah cuciannya sudah selesai. Sampai sekarang aplikasi Bab 7 dan Bab 8 tidak tahu di gerai mana sebuah pesanan diterima. Setiap pesanan seolah-olah datang entah dari mana.

Tiga hal lain mulai terasa menyakitkan pada saat yang sama. Yang pertama adalah basis datanya sendiri. Dimas, kasir yang juga mengurus komputer, menyimpan salinan aplikasi di laptopnya untuk mencoba perubahan sebelum dipasang di komputer toko. Ketika ia menambahkan kolom `rack` pada Bab 8, ia menjalankan perubahan itu di laptopnya, lalu di komputer toko, dan seminggu kemudian tidak ada yang ingat komputer mana memiliki kolom apa. Yang kedua adalah data uji. Halaman kasir, daftar pesanan dengan dua puluh pesanan per halaman, dan laporan harian Bab 8 semuanya tampak kosong pada basis data baru, sehingga setiap perubahan harus dicoba dengan mengetik pesanan satu per satu. Yang ketiga adalah kesalahan nyata yang merugikan uang: sebuah pesanan yang sudah selesai dibayar dua kali, sekali tunai di meja kasir dan sekali lagi lewat QRIS di ponsel, karena kasir kedua tidak melihat pembayaran pertama, dan tidak ada satu pun bagian aplikasi yang mencegah pembayaran pada pesanan yang sudah dibawa pulang pelanggan.

Masing-masing masalah ini memiliki jawaban baku di Laravel [1]. Perubahan pada basis data ditulis sebagai *migration* (migrasi), yaitu kelas kecil yang tahu cara membuat perubahan dan cara membatalkannya, dan Laravel mencatat migrasi mana yang sudah dijalankan oleh setiap basis data. Data rekaan berasal dari *factory*, yang menggambarkan cara membuat satu pelanggan atau satu pesanan yang masuk akal, dan dari *seeder*, yang menentukan berapa banyak yang dibuat dan

dengan urutan apa. Pemeriksaan yang harus terjadi sebelum permintaan mencapai *controller*, seperti "pesanan yang sudah meninggalkan toko tidak boleh diubah", ditulis sekali sebagai *middleware*. Daftar aturan validasi yang terus bertambah pindah dari *controller* ke kelas *form request*, tempat aturan yang memerlukan basis data, seperti "tidak boleh membayar lebih dari sisa tagihan", mendapat rumah yang layak.

Tidak satu pun gagasan ini baru bagi Anda. Bab 2 memperkenalkan migrasi dan injeksi dependensi, Bab 6 memakai *middleware pipeline* dan validasi model di ASP.NET Core, dan Bab 8 sudah menulis aturan harga serta alur kerja pesanan sebagai kelas PHP biasa. Bab ini memperlihatkan cara Laravel mengungkapkan setiap gagasan, mengapa ia mengungkapkannya dengan cara itu, dan bagaimana semua bagian itu menyatu dalam aplikasi Profitina Laundry yang sama.

9.2 Gagasannya dengan Kata-Kata Sederhana

9.2.1 Migrasi: buku catatan perubahan basis data

Bayangkan bangunan toko. Ketika Bu Wati menambahkan rak di ruang belakang, ia tidak menggambar ulang denah seluruh bangunan; ia menulis di buku catatan "20 September: rak B dipasang di dinding timur", dan tukang di gerai mana pun dapat mengulangi perubahan yang persis sama. Jika rak itu ternyata keliru, catatan itu juga menjelaskan cara membongkarnya.

Migrasi adalah satu catatan semacam itu untuk basis data. Ia memiliki dua metode: `up()` membuat perubahan dan `down()` membatalkannya. Nama berkasnya diawali tanggal dan jam, sehingga catatan-catatan itu selalu dijalankan dengan urutan yang sama di setiap komputer. Laravel menyimpan tabel bernama `migrations` di setiap basis data, dengan satu baris untuk setiap catatan yang sudah dijalankan di basis data itu beserta nomor *batch* tempat catatan itu dijalankan. Ketika Dimas mengetik `php artisan migrate`, Laravel membandingkan berkas-berkas di `database/migrations` dengan baris-baris di tabel itu dan hanya menjalankan berkas yang belum ada, semuanya dalam satu *batch* baru [2]. Ketika ia mengetik `php artisan migrate:rollback`, Laravel membatalkan *batch* terakhir dengan menjalankan metode `down()` mulai dari yang terbaru.

Akibatnya penting: kumpulan berkas itulah, bukan ingatan siapa pun, yang menentukan bentuk basis data. Laptop Dimas dan komputer toko dapat dibawa ke keadaan yang sama dengan perintah yang sama, dan komputer gerai baru dapat dibangun dari nol dengan menjalankan setiap catatan secara berurutan. Praktik ini memiliki nama, *evolutionary database design* (perancangan basis data evolusioner): skema berubah dalam langkah-langkah kecil yang tercatat dan dapat dibatalkan, bersama kode yang memakainya [3].

9.2.2 Factory dan seeder: satu bulan yang masuk akal, sengaja direka

Untuk mencoba halaman yang menampilkan dua puluh pesanan per halaman, Anda memerlukan lebih dari dua puluh pesanan. Mengetiknya satu per satu lambat, dan pesanan yang Anda ketik semuanya mirip. *Factory* menggambarkan cara membuat satu rekaman yang masuk akal: seorang pelanggan memiliki nama, nomor telepon yang diawali 0812, dan kadang-kadang alamat; sebuah pesanan dimiliki seorang pelanggan, sebuah gerai, dan sebuah layanan yang masih aktif, beratnya antara satu dan sembilan kilogram, dan kadang-kadang dijemput-antar. *Factory* memakai pustaka bernama Faker untuk mengarang nama dan angka [4], dan ia dapat diminta membuat satu rekaman atau empat ratus [5].

Seeder adalah program kecil yang mengisi basis data. Ia menentukan apa yang dibuat dan dengan urutan apa: layanan dan gerai lebih dahulu karena pesanan merujuk kepadanya, lalu pelanggan, dan terakhir pesanan serta pembayaran. Laravel menjalankan *seeder* dengan `php artisan db:seed`, atau bersama pembuatan ulang semua tabel dengan `php artisan migrate:fresh --seed` [6].

Ada dua jenis data isian, dan mencampurkannya adalah kesalahan klasik. Daftar harga dan ketiga gerai adalah data *sungguhan*: setiap basis data memerlukannya, termasuk basis data toko sendiri. Empat puluh pelanggan rekaan beserta pesanannya adalah data *rekaan*: berguna di laptop, berbahaya di toko, tempat seorang kasir bisa saja menelepon pelanggan yang tidak pernah ada atau menghitung uang rekaan dalam laporan harian. Bab ini menyimpannya di *seeder* yang terpisah dan hanya menjalankan *seeder* rekaan di komputer pengembang.

9.2.3 Middleware: pos pemeriksaan yang dilewati setiap permintaan

Sebelum mencapai *controller*, sebuah permintaan melewati sederet pos pemeriksaan, dan dalam perjalanan pulang responsnya melewati pos-pos yang sama dengan urutan terbalik. Setiap pos adalah sebuah *middleware*: kelas dengan satu metode, `handle`, yang menerima permintaan dan sebuah fungsi bernama `$next` yang mewakili "semua yang ada di belakang saya". *Middleware* dapat memeriksa permintaan lalu memanggil `$next` untuk meneruskannya; ia dapat menolak memanggil `$next` dan menjawab permintaan itu sendiri, misalnya dengan pengalihan atau galat; dan ia dapat mengambil respons yang dikembalikan `$next` lalu mengubahnya sebelum menyerahkannya kembali [7]. Anda sudah bertemu rancangan yang sama di ASP.NET Core pada Bab 6, yang juga menyebutnya *middleware pipeline*.

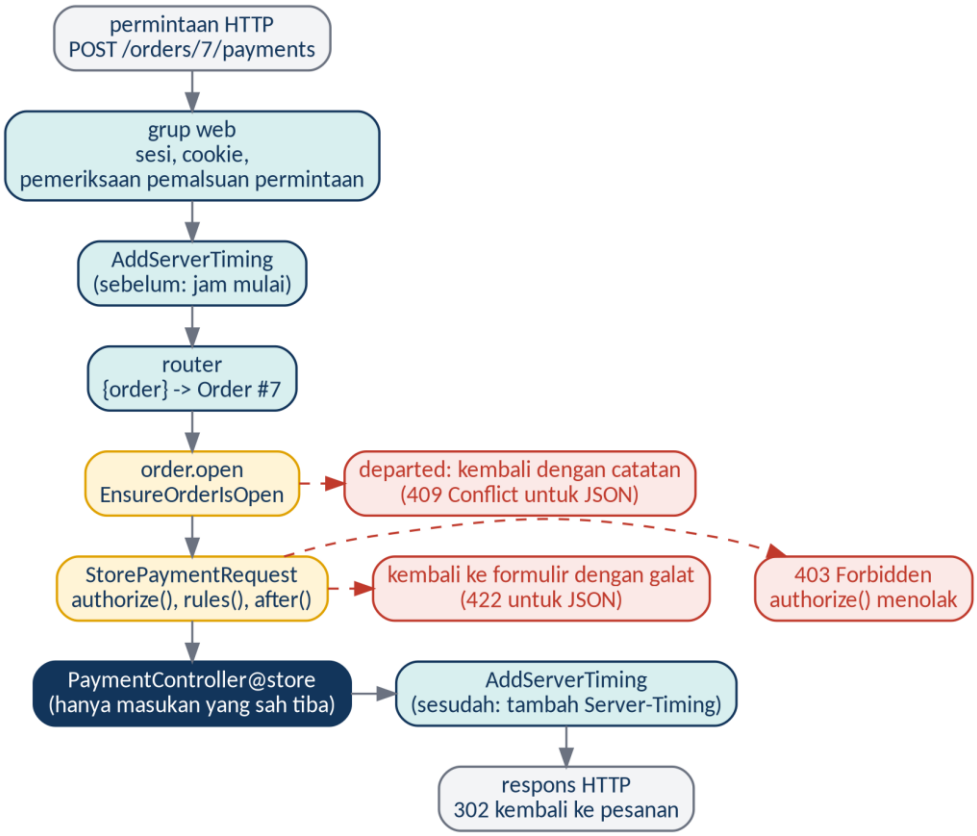
Gambaran yang paling membantu adalah bawang: *controller* adalah intinya, setiap *middleware* adalah satu lapisan yang membungkusnya, dan permintaan berjalan masuk menembus setiap lapisan lalu keluar lagi. Kode yang berjalan sebelum `$next` adalah *middleware* "sebelum"; kode yang berjalan sesudahnya adalah *middleware* "sesudah"; banyak kelas *middleware* melakukan keduanya. Sesi, *cookie*, perlindungan terhadap pengiriman formulir palsu, pemeriksaan masuk, dan pembatasan laju di Laravel sendiri semuanya adalah *middleware*, sehingga aturan baru toko dapat bergabung dengan mereka sebagai anggota yang setara.

9.2.4 Form request: daftar periksa meja kasir, di satu tempat

Ketika pelanggan menyerahkan cucian, kasir menelusuri daftar periksa: siapa pelanggannya, layanan apa, berapa beratnya, diantar atau diambil sendiri, dan jika diantar, ke mana. Pada Bab 7 dan Bab 8 daftar periksa itu ditulis di dalam *controller*, sebagai larik yang diberikan kepada `$request->validate()`. Cara itu cukup untuk tiga isian; ia menjadi sulit dibaca pada delapan isian, dan sulit mengungkapkan pemeriksaan yang memerlukan pesanan itu sendiri, seperti "pembayaran tidak boleh melebihi sisa tagihan".

Form request adalah kelas yang memuat seluruh daftar periksa untuk satu jenis permintaan: apakah pengguna ini boleh membuatnya (`authorize`), cara merapikan masukan mentah lebih dahulu (`prepareForValidation`), aturan-aturannya (`rules`), pesan yang lebih ramah (`messages` dan `attributes`), serta pemeriksaan yang berjalan setelah aturan lolos dan boleh memakai basis data (`after`) [8]. Ketika sebuah metode *controller* meminta *form request* sebagai parameter, Laravel membangun dan memvalidasinya *sebelum* metode itu berjalan. Jika ada yang gagal, *controller* tidak pernah berjalan: manusia dikembalikan ke formulir beserta pesannya, dan program menerima jawaban JSON berstatus 422 dengan daftar galatnya.

Gambar 9.1 menerapkan keempat gagasan subbab ini pada satu permintaan: seorang kasir mencatat pembayaran pada pesanan 7. Permintaan itu melewati grup *middleware* `web`, *middleware* pengukur waktu kita, *router* yang mengubah angka `7` di URL menjadi objek `Order`, *middleware* `order.open` kita, dan *form request*, sebelum *controller* berjalan. Setiap kotak merah adalah tempat permintaan dapat berhenti, dan *controller* hanya pernah melihat permintaan yang sudah lolos dari semuanya.



Gambar 9.1 Perjalanan satu permintaan, yaitu pembayaran pada pesanan 7, melalui *middleware*, *route model binding*, dan *form request*; kotak merah adalah tempat permintaan berhenti sebelum mencapai *controller*.

9.3 Keadaan Proyek Setelah Bab 8

Bab ini mengubah proyek Bab 8, yaitu proyek Bab 7 ditambah penyelesaian Bab 8. Bab ini menambahkan berkas dan mengganti beberapa berkas, dan folder `laravel` bab ini berisi tepat berkas-berkas itu. **Tabel 9.1** mendaftarnya, sehingga Anda selalu tahu mana yang baru dan mana yang berubah.

Tabel 9.1 Berkas yang ditambahkan atau diubah bab ini pada proyek Bab 8.

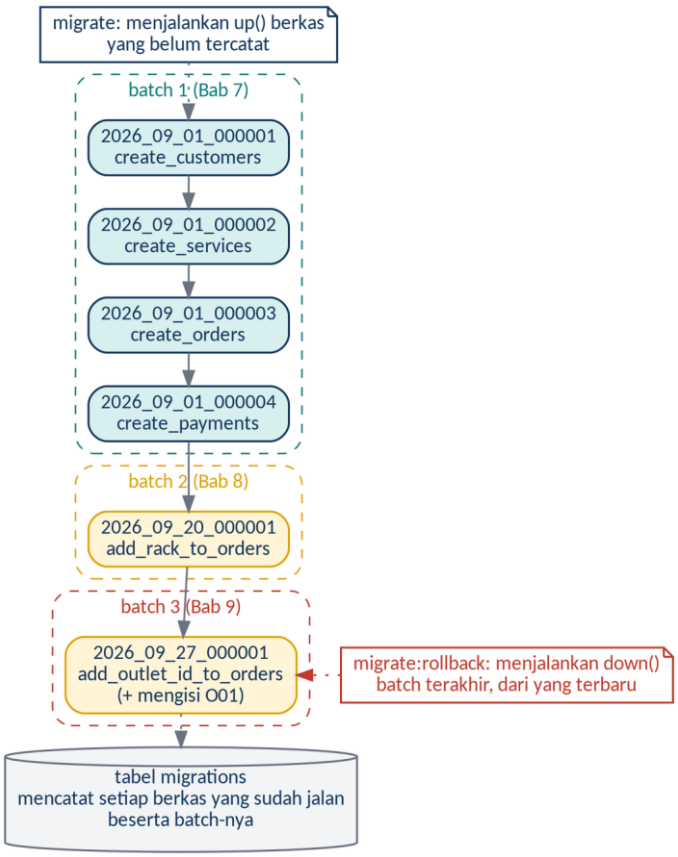
Berkas	Baru atau berubah	Alasan
`database/migrations/2026_09_27_000001_add_outlet_id_to_orders_table.php`	baru	pesanan mengingat gerainya
`app/Models/Customer.php`, `Order.php`, `Payment.php`	berubah	`HasFactory`; `Order` mendapat `outlet()`
`database/factories/*Factory.php` (tiga)	baru	cara mereka satu rekaman
`database/seeders/*.php` (empat)	baru atau berubah	data sungguhan dan rekaan dipisah
`app/Http/Middleware/EnsureOrderIsOpen.php`	baru	pesanan yang sudah pergi ditutup
`app/Http/Middleware/AddServerTiming.php`	baru	*middleware* "sesudah"
`bootstrap/app.php`, `routes/web.php`	berubah	mendaftarkan dan memakai *middleware*
`app/Http/Requests/StoreOrderRequest.php`, `StorePaymentRequest.php`	baru	daftar periksa
`app/Http/Controllers/OrderController.php`, `PaymentController.php`	berubah	memakai *form request*
`resources/views/orders/create.blade.php`	berubah	daftar gerai
`tools/pipeline_demo.php`	baru	bawang dalam PHP biasa

Kelas domain Bab 8 tetap seperti semula: `App\Domain\PriceCalculator` masih menghitung setiap total dengan bilangan bulat, dan `App\Domain\OrderWorkflow` masih menetapkan bahwa pesanan bergerak dari *queued* ke *finished* ke *departed*, dan hanya boleh pergi bila sudah lunas. Bab ini memakai keduanya dan tidak mengubah satu pun. Itulah hasil dari menaruh aturan toko di kelas biasa: fitur kerangka kerja yang baru membungkusnya tanpa menyentuhnya.

9.4 Migrasi Lebih Dalam

9.4.1 Apa yang diingat basis data

Salinan baru proyek yang sudah menjalankan migrasi Bab 7 dan Bab 8 memiliki, selain tabel-tabelnya sendiri, tabel bernama `migrations` dengan dua kolom yang menarik: `migration`, yaitu nama berkas tanpa `.php`, dan `batch`, sebuah angka. Keempat berkas `create_*` dari Bab 7 dijalankan bersama pada saat `php artisan migrate` diketik pertama kali, sehingga keempatnya berada di *batch* 1. Berkas `add_rack_to_orders` dari Bab 8 dijalankan kemudian, sendirian, di *batch* 2. **Gambar 9.2** menunjukkan riwayat itu beserta berkas baru bab ini sebagai *batch* 3.



Gambar 9.2 Riwayat migrasi proyek: setiap kali `migrate` dijalankan terbentuk satu *batch*, tabel `migrations` mencatatnya, dan `migrate:rollback` hanya membatalkan *batch* terakhir.

Dua akibat muncul dari pencatatan ini. Pertama, berkas migrasi yang sudah dijalankan tidak akan pernah dijalankan lagi di basis data itu, apa pun yang Anda ubah di dalamnya. Seandainya Dimas menyunting berkas Bab 8 agar kolom rak lebih panjang, laptopnya, yang sudah menjalankan berkas itu, akan tetap memiliki kolom yang pendek, sedangkan basis data baru akan mendapat kolom yang panjang: dua basis data yang dibangun dari berkas yang "sama", kini berbeda. Maka migrasi yang sudah meninggalkan komputer Anda sendiri adalah sejarah; perubahan pikiran menjadi migrasi *baru*. Kedua, `migrate:rollback` itu tepat sasaran: ia hanya membatalkan *batch* yang tercatat terakhir, sehingga kolom rak di *batch* 2 aman ketika *batch* 3 dibatalkan.

9.4.2 Mengubah tabel yang sudah berisi pesanan

Toko ingin setiap pesanan mengingat gerainya. Di basis data, ini berarti kolom baru `outlet_id` pada `orders` yang berisi salah satu kode `O01`, `O02`, atau `O03` dari tabel `outlets` Bab 7, dengan *foreign key* (kunci tamu) agar tidak ada pesanan yang menunjuk gerai yang tidak ada. Kesulitannya, tabel itu sudah berisi pesanan, mungkin ribuan, dan masing-masing memerlukan nilai yang masuk akal.

Kode 9.1 memperlihatkan migrasinya. Berkas ini dibuat dengan `php artisan make:migration add_outlet_id_to_orders_table --table=orders`, yang menulis kerangka dengan `Schema::table` alih-alih `Schema::create`, karena tabelnya sudah ada [2].

Kode 9.1 `2026_09_27_000001_add_outlet_id_to_orders_table.php`: kolom baru dengan *foreign key*, dan pesanan lama diisi.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Schema;

// Which of the three outlets took the order in.
return new class extends Migration
{
    public function up(): void
    {
        Schema::table('orders', function (Blueprint $table) {
            $table->string('outlet_id', 3)->nullable()->after('customer_id');
            $table->foreign('outlet_id')->references('id')->on('outlets')
                ->restrictOnDelete();
        });

        // Every order so far was taken in at the first shop, Rungkut.
        DB::table('orders')->whereNull('outlet_id')
            ->update(['outlet_id' => 'O01']);
    }

    public function down(): void
    {
        Schema::table('orders', function (Blueprint $table) {
            $table->dropForeign(['outlet_id']);
            $table->dropColumn('outlet_id');
        });
    }
};
```

Bacalah `up()` dalam tiga langkah. Pertama, kolom ditambahkan sebagai `nullable()`. Ini bukan kemalasan: basis data tidak dapat menambahkan kolom `NOT NULL` tanpa nilai bawaan ke tabel yang sudah berisi baris, karena ia tidak tahu apa yang harus diisikan ke baris lama, dan nilai bawaan `'O01'` untuk setiap pesanan *mendatang* juga keliru. Kedua, *foreign key*: `references('id')->on('outlets')` mengikat kolom itu ke kunci primer `outlets`, dan `restrictOnDelete()` melarang penghapusan gerai yang masih memiliki pesanan, pilihan yang sama dengan yang dibuat Bab 7 untuk pelanggan dan layanan. Ketiga, datanya: setiap pesanan yang sudah ada sebelum migrasi ini diterima di toko pertama, sehingga migrasi mengisi `O01` ke semua baris yang `outlet_id`-nya masih kosong. Ini adalah *migrasi data* di dalam migrasi skema, dan sengaja ditulis dengan *query builder* (`DB::table`), bukan dengan model `Order`: migrasi harus tetap berjalan setahun lagi, ketika kelas `Order` mungkin sudah berubah bentuk, sedangkan *query builder* hanya memerlukan tabelnya.

`after('customer_id')` hanya menentukan letak kolom ketika tabel dilihat dengan perangkat basis data; ia tidak berpengaruh pada aplikasi dan diabaikan oleh basis data yang tidak dapat mengatur letak kolom, tetapi membuat tabel tetap mudah dibaca. `down()` membatalkan perubahan dengan urutan terbalik: *foreign key* lebih dahulu, lalu kolomnya. Pada SQLite, yang dipakai proyek Laravel baru secara bawaan, Laravel membangun ulang tabel di balik layar untuk membuang kolom yang memiliki *foreign key*; pada MySQL dan PostgreSQL ia mengirim perintah `ALTER TABLE` [2]. Migrasinya sama untuk semuanya, dan justru itulah gunanya menulisnya dalam PHP, bukan dalam SQL milik satu basis data.

9.4.3 Perintah-perintah yang Anda perlukan

Tabel 9.2 mendaftar perintah yang bekerja dengan migrasi. Setiap perintah dijalankan di folder proyek, di terminal atau di panel terminal VS Code, sama saja di Windows, macOS, dan Linux.

Tabel 9.2 Perintah migrasi Artisan dan kapan masing-masing dipakai.

Perintah	Yang dilakukannya	Gunakan ketika
`php artisan make:migration add_x_to_orders_table --table=orders`	menulis berkas migrasi baru dengan `Schema::table`	Anda perlu perubahan
`php artisan migrate:status`	mendaftar setiap berkas dengan "Ran" atau "Pending" dan *batch*-nya	sebelum dan sesudah setiap perubahan
`php artisan migrate`	menjalankan berkas yang tertunda sebagai satu *batch* baru	untuk menerapkan perubahan
`php artisan migrate --pretend`	mencetak SQL-nya tanpa menjalankannya	untuk memeriksa perubahan pada basis data sungguhan
`php artisan migrate:rollback`	menjalankan `down()` *batch* terakhir, mulai dari yang terbaru	untuk membatalkan yang baru saja dilakukan
`php artisan migrate:rollback --step=1`	hanya membatalkan berkas terbaru	untuk membatalkan satu berkas dari *batch* yang lebih besar
`php artisan migrate:fresh --seed`	membuang semua tabel, menjalankan semua migrasi, lalu *seeder*	hanya di komputer Anda sendiri

Baris terakhir pantas mendapat peringatannya sendiri. `migrate:fresh` membuang semua tabel beserta semua barisnya sebelum membangunnya kembali. Di laptop yang penuh data rekaan, itulah yang Anda inginkan; di komputer toko, perintah itu akan menghapus usaha Bu Wati. Laravel meminta konfirmasi bila lingkungan aplikasi adalah `production`, tetapi kebiasaan yang lebih aman adalah tidak pernah mengetiknya di mana pun selain di komputer Anda sendiri.

9.4.4 Empat aturan untuk basis data yang dipakai bersama

Praktik migrasi dapat diringkas menjadi empat aturan, dan aturan itu berlaku pula untuk migrasi Entity Framework Core di Bab 5 dan untuk kerangka kerja lain mana pun.

- **Jangan pernah menyunting migrasi yang sudah dijalankan di tempat lain selain komputer Anda sendiri.** Tulislah migrasi baru.
- **Jadikan setiap `down()` pembatalan yang sesungguhnya.** Jika `up()` menambahkan dua kolom, `down()` membuang keduanya; ujilah sekali dengan `migrate:rollback` lalu `migrate`.
- **Tambahkan kolom sebagai *nullable* atau dengan nilai bawaan, lalu isi.** Kolom yang kelak harus wajib dapat diperketat oleh migrasi berikutnya, setelah datanya lengkap.
- **Buat perubahan data di dalam migrasi tetap kecil dan ditulis dengan *query builder*.** Perubahan data yang besar lebih baik ditempatkan di perintah tersendiri yang dapat dijalankan, diawasi, dan diulang secara terpisah.

9.5 Factory dan Seeder

9.5.1 Model yang dapat direka

Sebuah model hanya dapat dibuat oleh *factory* jika ia menyatakannya, dengan *trait* `HasFactory`. **Kode 9.2** memperlihatkan model `Order` bab ini: dibandingkan Bab 8, ia memiliki *trait* itu, `outlet_id` di daftar isian yang boleh diisi, dan relasi baru `outlet()`. `Customer` dan `Payment` mendapat *trait* yang sama dengan cara serupa. Atribut `#[Fillable]` di atas kelas adalah cara Laravel 13 mendaftar isian yang boleh diisi dari larik, yang sudah diperkenalkan Bab 7 [1].

Kode 9.2 `app/Models/Order.php`: model Bab 8 dengan `HasFactory` dan relasi baru ke gerai.

```php
<?php

namespace App\Models;

use App\Enums\OrderStatus;
use Illuminate\Database\Eloquent\Attributes\Fillable;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;

#[Fillable(['customer_id', 'outlet_id', 'service_id', 'quantity', 'pickup',
    'delivery_address', 'total', 'status', 'note', 'rack', 'received_at',
    'finished_at', 'departed_at'])]
class Order extends Model
{
    use HasFactory;

    protected function casts(): array
    {
        return [
            'quantity' => 'decimal:2',
            'pickup' => 'boolean',
            'total' => 'integer',
            'status' => OrderStatus::class,
            'received_at' => 'datetime',
            'finished_at' => 'datetime',
            'departed_at' => 'datetime',
        ];
    }

    public function customer(): BelongsTo
    {
        return $this->belongsTo(Customer::class);
    }

    public function outlet(): BelongsTo
    {
        return $this->belongsTo(Outlet::class);
    }

    public function service(): BelongsTo
    {
        return $this->belongsTo(Service::class);
    }
```

Kode 9.2 (Lanjutan) *app/Models/Order.php: model Bab 8 dengan HasFactory dan relasi baru ke gerai.*

```php
    public function payments(): HasMany
    {
        return $this->hasMany(Payment::class);
    }

    // Sums the loaded payments; load them with with('payments') first.
    public function paid(): int
    {
        return (int) $this->payments->sum('amount');
    }

    public function balance(): int
    {
        return $this->total - $this->paid();
    }
}
```

Menurut konvensi, `Order::factory()` mencari kelas `Database\Factories\OrderFactory`; berkasnya berada di `database/factories` dan dibuat dengan `php artisan make:factory OrderFactory --model=Order` [5].

9.5.2 Menulis factory

Factory pelanggan di **Kode 9.3** adalah yang paling sederhana. Satu-satunya metodenya, `definition()`, mengembalikan isian satu pelanggan. `fake()` mengembalikan pembangkit Faker: `name()` mengarang nama lengkap, `numerify('0812########')` mengganti setiap `#` dengan angka acak, dan `optional(0.6)` mengembalikan alamat jalan enam dari sepuluh kali dan `null` selebihnya, karena pelanggan sungguhan tidak selalu meninggalkan alamat [4]. `unique()` membuat pembangkit mengingat nomor telepon yang sudah dihasilkannya dalam satu kali jalan, karena kolom `phone` di `customers` bersifat unik dan nomor ganda akan menghentikan *seeder* dengan galat.

Kode 9.3 `database/factories/CustomerFactory.php`: satu pelanggan rekaan.

```php
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

/**
 * @extends Factory<\App\Models\Customer>
 */
class CustomerFactory extends Factory
{
    public function definition(): array
    {
        return [
            'name' => fake()->name(),
            'phone' => fake()->unique()->numerify('0812########'),
            'address' => fake()->optional(0.6)->streetAddress(),
        ];
    }
}
```

Faker menguasai banyak bahasa. Dengan `APP_FAKER_LOCALE=id_ID` di berkas `.env` proyek, `fake()->name()` menghasilkan nama-nama Indonesia seperti pelanggan yang benar-benar dilayani toko, dan halaman demo tampak jauh lebih mirip meja kasir yang sesungguhnya.

Factory pesanan di **Kode 9.4** lebih banyak mengajarkan. Tiga isiannya bukan nilai, melainkan *petunjuk*. `'customer_id' => Customer::factory()` berarti "buatkan pelanggan baru untuk pesanan ini, kecuali saya diberi tahu lain". `'outlet_id'` dan `'service_id'` adalah *closure* yang baru dijalankan ketika rekaman benar-benar dibuat, dan masing-masing memilih gerai acak yang ada dan layanan aktif yang acak. Perbedaannya penting: *closure* dievaluasi sekali untuk setiap pesanan, sehingga pesanan tersebar ke semua gerai dan layanan, sedangkan nilai biasa akan sama untuk semua pesanan.

Kode 9.4 `database/factories/OrderFactory.php`: satu pesanan rekaan, diberi harga dengan aturan yang sungguhan, dengan *state* untuk pesanan yang sudah selesai dan sudah pergi.

```php
<?php

namespace Database\Factories;

use App\Domain\PriceCalculator;
use App\Enums\OrderStatus;
use App\Models\Customer;
use App\Models\Order;
use App\Models\Outlet;
use App\Models\Service;
use Illuminate\Database\Eloquent\Factories\Factory;

/**
 * @extends Factory<Order>
 */
class OrderFactory extends Factory
{
    public function definition(): array
    {
        $pickup = fake()->boolean(25);

        return [
            'customer_id' => Customer::factory(),
            'outlet_id' => fn () => Outlet::inRandomOrder()->value('id'),
            'service_id' => fn () => Service::where('is_active', true)
                ->inRandomOrder()->value('id'),
            'quantity' => fake()->randomFloat(1, 1, 9),
            'pickup' => $pickup,
            'delivery_address' => $pickup ? fake()->streetAddress() : null,
            'total' => 0,                       // worked out in configure()
            'status' => OrderStatus::Queued,
            'received_at' => fake()->dateTimeBetween('-30 days', 'now'),
        ];
    }

    // The price always follows the shop's rule; it is never made up.
    public function configure(): static
    {
        return $this->afterMaking(function (Order $order) {
            $service = Service::findOrFail($order->service_id);
            $order->total = PriceCalculator::total($service->per_kg,
                $service->price, (string) $order->quantity, $order->pickup);
        });
    }
```

Kode 9.4 (Lanjutan) *database/factories/OrderFactory.php: satu pesanan rekaan, diberi harga dengan aturan yang sungguhan, dengan state untuk pesanan yang sudah selesai dan sudah pergi.*

```php
    public function finished(): static
    {
        return $this->state(fn (array $attributes) => [
            'status' => OrderStatus::Finished,
            'finished_at' => now(),
        ]);
    }

    public function departed(): static
    {
        return $this->state(fn (array $attributes) => [
            'status' => OrderStatus::Departed,
            'finished_at' => now(),
            'departed_at' => now(),
        ]);
    }
}
```

Baris terpenting justru baris yang *tidak* acak: totalnya. Pesanan rekaan dengan total rekaan akan membuat laporan Bab 8 menampilkan angka yang tidak mungkin dihasilkan pesanan sungguhan, dan kekeliruan dalam aturan harga akan tersembunyi di baliknya. Karena itu definisinya mengisi `total` dengan 0, dan `configure()` mendaftarkan *callback* `afterMaking` yang menghitung total sebenarnya dengan `PriceCalculator::total`, metode yang sama dengan yang dipakai meja kasir. `afterMaking` berjalan untuk setiap rekaman yang dibuat *factory*, baik hanya dibuat di memori maupun juga disimpan, sehingga totalnya selalu benar.

Dua metode di akhir adalah *state*: variasi definisi yang diberi nama. `Order::factory()->finished()` membuat pesanan yang sudah dicuci, dan `Order::factory()->departed()` membuat pesanan yang sudah meninggalkan toko. *State* membuat uji dan *seeder* mudah dibaca; pembaca `Order::factory()->departed()->create()` langsung tahu jenis pesanan yang diperlukan tanpa membaca daftar isiannya.

9.5.3 Menulis seeder

`DatabaseSeeder` Bab 8 mengerjakan semuanya dalam satu metode: membaca daftar harga dan gerai dari berkas CSV, lalu membuat tiga pelanggan beserta pesanannya. Bab ini memecahnya menjadi tiga *seeder* yang masing-masing memiliki satu tugas, ditambah `DatabaseSeeder` yang memutuskan mana yang dijalankan.

Kode 9.5 adalah *seeder* untuk data sungguhan. Ia membaca berkas `Service.csv` dan `Outlet.csv` yang sama dengan Bab 3, 5, dan 7, sehingga versi ASP.NET Core dan versi Laravel toko ini tetap mengenakan harga yang persis sama. Satu-satunya perubahan dari Bab 8 adalah `updateOrCreate` sebagai pengganti `create`: argumen pertama menyatakan cara menemukan baris yang sudah ada (dengan kodenya, seperti `S0001`) dan argumen kedua menyatakan isinya. Menjalankan *seeder* ini dua kali tidak mengubah apa pun pada kali kedua, dan menjalankannya setelah pemilik membetulkan harga di berkas CSV akan memperbarui harga itu, bukan gagal karena kunci ganda. *Seeder* yang aman dijalankan ulang disebut *idempotent*, dan *seeder* data sungguhan seharusnya selalu demikian.

Kode 9.5 `database/seeders/PriceListSeeder.php`: daftar harga dan gerai yang sungguhan, aman dijalankan ulang.

```php
<?php

namespace Database\Seeders;

use App\Models\Outlet;
use App\Models\Service;
use Illuminate\Database\Seeder;

// The real price list and the three outlets: needed in every database.
class PriceListSeeder extends Seeder
{
    public function run(): void
    {
        foreach ($this->csv('Service.csv') as [$id, $description, $price]) {
            Service::updateOrCreate(['id' => $id], [
                'description' => $description, 'price' => (int) $price]);
        }
        foreach ($this->csv('Outlet.csv') as [$id, $name, $address, $phone,
            $opened]) {
            Outlet::updateOrCreate(['id' => $id],
                compact('name', 'address', 'phone', 'opened'));
        }
    }

    // The rows of database/data/<file>, without the header line.
    private function csv(string $file): array
    {
        $lines = file(database_path("data/{$file}"), FILE_IGNORE_NEW_LINES);
        $rows = array_map(fn ($line) => str_getcsv($line, ',', '"', ''), $lines);

        return array_slice($rows, 1);
    }
}
```

Tiga pelanggan tulisan tangan dari Bab 7, yaitu Sari, Budi, dan Dewi, pindah tanpa perubahan ke `DemoSeeder`, dengan gerai ditambahkan pada setiap pesanannya. Mereka berguna karena buku ini menyebut mereka dengan nama: pesanan Sari seberat 4,5 kg selalu ada dan selalu berharga Rp 36.000. **Kode 9.6** menambahkan keramaian di sekitar mereka.

Kode 9.6 `database/seeders/BusyMonthSeeder.php`: empat puluh pelanggan rekaan dan satu bulan pesanan yang bergerak melalui alur kerja seperti pesanan sungguhan.

```php
<?php

namespace Database\Seeders;

use App\Enums\OrderStatus;
use App\Models\Customer;
use App\Models\Order;
use App\Models\Payment;
use Illuminate\Database\Seeder;

// A made-up month of work, so that lists, paging and reports have data.
class BusyMonthSeeder extends Seeder
{
    public function run(): void
    {
        Customer::factory()->count(40)->create()->each(function ($customer) {
            Order::factory()->count(fake()->numberBetween(0, 5))
                ->for($customer)->create()
                ->each(fn (Order $order) => $this->liveThrough($order));
        });
    }

    // Older orders are further along, as they would be in a real shop.
    private function liveThrough(Order $order): void
    {
        if (fake()->boolean(70)) {                  // most pay when they drop
            Payment::factory()->for($order)->create([
                'amount' => $order->total, 'paid_at' => $order->received_at]);
        }
        $age = $order->received_at->diffInDays(now());
        if ($age >= 2) {
            $order->status = OrderStatus::Finished;
            $order->finished_at = $order->received_at->copy()->addDays(2);
        }
        if ($age >= 5 && $order->payments()->sum('amount') >= $order->total) {
            $order->status = OrderStatus::Departed;
            $order->departed_at = $order->received_at->copy()->addDays(5);
        }
        $order->save();
    }
}
```

`Customer::factory()->count(40)->create()` membuat dan menyimpan empat puluh pelanggan lalu mengembalikannya sebagai koleksi. Untuk setiap pelanggan, `Order::factory()->count(...)->for($customer)->create()` membuat nol sampai lima pesanan milik pelanggan itu; `for()` menggantikan petunjuk "buatkan pelanggan baru" pada *factory*. Metode privat `liveThrough` kemudian memberi setiap pesanan riwayat yang masuk akal. Tujuh dari sepuluh pesanan dibayar lunas saat dititipkan. Pesanan yang berumur sedikitnya dua hari sudah selesai, dan pesanan yang berumur sedikitnya lima hari serta sudah lunas sudah pergi. Ini mengikuti aturan `OrderWorkflow` Bab 8 bahwa cucian hanya boleh pergi bila sudah dibayar, sehingga data rekaan tidak pernah memuat keadaan yang akan ditolak oleh aplikasi yang sesungguhnya.

Kode 9.7 menyatukan ketiganya.

Kode 9.7 `database/seeders/DatabaseSeeder.php`: data sungguhan selalu, data rekaan hanya di komputer pengembang atau komputer uji.

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    public function run(): void
    {
        $this->call(PriceListSeeder::class);      // real data: always

        if (app()->environment('local', 'testing')) {
            $this->call([DemoSeeder::class,       // made-up data: never in
                BusyMonthSeeder::class]);         // the shop's real database
        }
    }
}
```

`app()->environment('local', 'testing')` bernilai benar bila pengaturan `APP_ENV` di `.env` adalah `local`, seperti pada setiap proyek baru, atau `testing`, seperti ketika uji Bab 10 dijalankan. Di komputer toko `APP_ENV` adalah `production`, sehingga `php artisan db:seed` di sana memasang atau memperbarui daftar harga dan gerai, tidak lebih. Urutan pemanggilan di `$this->call([...])` adalah urutan jalannya *seeder*, dan urutan itu harus mengikuti *foreign key*: layanan dan gerai sebelum pesanan.

9.5.4 Menjalankannya

Setelah berkas-berkasnya terpasang, satu perintah membangun ulang basis data laptop dari nol:

```
php artisan migrate:fresh --seed
```

Perintah ini membuang semua tabel, menjalankan keenam berkas migrasi, dari empat berkas Bab 7, kolom rak Bab 8, hingga kolom gerai bab ini, lalu menjalankan `DatabaseSeeder`. Bukalah daftar pesanan sesudahnya: ada beberapa halaman, dengan pesanan di setiap status dan di setiap gerai; laporan harian Bab 8 memiliki satu kolom angka untuk setiap hari dalam tiga puluh hari terakhir. Karena datanya acak, daftar Anda tidak akan sama dengan milik orang lain, tetapi pesanan Sari, Budi, dan Dewi selalu ada. Untuk melihat data dari terminal, `php artisan tinker` membuka *prompt* PHP di dalam aplikasi, tempat `Order::where('status', 'departed')->count()` atau `Order::first()->outlet->name` dapat diketik dan langsung dijawab.

9.6 Middleware

9.6.1 Bawang, tanpa Laravel

Middleware menjadi jelas begitu Anda melihat betapa sedikit kode yang diperlukannya. **Kode 9.8** membangun *pipeline* dua lapis di sekitar *controller* pura-pura dalam PHP biasa, hanya dengan *closure* dan `array_reduce`. Program ini berjalan di mana pun PHP berjalan, tanpa kerangka kerja apa pun.

Kode 9.8 `tools/pipeline_demo.php`: *middleware pipeline* dalam PHP biasa, gagasan yang sama dengan dasar *pipeline* Laravel.

```php
<?php

// How a middleware pipeline works, in plain PHP (no Laravel needed):
//   php tools/pipeline_demo.php
// Each layer gets the request and $next, the rest of the pipeline.

$log = fn (string $line) => print("  $line\n");

$timing = function (array $request, Closure $next) use ($log) {
    $log('timing: start the clock');
    $response = $next($request);
    $log('timing: add the Server-Timing header');
    return $response + ['Server-Timing' => 'app;dur=1.2'];
};

$orderOpen = function (array $request, Closure $next) use ($log) {
    if ($request['status'] === 'departed') {
        $log('order.open: departed, stop here');
        return ['status' => 409];               // $next is never called
    }
    $log('order.open: still open, go on');
    return $next($request);
};

$controller = function (array $request) use ($log) {
    $log("controller: record the payment on order #{$request['order']}");
    return ['status' => 302];
};

// Wrap the controller in the layers, the last layer innermost.
$pipeline = array_reduce(array_reverse([$timing, $orderOpen]),
    fn (Closure $next, Closure $layer) =>
        fn (array $request) => $layer($request, $next),
    $controller);

foreach (['finished', 'departed'] as $status) {
    echo "Payment on an order that is $status:\n";
    $response = $pipeline(['order' => 7, 'status' => $status]);
    echo '  -> ', json_encode($response), "\n";
}
```

Setiap lapisan adalah fungsi dengan dua argumen, yaitu permintaan dan `$next`. Baris yang menarik adalah `array_reduce`: ia mulai dari *controller* lalu membungkusnya, lapis demi lapis, dengan fungsi baru yang memanggil lapisan itu dengan permintaan dan dengan semua yang ada di dalamnya sebagai `$next`. Lapisan-lapisannya dibalik lebih dahulu, agar lapisan pertama dalam daftar menjadi yang paling luar, sama seperti urutan Laravel. Nilai “Server-Timing” pada lapisan pengukur waktu di sini adalah angka tetap, karena demo ini membahas urutan kejadian, bukan pengukuran. Menjalankannya mencetak:

```
php tools/pipeline_demo.php
```

```
Payment on an order that is finished:
  timing: start the clock
  order.open: still open, go on
  controller: record the payment on order #7
  timing: add the Server-Timing header
  -> {"status":302,"Server-Timing":"app;dur=1.2"}
Payment on an order that is departed:
  timing: start the clock
  order.open: departed, stop here
  timing: add the Server-Timing header
  -> {"status":409,"Server-Timing":"app;dur=1.2"}
```

Bacalah jalannya yang kedua dengan cermat. Lapisan `order.open` tidak memanggil `$next`, sehingga *controller* tidak pernah berjalan dan tidak ada yang dicatat. Namun lapisan pengukur waktu, yang berada *di luarnya*, tetap menjalankan bagian "sesudah"-nya dan menambahkan *header*-nya pada jawaban 409. *Middleware* yang menghentikan permintaan menghentikan semua yang ada *di dalamnya*, tidak pernah yang ada di luarnya. *Pipeline* Laravel menambahkan layanan dan kemudahan di sekitar gagasan ini, tetapi gagasannya persis dua puluh baris itu.

9.6.2 Middleware "sebelum" untuk aturan toko

Kode 9.9 adalah aturan yang seharusnya menyelamatkan Bu Wati dari pembayaran ganda: pesanan yang sudah meninggalkan toko ditutup. Kelas ini dibuat dengan `php artisan make:middleware EnsureOrderIsOpen` dan berada di `app/Http/Middleware`.

Kode 9.9 `app/Http/Middleware/EnsureOrderIsOpen.php`: pesanan yang sudah meninggalkan toko tidak boleh diubah.

```php
<?php

namespace App\Http\Middleware;

use App\Enums\OrderStatus;
use App\Models\Order;
use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

// An order that has left the shop is history: nothing may change it.
class EnsureOrderIsOpen
{
    public function handle(Request $request, Closure $next): Response
    {
        $order = $request->route('order');      // already bound by the router

        if ($order instanceof Order
            && $order->status === OrderStatus::Departed) {
            $message = "Order #{$order->id} has left the shop and is closed.";
            if ($request->expectsJson()) {
                abort(409, $message);           // for programs: 409 Conflict
            }
            return back()->with('error', $message);   // for people: a note
        }

        return $next($request);
    }
}
```

`$request->route('order')` meminta *router* memberikan parameter rute bernama `order`. Karena *controller* Bab 7 menyatakan `Order $order` sebagai parameter, *route model binding* Laravel sudah mengubah angka `7` di URL menjadi `Order` dengan id 7, atau menjawab 404 bila tidak ada, sebelum *middleware* kita berjalan; kita menerima modelnya, bukan teksnya [9]. Uji `instanceof` adalah penjaga: jika *middleware* ini suatu saat dipasang pada rute tanpa parameter `{order}`, ia cukup meneruskan permintaannya.

Jawaban untuk pesanan yang sudah ditutup bergantung pada siapa yang bertanya. Manusia di meja kasir menekan tombol di halaman pesanan, sehingga jawaban yang ramah adalah kembali ke halaman itu dengan catatan, memakai pesan kilat `error` yang sudah ditampilkan merah oleh tata letak Bab 7. Program, seperti aplikasi ponsel di Bab 10, tidak dapat membaca catatan merah; ia mengirim `Accept: application/json` dan memerlukan kode status. `expectsJson()` membedakan keduanya, dan `abort(409, ...)` menjawab program dengan *409 Conflict*, status

HTTP untuk permintaan yang bertentangan dengan keadaan sumber daya saat ini [10]. Laravel mengubah `abort` itu menjadi badan JSON `{"message": "..."}` bagi klien semacam itu.

9.6.3 Middleware "sesudah"

Kode 9.10 memperlihatkan belahan bawang yang lain. `AddServerTiming` menyalakan jam, membiarkan sisa *pipeline* berjalan dengan memanggil `$next`, lalu menambahkan *header* pada respons yang diterimanya kembali.

Kode 9.10 `app/Http/Middleware/AddServerTiming.php`: *middleware* "sesudah" yang mengukur sisa *pipeline*.

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

// Measures how long the rest of the pipeline took, for the browser's tools.
class AddServerTiming
{
    public function handle(Request $request, Closure $next): Response
    {
        $start = hrtime(true);                  // before: start the clock

        $response = $next($request);

        $ms = (hrtime(true) - $start) / 1e6;    // after: stop it
        $response->headers->set('Server-Timing',
            sprintf('app;dur=%.1f', $ms));

        return $response;
    }
}
```

`hrtime(true)` mengembalikan pembacaan jam dalam nanodetik; selisihnya dibagi 10^6 adalah milidetik. *Header* `Server-Timing` adalah standar yang dipahami peramban: bukalah *developer tools* di Chrome, Edge, atau Firefox, pilih tab Network, klik sebuah permintaan dan lihat bagian "Timing", maka nilainya muncul sebagai batang bernama `app` [11]. Ketika daftar pesanan terasa lambat di gerai Keputih, *header* ini langsung memberi tahu apakah waktunya habis di aplikasi atau di jaringan.

9.6.4 Mendaftarkan middleware

Kelas *middleware* tidak melakukan apa pun sampai didaftarkan. Sejak Laravel 11 pendaftaran ini dilakukan di `bootstrap/app.php`, yang menggantikan `app/Http/Kernel.php` versi lama [7]. **Kode 9.11** memperlihatkan berkas bab ini.

Kode 9.11 `bootstrap/app.php`: `AddServerTiming` bergabung dengan grup `web`, dan `EnsureOrderIsOpen` mendapat nama pendek `order.open`.

```php
<?php

use App\Http\Middleware\AddServerTiming;
use App\Http\Middleware\EnsureOrderIsOpen;
use Illuminate\Foundation\Application;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;

return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__.'/../routes/web.php',
        commands: __DIR__.'/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware): void {
        $middleware->web(append: [AddServerTiming::class]);
        $middleware->alias([
            'order.open' => EnsureOrderIsOpen::class,
        ]);
    })
    ->withExceptions(function (Exceptions $exceptions): void {
        //
    })->create();
```

Ada tiga cara mendaftarkan *middleware*, dan berkas ini memakai dua di antaranya. `$middleware->web(append: [...])` menambahkan kelas ke ujung *grup* `web`, yaitu daftar yang berjalan untuk setiap rute di `routes/web.php`; itu tepat untuk pengukur waktu, yang seharusnya mengukur setiap halaman. `$middleware->alias([...])` memberi kelas nama pendek yang dapat diminta oleh rute; itu tepat untuk `order.open`, yang hanya diperlukan beberapa rute. Cara ketiga, `$middleware->append(...)`, akan menambahkan kelas ke *middleware global* yang berjalan untuk setiap permintaan dalam bentuk apa pun, termasuk yang tidak cocok dengan rute mana pun; kedua kelas kita tidak memerlukannya. **Tabel 9.3** mendaftar *middleware* yang sudah disediakan Laravel dengan nama pendek dan dipakai di bab-bab berikutnya.

Tabel 9.3 *Middleware* rute bawaan Laravel yang dipakai buku ini, beserta nama pendeknya.

Nama	Yang diperiksa atau dilakukannya	Pemakaian umum
`auth`	pengguna sudah masuk, jika tidak dialihkan ke halaman masuk (atau 401 untuk JSON)	halaman staf (Bab 10)
`guest`	pengguna *belum* masuk	halaman masuk (Bab 10)
`throttle:5,1`	paling banyak 5 permintaan per menit per pengguna atau alamat, jika tidak 429	formulir masuk
`can:manage-prices`	pengguna lolos pemeriksaan otorisasi, jika tidak 403	daftar harga (Bab 10)
`verified`	pengguna sudah mengonfirmasi alamat surel	alur pendaftaran opsional
`signed`	URL membawa tanda tangan yang sah	tautan yang dikirim lewat surel atau WhatsApp

9.6.5 Memasang middleware pada rute

Kode 9.12 adalah berkas rute Bab 8 dengan satu perubahan di bagian bawah: dua rute yang mengubah pesanan, "maju" dan "catat pembayaran", kini berada di dalam grup yang memakai `order.open`.

Kode 9.12 `routes/web.php`: rute-rute Bab 8, dengan rute yang mengubah pesanan dikelompokkan di belakang `order.open`.

```php
<?php

use App\Http\Controllers\CounterController;
use App\Http\Controllers\CustomerController;
use App\Http\Controllers\HomeController;
use App\Http\Controllers\OrderController;
use App\Http\Controllers\OutletController;
use App\Http\Controllers\PaymentController;
use App\Http\Controllers\PriceController;
use App\Http\Controllers\ReportController;
use App\Http\Controllers\ServiceController;
use Illuminate\Support\Facades\Route;

// The public pages (Chapter 3 built them in ASP.NET Core).
Route::get('/', HomeController::class)->name('home');
Route::get('services', [ServiceController::class, 'index'])
    ->name('services.index');
Route::get('services/{service}', [ServiceController::class, 'show'])
    ->name('services.show');
Route::get('outlets', [OutletController::class, 'index'])
    ->name('outlets.index');

// The staff pages (Chapter 5 built them in ASP.NET Core).
Route::get('counter', CounterController::class)->name('counter');
Route::resource('customers', CustomerController::class);
Route::resource('prices', PriceController::class)
    ->parameters(['prices' => 'service'])
    ->except(['show', 'destroy']);
Route::resource('orders', OrderController::class)
    ->only(['index', 'create', 'store', 'show']);
Route::get('reports/daily', [ReportController::class, 'daily'])
    ->name('reports.daily');

// Chapter 9: nothing may change an order that has left the shop.
Route::middleware('order.open')->group(function () {
    Route::post('orders/{order}/advance', [OrderController::class, 'advance'])
        ->name('orders.advance');
    Route::post('orders/{order}/payments', [PaymentController::class, 'store'])
        ->name('orders.payments.store');
});
```

`Route::middleware('order.open')->group(function () { ... })` menerapkan *middleware* itu pada setiap rute yang didefinisikan di dalam *closure*. Efek yang sama untuk satu rute adalah `->middleware('order.open')` di akhir definisinya. Pengelompokan memiliki keunggulan yang mudah terlewat: aturannya kini terlihat *di berkas rute*. Siapa pun yang membuka `routes/web.php` melihat halaman mana dijaga aturan mana, tanpa membuka satu *controller* pun. Menampilkan pesanan dan mendaftar pesanan tentu tetap di luar grup; pesanan yang sudah pergi tetap harus dapat dilihat.

`OrderController::advance` Bab 8 juga menolak memajukan pesanan yang sudah pergi, melalui `OrderWorkflow`, dan hal itu tetap berlaku. Kedua pemeriksaan itu bukan duplikasi, melainkan dua lapis pertahanan: alur kerja melindungi aturan dari mana pun ia dipanggil, bahkan dari perintah Artisan atau uji, sedangkan *middleware* menghentikan permintaan lebih awal untuk setiap rute di dalam grup, termasuk rute yang sama sekali tidak memakai alur kerja, seperti pembayaran.

9.7 Form Request

9.7.1 Daftar periksa pesanan

Kode 9.13 memindahkan validasi pesanan baru Bab 8 keluar dari *controller*, menambahkan gerai, dan menambahkan perapian yang diperlukan meja kasir. Kelas ini dibuat dengan `php artisan make:request StoreOrderRequest` dan berada di `app/Http/Requests`.

Kode 9.13 `app/Http/Requests/StoreOrderRequest.php`: daftar periksa untuk pesanan baru, dengan perapian, aturan, pesan, dan nama isian.

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;
use Illuminate\Validation\Rule;

class StoreOrderRequest extends FormRequest
{
    public function authorize(): bool
    {
        return true;               // Chapter 10 decides who may do this
    }

    // Tidy the raw input before any rule looks at it.
    protected function prepareForValidation(): void
    {
        $this->merge([
            'quantity' => str_replace(',', '.', (string) $this->quantity),
            'pickup' => $this->boolean('pickup'),
            'rack' => strtoupper(trim((string) $this->rack)) ?: null,
        ]);
    }

    public function rules(): array
    {
        return [
            'customer_id' => ['required', 'integer', 'exists:customers,id'],
            'outlet_id' => ['required', 'string', 'exists:outlets,id'],
            'service_id' => ['required', 'string',
                Rule::exists('services', 'id')->where('is_active', true)],
            'quantity' => ['required', 'numeric', 'decimal:0,2', 'min:0.1',
                'max:100'],
            'pickup' => ['boolean'],
            'delivery_address' => ['nullable', 'required_if_accepted:pickup',
                'string', 'max:200'],
            'rack' => ['nullable', 'regex:/^[A-Z]-[0-9]{2}$/'],
            'note' => ['nullable', 'string', 'max:200'],
        ];
    }
```

Kode 9.13 (Lanjutan) *app/Http/Requests/StoreOrderRequest.php: daftar periksa untuk pesanan baru, dengan perapian, aturan, pesan, dan nama isian.*

```
    public function messages(): array
    {
        return [
            'rack.regex' => 'A rack looks like B-03: a letter, a dash, two digits.',
            'delivery_address.required_if_accepted' =>
                'Where should the laundry be delivered?',
        ];
    }

    public function attributes(): array
    {
        return ['outlet_id' => 'outlet', 'service_id' => 'service',
            'customer_id' => 'customer'];
    }
}
```

Metode-metodenya berjalan dengan urutan tetap. `authorize()` berjalan pertama; mengembalikan `false` akan menjawab 403 sebelum aturan mana pun diperiksa, dan Bab 10 mengganti `true` dengan pemeriksaan yang sesungguhnya. `prepareForValidation()` kemudian merapikan masukan mentah. Kasir di Indonesia mengetik “4,5” sama seringnya dengan “4.5”, karena koma adalah pemisah desimal dalam bahasa Indonesia, sehingga koma diubah menjadi titik. Kotak centang yang tidak dicentang tidak mengirim apa pun, sehingga `boolean('pickup')` mengubah “tidak ada” menjadi `false` dan “1” menjadi `true`. Rak yang diketik “b-03” menjadi “B-03”, dan rak yang kosong menjadi `null`. Aturan-aturan kemudian memeriksa nilai yang sudah rapi.

Sebagian besar aturannya berasal dari Bab 8. `exists:outlets,id` adalah aturan baru yang memeriksa bahwa kode gerai adalah baris di tabel `outlets`; `Rule::exists('services', 'id')->where('is_active', true)` memeriksa lebih tepat bahwa layanannya ada *dan* masih ditawarkan. `decimal:0,2` mengizinkan paling banyak dua angka desimal, sesuai dengan yang dapat disimpan kolom `quantity`, dan `required_if_accepted:pickup` membuat alamat pengantaran wajib hanya bila kotak jemput-antar dicentang. `messages()` mengganti dua pesan umum Laravel dengan kata-kata yang diperlukan kasir, dan `attributes()` membuat pesan lainnya berbunyi “The outlet field is required” alih-alih “The outlet id field is required” [8]. Bila aplikasi diberi paket bahasa Indonesia untuk pesan validasinya, kalimat yang sama tampil dalam bahasa Indonesia.

9.7.2 Aturan yang memerlukan pesanannya

Kode 9.14 adalah daftar periksa untuk pembayaran, dan ia memperlihatkan metode yang sulit diungkapkan dengan baik oleh versi *controller*: `after()`.

Kode 9.14 `app/Http/Requests/StorePaymentRequest.php`: aturan sederhana lebih dahulu, lalu pemeriksaan terhadap sisa tagihan.

```php
<?php

namespace App\Http\Requests;

use App\Enums\PaymentMethod;
use App\Support\Money;
use Illuminate\Foundation\Http\FormRequest;
use Illuminate\Validation\Rule;
use Illuminate\Validation\Validator;

class StorePaymentRequest extends FormRequest
{
    public function authorize(): bool
    {
        return true;
    }

    public function rules(): array
    {
        return [
            'amount' => ['required', 'integer', 'min:1000'],
            'method' => ['required', Rule::enum(PaymentMethod::class)],
        ];
    }

    // A check that needs the order itself, run after the rules above pass.
    public function after(): array
    {
        return [
            function (Validator $validator) {
                if ($validator->errors()->isNotEmpty()) {
                    return;
                }
                $order = $this->route('order')->load('payments');
                if ((int) $this->input('amount') > $order->balance()) {
                    $validator->errors()->add('amount', 'Only '
                        . Money::rp($order->balance()) . ' is still owed.');
                }
            },
        ];
    }
}
```

`after()` mengembalikan daftar fungsi yang dipanggil Laravel setelah aturan-aturan diperiksa, masing-masing dengan objek validatornya. Fungsi kita langsung kembali bila aturan sederhana sudah gagal: tidak ada gunanya membandingkan jumlah yang bukan angka dengan sisa tagihan. Jika tidak, ia mengambil pesanan dari rute, seperti yang dilakukan *middleware*, memuat pembayarannya, dan menambahkan galat pada `amount` bila jumlahnya melebihi sisa tagihan. Galat itu bergabung dengan galat lainnya: kasir melihat "Only Rp 16.000 is still owed." di bawah isian jumlah, persis seperti pesan dari aturan biasa, dan *controller* tidak pernah dipanggil. Bab 8 memecahkan masalah yang sama dengan aturan `max:` yang dibangun dari sisa tagihan di dalam *controller*; *form request* menyimpan pengetahuan itu di samping aturan pembayaran lainnya, tempat pemrogram berikutnya akan mencarinya.

9.7.3 Controller yang memercayai masukannya

Setelah daftar periksa dipindahkan, *controller* menjadi pendek. **Kode 9.15** adalah seluruh *controller* pembayaran.

Kode 9.15 `app/Http/Controllers/PaymentController.php`: ketika `store` berjalan, permintaannya sudah diperiksa.

```php
<?php

namespace App\Http\Controllers;

use App\Http\Requests\StorePaymentRequest;
use App\Models\Order;
use App\Support\Money;

class PaymentController extends Controller
{
    // POST /orders/5/payments: the request object has checked everything.
    public function store(StorePaymentRequest $request, Order $order)
    {
        $data = $request->validated();
        $order->payments()->create([...$data, 'paid_at' => now()]);

        return back()->with('message',
            'Payment of ' . Money::rp((int) $data['amount']) . ' recorded.');
    }
}
```

Satu-satunya perubahan pada tanda tangan metode ini adalah tipe parameter pertamanya: `StorePaymentRequest` alih-alih `Request`. Hanya itu yang diperlukan. *Service container* Laravel melihat tipenya, membangun *form request*, memvalidasinya, dan baru kemudian memanggil `store`. `$request->validated()` mengembalikan *hanya* isian yang memiliki aturan, di sini `amount` dan `method`; isian lain yang mungkin ditambahkan formulir jahat, seperti `order_id` atau `paid_at`, ditinggalkan, sehingga tidak pernah dapat mencapai `create()`. **Kode 9.16** memperlihatkan *controller* pesanan, tempat `store` kini menerima `StoreOrderRequest` dan `create` juga mengirim daftar gerai ke formulir.

Kode 9.16 `app/Http/Controllers/OrderController.php`: *controller* Bab 8 dengan `StoreOrderRequest` di `store` dan daftar gerai di `create`.

```php
<?php

namespace App\Http\Controllers;

use App\Domain\OrderWorkflow;
use App\Domain\PriceCalculator;
use App\Enums\OrderStatus;
use App\Http\Requests\StoreOrderRequest;
use App\Models\Customer;
use App\Models\Order;
use App\Models\Outlet;
use App\Models\Service;
use App\Support\Money;
use DomainException;
use Illuminate\Http\Request;

class OrderController extends Controller
{
    public function index(Request $request)
    {
        $status = OrderStatus::tryFrom((string) $request->query('status'));
        $orders = Order::query()
            ->select(['id', 'customer_id', 'service_id', 'total', 'status',
                'received_at'])
            ->with(['customer:id,name', 'service:id,description'])
            ->withSum('payments', 'amount')     // adds payments_sum_amount
            ->when($status, fn ($query) => $query->where('status', $status))
            ->orderByDesc('received_at')->orderByDesc('id')
            ->paginate(20)
            ->withQueryString();

        return view('orders.index', compact('orders', 'status'));
    }

    public function show(Order $order)
    {
        $order->load(['customer', 'service',
            'payments' => fn ($q) => $q->orderBy('paid_at')]);

        return view('orders.show', compact('order'));
    }
```

Kode 9.16 (Lanjutan) *app/Http/Controllers/OrderController.php: controller Bab 8 dengan StoreOrderRequest di store dan daftar gerai di create.*

```
    public function create(Request $request)
    {
        return view('orders.create', [
            'customers' => Customer::orderBy('name')->get(),
            'outlets' => Outlet::orderBy('id')->get(),
            'services' => Service::where('is_active', true)
                ->orderBy('id')->get(),
            'customerId' => $request->integer('customer_id'),
        ]);
    }

    // Only checked input arrives here; see StoreOrderRequest.
    public function store(StoreOrderRequest $request)
    {
        $data = $request->validated();
        $service = Service::findOrFail($data['service_id']);

        $order = Order::create([
            ...$data,
            'total' => PriceCalculator::total($service->per_kg,
                $service->price, (string) $data['quantity'], $data['pickup']),
            'status' => OrderStatus::Queued,
            'received_at' => now(),
        ]);

        return redirect()->route('orders.show', $order)->with('message',
            "Order #{$order->id} created: " . Money::rp($order->total) . '.');
    }

    public function advance(Order $order)
    {
        $balance = $order->total - (int) $order->payments()->sum('amount');
        try {
            $next = OrderWorkflow::advance($order->status, $balance);
        } catch (DomainException $e) {
            return back()->with('error', $e->getMessage());
        }
        $order->status = $next;
        if ($next === OrderStatus::Finished) {
            $order->finished_at = now();
        } else {
            $order->departed_at = now();
        }
        $order->save();
```

Kode 9.16 (Lanjutan) *app/Http/Controllers/OrderController.php: controller Bab 8 dengan StoreOrderRequest di store dan daftar gerai di create.*

```
        return back()->with('message',
            "Order #{$order->id} is now {$next->label()}.");
    }
}
```

`...$data` menebarkan isian yang sudah divalidasi ke dalam larik untuk `Order::create`, dan *controller* menambahkan tiga isian yang tidak pernah dipilih pelanggan: total dari `PriceCalculator`; status, yang selalu *queued*; dan waktu penerimaan. Formulir Bab 8 memerlukan satu isian lagi, yaitu daftar gerai; `resources/views/orders/create.blade.php` di folder bab ini menambahkannya di atas daftar layanan.

9.8 Menjalankannya

Setelah menyalin berkas bab ini ke atas proyek Bab 8, jalankan langkah-langkah di `README.md` bab ini. Laravel 13 memerlukan PHP 8.3 atau yang lebih baru; PHP 8.5, yang dirilis pada November 2025, adalah versi terkini dan versi yang dianjurkan buku ini [12]. Pada basis data yang sudah berisi pesanan Bab 8 Anda, jalankan dahulu `php artisan migrate:status`: keenam berkas sebelumnya menunjukkan "Ran" dan migrasi gerai yang baru menunjukkan "Pending". `php artisan migrate` kemudian hanya menjalankan berkas itu, sebagai *batch* 3, dan setiap pesanan yang sudah ada kini milik gerai Rungkut. `php artisan migrate:status` mengonfirmasinya, dan `php artisan migrate:rollback` diikuti `php artisan migrate` adalah cara yang baik untuk membuktikan kepada diri sendiri bahwa `down()` benar-benar membatalkan `up()`.

Untuk melihat fitur baru bekerja, bangun ulang dengan data demo (`php artisan migrate:fresh --seed`), jalankan server (`php artisan serve --port=4090`), lalu coba empat hal berikut.

1. Buka **Orders**, lalu **New order**, pilih Budi, gerai Wiyung, "Wash, dry & fold", ketik `2,5` sebagai jumlah, centang *pickup*, dan biarkan alamat kosong. Formulir kembali dengan "Where should the laundry be delivered?" dan tetap memuat semua yang sudah Anda ketik. Tambahkan alamat lalu simpan: pesanan dibuat untuk 3 kg, batas minimumnya, ditambah biaya jemput-antar, Rp 34.000.
2. Pada pesanan itu, catat pembayaran Rp 40.000. Halamannya kembali dengan "Only Rp 34.000 is still owed."
3. Bayar Rp 34.000, lalu tekan **Mark finished** dan **Mark picked up**. Pesanan kini berstatus *departed*. Buka halaman itu lagi di tab peramban kedua dan coba catat pembayaran di sana: Anda kembali ke pesanan dengan catatan merah "Order #… has left the shop and is closed."
4. Buka *developer tools* peramban di halaman mana pun, pilih tab Network, dan klik permintaan halamannya: bagian Timing menampilkan nilai `app` dari `AddServerTiming`.

Langkah-langkah ini ditulis berdasarkan kode dan dokumentasi Laravel 13; lingkungan tempat buku ini ditulis tidak dapat memasang Laravel, sebagaimana dijelaskan `README.md` bab ini, sehingga tidak ada tangkapan layar aplikasi ini yang ditampilkan. Demo *pipeline* PHP biasa di Subbab 9.6.1 sudah dijalankan, dan keluarannya adalah keluaran yang sesungguhnya.

9.9 Membuatnya Lebih Baik

Parameter middleware. *Middleware* rute dapat menerima argumen setelah titik dua: `->middleware('throttle:5,1')` mengirim "5" dan "1" ke pembatas laju Laravel. *Middleware* Anda sendiri menerima parameternya sebagai argumen tambahan `handle` setelah `$next`, selalu berupa *string*. Latihan 9.3 menulis *middleware* yang mencatat permintaan yang lambat, dengan batas dalam milidetik sebagai parameter.

Middleware pada controller. Laravel 13 memungkinkan *controller* menyatakan *middleware*-nya dengan atribut PHP, seperti atribut `#[Middleware(...)]` pada kelas atau pada metode [1]. Cara itu menyimpan aturan di dekat kode yang dijaganya; berkas rute menyimpannya di dekat URL. Keduanya baik, tetapi pilihlah satu untuk setiap proyek, agar tidak ada yang harus mencari di dua tempat.

Data tervalidasi yang lebih aman. `$request->safe()->only(['amount', 'method'])` dan `$request->safe()->except([...])` mengembalikan bagian tertentu dari data yang sudah divalidasi, berguna ketika *controller* perlu memperlakukan sebagian isian secara berbeda dari yang lain [8].

Migrasi data yang besar. Migrasi gerai mengisi semua pesanan lama dengan satu perintah `UPDATE`, dan itu wajar untuk ribuan baris. Ketika setiap baris memerlukan nilainya sendiri, `lazyById()` pada *query builder* menelusuri tabel dalam potongan seribu baris tanpa memuat semuanya ke memori; Latihan 9.2 memakainya.

Factory dalam uji. *State* `finished()` dan `departed()` ditulis untuk uji sama banyaknya dengan untuk *seeder*. Bab 10 membuat pesanan yang sudah pergi dalam satu baris, `Order::factory()->departed()->create()`, lalu memeriksa bahwa API menolak pembayaran padanya.

9.10 Memakainya Sungguhan

Keempat alat bab ini dapat dibawa ke usaha kecil mana pun yang memiliki meja layanan, entah penjahit, percetakan, bengkel sepeda motor, atau meja resep apotek.

- **Mulailah riwayat migrasi sejak hari pertama.** Bahkan purwarupa satu tabel sebaiknya membuat tabelnya dengan migrasi. Pada hari muncul komputer kedua, pengembang kedua, atau layanan *hosting*, riwayatnya sudah tersedia. Ketika sebuah usaha berjalan di atas basis data yang dibuat dengan tangan, tulislah satu migrasi yang menggambarkan tabel-tabel saat ini, tandai sebagai sudah dijalankan di basis data itu, lalu lanjutkan dari sana.
- **Isikan data rujukan yang sungguhan dari berkas yang dapat dibaca pemilik.** Harga, cabang, kategori produk, dan tarif pajak berubah, dan pemiliklah yang mengetahuinya. Berkas CSV di `database/data`, dibaca oleh *seeder* yang *idempotent*, memungkinkan pemilik membetulkan harga dan pengembang menerapkan pembetulan itu dengan satu perintah, seperti yang dilakukan Profitina Laundry dengan `Service.csv`.
- **Rekalah data yang realistis, jangan pernah orang yang nyata.** Pelanggan rekaan harus jelas rekaan dan tidak boleh sampai ke basis data sungguhan. Penjaga `environment('local', 'testing')` di **Kode 9.7** adalah batas minimum. Ketika data sungguhan disalin ke laptop untuk menelusuri masalah, data itu memuat data pribadi, yaitu nama, nomor telepon, dan alamat, yang dilindungi Undang-Undang Pelindungan Data Pribadi, UU PDP 27/2022; samarkan atau hapus data itu sebelum meninggalkan komputer toko.
- **Ubahlah setiap "ini tidak boleh terjadi lagi" menjadi *middleware* atau *form request*.** Bengkel yang tidak boleh menyerahkan sepeda motor sebelum tagihan lunas, percetakan yang tidak boleh mencetak ulang pekerjaan yang sudah "diserahkan", apotek yang tidak boleh mengubah resep yang sudah diserahkan: masing-masing adalah `EnsureOrderIsOpen` yang sama dengan kata-kata yang berbeda. Letakkan aturan itu di tempat yang dilewati setiap rute yang dapat melanggarnya.
- **Validasilah dengan bahasa meja kasir.** Pesan khusus dalam bahasa staf, dan perapian seperti aturan koma-menjadi-titik, hanya memerlukan beberapa baris tetapi mencegah banyak kekeliruan di meja kasir [13].

9.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Menyunting migrasi yang sudah dijalankan.** Perubahannya diam-diam tidak berbuat apa pun di basis data yang sudah menjalankan versi lamanya. Tulislah migrasi baru.
- **Menambahkan kolom wajib ke tabel yang berisi baris.** `$table->string('outlet_id', 3)` tanpa `nullable()` atau nilai bawaan gagal pada tabel yang berisi pesanan. Tambahkan sebagai *nullable*, isi, lalu perketat kemudian bila perlu.

- **Memakai model di dalam migrasi.** `Order::query()->update(...)` di dalam migrasi berjalan hari ini dan rusak pada hari model itu mendapat *cast* atau *global scope* yang tidak dapat dipenuhi tabel lama. Pakailah `DB::table()`.
- **`down()` yang tidak membatalkan `up()`.** Membuang kolom tetapi tidak membuang *foreign key*-nya, atau melupakan satu dari dua kolom, membuat `migrate:rollback` gagal di tengah jalan dan meninggalkan basis data di antara dua keadaan. Ujilah pembatalannya sekali.
- **Data rekaan di basis data sungguhan.** `DatabaseSeeder` yang memanggil *seeder* demo tanpa syarat memasukkan pelanggan rekaan ke sistem toko begitu seseorang menjalankan `db:seed` di sana. Jagalah dengan lingkungan aplikasi.
- **Nilai acak di tempat yang ditentukan aturan.** *Factory* yang mengarang total, atau pesanan yang sudah pergi tetapi belum dibayar, menciptakan data yang tidak mungkin dihasilkan aplikasi itu sendiri, dan laporan serta uji yang dibangun di atasnya berbohong. Hitunglah isian turunan dengan aturan yang sungguhan, seperti yang dilakukan `OrderFactory::configure`.
- **Lupa `return $next($request)`.** *Middleware* yang memeriksa sesuatu lalu lupa memanggil `$next` menjawab setiap permintaan dengan respons kosong. Setiap jalur melalui `handle` harus mengembalikan respons: responsnya sendiri atau respons dari `$next`.
- **Mendaftarkan *middleware* secara global padahal ia memerlukan parameter rute.** *Middleware* global berjalan sebelum perutean, sehingga `$request->route('order')` di sana bernilai `null`. `EnsureOrderIsOpen` tempatnya di rute, sebagai alias.
- **Memakai `$request->all()` setelah validasi.** Metode itu mengembalikan semua isian yang dikirim, termasuk yang tidak memiliki aturan. Pakailah `$request->validated()` atau `$request->safe()`.
- **Memeriksa basis data di `rules()` tanpa penjaga.** Aturan yang memanggil `$this->route('order')->balance()` berjalan baik sampai datang permintaan dengan jumlah yang tidak sah; letakkan pemeriksaan yang bergantung pada isian lain di `after()`, di belakang penjaga "aturan sederhana sudah lolos".

9.12 Ringkasan Bab

- Migrasi adalah perubahan basis data yang tercatat dan dapat dibatalkan. Tabel `migrations` mengingat berkas mana yang sudah dijalankan dan di *batch* mana; `migrate` menjalankan berkas yang belum ada sebagai *batch* baru dan `migrate:rollback` membatalkan *batch* terakhir. Migrasi yang sudah dijalankan di tempat lain tidak pernah disunting, hanya disusul migrasi baru.
- Mengubah tabel yang berisi data berarti menambahkan kolom sebagai *nullable* atau dengan nilai bawaan, mengisinya (di sini: semua pesanan lama milik gerai O01) dengan *query builder*, dan menulis `down()` yang benar-benar membatalkan perubahan.
- *Factory* menggambarkan satu rekaman yang masuk akal, dengan Faker untuk nilai rekaan, *closure* dan *factory* terkait untuk *foreign key*, `afterMaking` untuk isian yang dihitung aturan sungguhan, dan *state* untuk variasi yang diberi nama. *Seeder* menentukan apa yang dibuat dan dengan urutan apa.
- Data rujukan sungguhan (daftar harga, gerai) diisikan secara *idempotent* dengan `updateOrCreate` di setiap basis data; data rekaan hanya di komputer pengembang dan komputer uji.
- *Middleware* membungkus *controller* seperti lapisan bawang. *Middleware* "sebelum" dapat menghentikan permintaan dengan tidak memanggil `$next`; *middleware* "sesudah" mengubah respons yang dikembalikan `$next`. *Middleware* didaftarkan di `bootstrap/app.php`, dalam grup atau dengan alias, dan dipasang pada rute satu per satu atau berkelompok.

- `EnsureOrderIsOpen` menutup pesanan yang sudah pergi untuk setiap rute yang mengubah pesanan, menjawab manusia dengan catatan dan program dengan 409; `AddServerTiming` mengukur setiap halaman.
- *Form request* memuat daftar periksa satu jenis permintaan: `authorize`, `prepareForValidation`, `rules`, `messages`, `attributes`, dan `after`. *Controller* menerimanya sebagai parameter bertipe dan hanya berjalan bila semuanya lolos, dan `validated()` hanya memberinya isian yang sudah diperiksa.

9.13 Soal Latihan

1. Sebuah laptop baru belum menjalankan migrasi apa pun. Dimas menjalankan `php artisan migrate` sekali dengan keenam berkas proyek ini tersedia, lalu `php artisan migrate:rollback`. Tabel dan kolom apa yang ada sesudahnya? Jawablah pertanyaan yang sama untuk komputer toko, yang menjalankan berkas Bab 7, lalu Bab 8, lalu bab ini, masing-masing pada hari yang berbeda.
2. Toko ingin mencatat kapan setiap pesanan *dijanjikan* kepada pelanggan. Tulislah migrasi yang menambahkan kolom *timestamp* `promised_at` yang *nullable* ke `orders`, mengisinya untuk pesanan yang sudah ada dengan waktu penerimaan ditambah dua hari, dan dapat dibatalkan. Telusurilah tabel per potongan, bukan dengan memuat semuanya sekaligus.
3. Tulislah *middleware* `LogSlowRequests` yang menerima batas dalam milidetik sebagai parameter (bawaan 500) dan menulis peringatan ke log Laravel untuk setiap permintaan yang lebih lambat dari batas itu. Tunjukkan cara mendaftarkannya dengan alias `slow` dan memasangnya dengan batas 300 md pada daftar pesanan.
4. Seorang kasir mengetik rak sebagai “b-3”. Ikutilah perjalanannya melalui `StoreOrderRequest`: menjadi apa ia setelah `prepareForValidation`, aturan mana yang gagal, dan pesan apa yang dilihat kasir? Apakah “B-03” (dengan spasi di akhir) akan lolos?
5. Seseorang menjalankan `php artisan db:seed` dua kali di laptop, tanpa `migrate:fresh` di antaranya. Apa yang terjadi pada `PriceListSeeder`, `DemoSeeder`, dan `BusyMonthSeeder`? Mana yang harus diubah jika jalan berulang harus berhasil, dan bagaimana caranya?
6. Pada *pipeline* di **Kode 9.8**, tukarlah kedua lapisannya sehingga `$orderOpen` berada paling luar. Apa yang dicetak program untuk pesanan yang sudah pergi, dan mengapa urutan lapisan yang baru lebih buruk bagi `AddServerTiming` dalam aplikasi yang sesungguhnya?

9.14 Penyelesaian Soal Latihan

1. `migrate` yang hanya sekali itu memasukkan keenam berkas ke *batch* 1, karena semuanya dijalankan bersama. `migrate:rollback` membatalkan *batch* terakhir, yaitu seluruh *batch* 1, mulai dari yang terbaru: kolom gerai, kolom rak, lalu *payments*, *orders*, *services* beserta *outlets*, dan *customers*. Laptop itu tinggal memiliki tabel-tabel milik Laravel sendiri dan tabel `migrations` yang kosong. Di komputer toko keenam berkas tersebar dalam tiga *batch* (empat berkas Bab 7 di *batch* 1, berkas Bab 8 di *batch* 2, berkas bab ini di *batch* 3), sehingga `migrate:rollback` yang sama hanya membatalkan *batch* 3: kolom `outlet_id` beserta *foreign key*-nya dibuang, dan semua yang lain, termasuk kolom rak dan setiap pesanan, tetap ada. Perintah yang sama melakukan hal yang sangat berbeda di dua komputer, dan karena itulah `migrate:status` harus selalu dijalankan lebih dahulu.

2. **Kode 9.17** menambahkan kolom sebagai *nullable*, tepat setelah `received_at`, lalu mengisinya pada langkah kedua. `lazyById()` membaca tabel dalam potongan 1.000 baris, diurutkan menurut

id, sehingga bahkan tabel berisi sejuta pesanan tidak pernah harus muat di memori sekaligus; setiap baris diperbarui dengan nilainya sendiri. Aritmetika tanggalnya memakai `strtotime` PHP pada teks yang tersimpan, bukan Carbon dan model `Order`, mengikuti aturan bahwa migrasi hanya memakai *query builder*. `down()` membuang kolomnya; tidak ada *foreign key* yang terlibat.

Kode 9.17 `2026_10_01_000001_add_promised_at_to_orders_table.php`: kolom baru yang diisi baris demi baris, per potongan.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Schema;

// Exercise 9.2: when the laundry was promised to the customer.
return new class extends Migration
{
    public function up(): void
    {
        Schema::table('orders', function (Blueprint $table) {
            $table->timestamp('promised_at')->nullable()->after('received_at');
        });

        // Old orders: the usual promise, two days after they came in.
        DB::table('orders')->whereNull('promised_at')->lazyById()
            ->each(function (object $order) {
                DB::table('orders')->where('id', $order->id)->update([
                    'promised_at' => date('Y-m-d H:i:s',
                        strtotime($order->received_at . ' +2 days')),
                ]);
            });
    }

    public function down(): void
    {
        Schema::table('orders', function (Blueprint $table) {
            $table->dropColumn('promised_at');
        });
    }
};
```

3. Kode 9.18 adalah *middleware*-nya. Parameternya datang sebagai argumen tambahan `$limitMs`, berupa *string*, setelah `$next`; nilai bawaan "500" dipakai bila rute tidak memberikannya. Ini *middleware* "sesudah": ia mengukur di sekitar `$next` dan baru kemudian memutuskan apakah akan menulis ke log dengan `Log::warning`, yang catatannya muncul di `storage/logs/laravel.log`.

Kode 9.18 `app/Http/Middleware/LogSlowRequests.php`: *middleware* rute dengan parameter.

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Log;
use Symfony\Component\HttpFoundation\Response;

// Exercise 9.3: route middleware with a parameter, e.g. 'slow:300'.
class LogSlowRequests
{
    public function handle(Request $request, Closure $next,
        string $limitMs = '500'): Response
    {
        $start = hrtime(true);
        $response = $next($request);
        $ms = (hrtime(true) - $start) / 1e6;

        if ($ms > (float) $limitMs) {
            Log::warning('Slow request', [
                'url' => $request->fullUrl(),
                'ms' => round($ms, 1),
                'limit' => (int) $limitMs,
            ]);
        }

        return $response;
    }
}
```

Daftarkan di `bootstrap/app.php` di dalam `withMiddleware` dengan menambahkan `'slow' => LogSlowRequests::class` ke larik yang diberikan kepada `alias`, dengan `use App\Http\Middleware\LogSlowRequests;` di bagian atas berkas. Kemudian pasang di `routes/web.php` dengan menambahkan `->middleware('slow:300')` pada definisi daftar pesanan; karena daftar itu didefinisikan oleh `Route::resource`, tulislah `Route::resource('orders', OrderController::class)->only([...])->middleware('slow:300')`, yang menerapkannya pada keempat rute *resource*, atau definisikan rute indeks tersendiri bila hanya daftar itu yang ingin diawasi.

4. `prepareForValidation` memangkas spasi dan mengubah teks menjadi huruf kapital, sehingga “b-3” menjadi “B-3”. Aturan `regex:/^[A-Z]-[0-9]{2}$/` mewajibkan tepat dua angka setelah tanda hubung, sehingga “B-3” gagal, dan kasir melihat pesan khusus “A rack looks like B-03: a letter, a dash, two digits.” “B-03” lolos: `trim` membuang spasinya dan hasilnya, “B-03”, cocok.

5. `PriceListSeeder` memakai `updateOrCreate` dengan kunci kode, sehingga jalan kedua menemukan setiap layanan dan gerai lalu memperbaruinya ke nilai yang sama: tidak ada yang berubah, tidak ada yang gagal. `DemoSeeder` memakai `Customer::create` dengan nomor telepon tetap; jalan kedua mencoba menyisipkan nomor telepon Sari sekali lagi dan berhenti dengan galat *unique constraint*, sehingga jalannya gagal di titik itu. `BusyMonthSeeder` akan menambahkan empat puluh pelanggan lagi beserta pesanannya (`unique()` Faker hanya mengingat nomor dalam satu kali jalan, sehingga bentrokan dengan nomor dari jalan pertama mungkin terjadi walaupun kecil kemungkinannya). Agar jalan berulang berhasil, ubahlah `DemoSeeder` menjadi `Customer::firstOrCreate(['phone' => ...], [...])` dan lewati pembuatan pesanan bila pelanggannya sudah ada, dan terimalah bahwa `BusyMonthSeeder` menambah data setiap kali, atau jagalah dengan pemeriksaan seperti `if (Customer::count() > 10) return;`.

6. Dengan `$orderOpen` paling luar, pesanan yang sudah pergi berhenti di lapisan luar, yang tidak pernah memanggil `$next`; lapisan pengukur waktu berada di dalamnya dan tidak pernah berjalan. Program mencetak "order.open: departed, stop here" dan jawaban `{"status":409}` tanpa nilai Server-Timing. Dalam aplikasi sesungguhnya, ini berarti permintaan yang ditolak sama sekali tidak diukur, dan waktu yang dihabiskan lapisan luar tidak pernah masuk ke pengukuran. Lapisan yang mengukur atau mencatat sebaiknya berada sejauh mungkin di luar, dan karena itulah `AddServerTiming` bergabung dengan grup `web`, sedangkan lapisan yang menolak sebaiknya berada sedalam yang diperlukan agar memiliki informasi yang diperiksanya, di sini setelah perutean.

Daftar Pustaka

[1] Laravel, "Release Notes," *Laravel 13.x Documentation*, Mar. 17, 2026. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/releases

[2] Laravel, "Database: Migrations," *Laravel 13.x Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/migrations

[3] P. Sadalage and M. Fowler, "Evolutionary Database Design," martinfowler.com, May 2016. Accessed: Sep. 25, 2026. [Online]. Available: https://martinfowler.com/articles/evodb.html

[4] FakerPHP, "FakerPHP / Faker documentation." Accessed: Sep. 25, 2026. [Online]. Available: https://fakerphp.org/

[5] Laravel, "Eloquent: Factories," *Laravel 13.x Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/eloquent-factories

[6] Laravel, "Database: Seeding," *Laravel 13.x Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/seeding

[7] Laravel, "Middleware," *Laravel 13.x Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/middleware

[8] Laravel, "Validation," *Laravel 13.x Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/validation

[9] Laravel, "Routing: Route Model Binding," *Laravel 13.x Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/routing#route-model-binding

[10] R. Fielding, M. Nottingham, and J. Reschke, Eds., "HTTP Semantics," RFC 9110, IETF, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9110

[11] MDN Web Docs, "Server-Timing header," Mozilla. Accessed: Sep. 25, 2026. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Server-Timing

[12] The PHP Group, "PHP 8.5 Release Announcement," php.net, Nov. 20, 2025. Accessed: Sep. 25, 2026. [Online]. Available: https://www.php.net/releases/8.5/en.php

[13] OWASP Foundation, "Input Validation Cheat Sheet," *OWASP Cheat Sheet Series*. Accessed: Sep. 25, 2026. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

Bab 10

Laravel III: Autentikasi, API Resource, dan Pengujian

Di akhir Bab 9 aplikasi Profitina Laundry sudah mengenal gerai-gerainya, dapat mengisi dirinya dengan satu bulan pesanan yang masuk akal, dan menolak mengubah pesanan yang sudah meninggalkan toko. Masih ada tiga celah. Siapa pun yang dapat menjangkau komputer kasir dapat membuka setiap halaman, termasuk daftar harga. Hanya orang yang memakai peramban yang dapat menggunakannya, padahal para sopir yang menjemput dan mengantar cucian ingin mencatat pekerjaan mereka di ponsel. Dan setiap pemeriksaan bahwa aplikasi berjalan benar dilakukan dengan tangan, dengan mengklik. Bab ini menutup ketiga celah itu. Di akhir bab ini Anda akan mampu menjelaskan perbedaan antara autentikasi dan otorisasi; menambahkan masuk dan keluar bagi staf dengan *session guard* Laravel, serta memberi pemilik hak yang tidak dimiliki kasir; menerbitkan, memakai, dan mencabut token API dengan Laravel Sanctum agar program lain dapat memakai aplikasi dengan aman; membentuk setiap jawaban JSON dengan *API resource* alih-alih mengirim baris basis data apa adanya; serta menulis uji unit dan uji fitur yang memeriksa aturan harga, proses masuk, hak akses, dan API secara otomatis, dalam hitungan detik, setiap kali ada perubahan.

10.1 Daftar Harga yang Dapat Diubah Siapa Saja

Suatu Sabtu siang di bulan September, gerai Keputih mengenakan harga Rp 6.000 per kilogram kepada seorang pelanggan untuk layanan “Wash, dry & fold”, bukan Rp 8.000. Tidak ada yang berniat mengubah harga. Seorang kasir paruh waktu yang baru membuka halaman **Prices** untuk *melihat* harga, seperti yang diminta pemilik, lalu mengklik **Edit** pada baris yang salah dan menyimpannya. Aplikasi Bab 9 tidak menanyakan siapa dirinya, sehingga tidak tahu bahwa ia seharusnya tidak dapat melakukan itu, dan sesudahnya tidak dapat mengatakan siapa yang melakukannya.

Pada minggu yang sama Bu Wati mempekerjakan dua sopir untuk layanan jemput-antar yang tarifnya ditetapkan Bab 3 sebesar Rp 10.000. Para sopir berada di jalan sepanjang hari. Mereka perlu melihat pesanan yang harus dijemput, menandai kantong yang sudah diantar, dan mencatat pembayaran yang dilakukan di depan pintu pelanggan, dan mereka akan melakukannya dengan aplikasi ponsel kecil yang sedang ditulis keponakan Dimas. Aplikasi itu bukan peramban: ia tidak dapat mengisi formulir Blade atau mengklik tombol Bab 7, dan berbahaya pula membiarkannya mencoba. Ia memerlukan *API*, yaitu sekumpulan URL yang menerima dan mengembalikan JSON, serta cara untuk membuktikan bahwa ponsel itu milik sopir toko.

Lalu ada Dimas. Setiap perubahan yang ia buat, entah aturan baru, kolom baru, atau halaman baru, ia periksa dengan mengklik seluruh aplikasi: membuat pesanan, membayarnya, memajukannya. Semua itu memakan dua puluh menit, sehingga makin lama makin jarang ia lakukan, dan pada Bab 9 sebuah perubahan pada formulir pembayaran nyaris merusak aturan pengambilan seandainya Bu Wati tidak melihat total yang salah keesokan paginya.

Laravel memiliki jawaban baku untuk setiap masalah itu [1]. *Autentikasi*, yaitu mencari tahu siapa yang membuat permintaan, sudah tersedia: sebuah *guard* yang memeriksa alamat surel dan kata sandi, mengingat orangnya dalam sesi, dan memungkinkan *middleware* menolak tamu [2]. *Otorisasi*, yaitu memutuskan apa yang boleh dilakukan orang itu, dikerjakan oleh *gate* dan *policy* [3]. Untuk program, *Laravel Sanctum* menerbitkan token API, yaitu *string* acak yang panjang yang disimpan ponsel dan dikirim bersama setiap permintaan [4]. *API resource* menentukan dengan tepat isi setiap jawaban JSON [5]. Dan perangkat pengujian Laravel, yang dibangun di atas PHPUnit, mengirim permintaan pura-pura melalui seluruh aplikasi dan memeriksa jawabannya tanpa peramban [6].

10.2 Gagasannya dengan Kata-Kata Sederhana

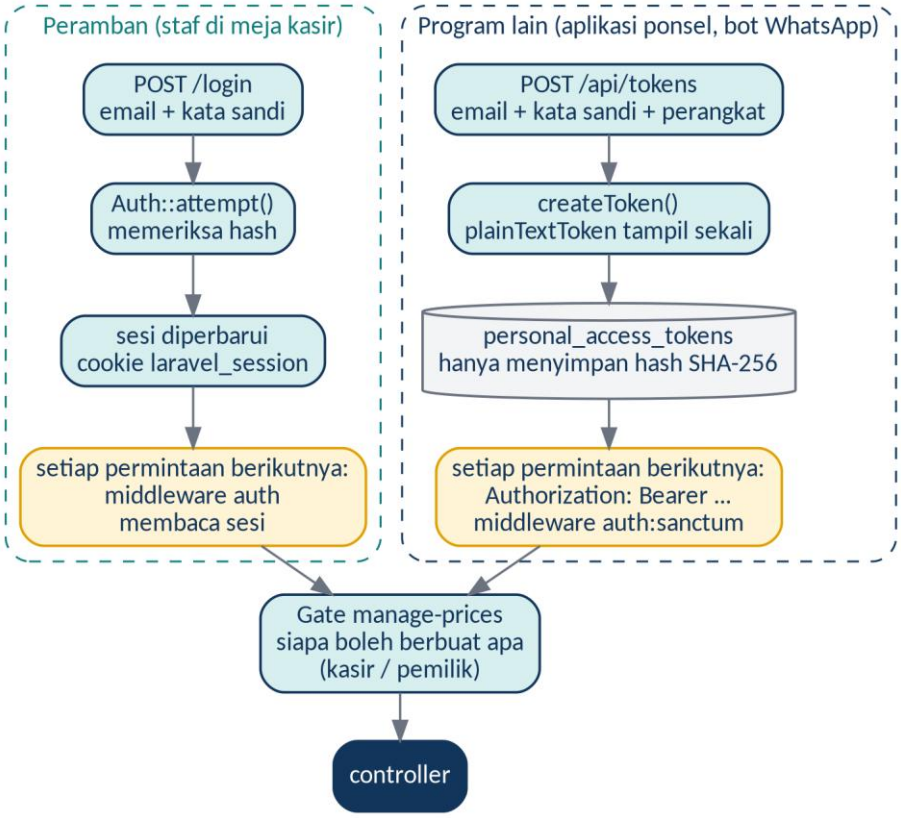
10.2.1 Siapa Anda, dan apa yang boleh Anda lakukan?

Dua pertanyaan mudah tertukar, dan sebagian besar keamanan bergantung pada upaya memisahkannya. *Autentikasi* bertanya "siapa Anda?", dan jawabannya adalah seseorang: Dimas, Bu Wati, seorang sopir. *Otorisasi* bertanya "bolehkah Anda melakukan ini?", dan jawabannya ya atau tidak untuk satu tindakan pada satu hal: bolehkah Dimas mengubah harga S0001? Pertanyaan pertama dijawab sekali, ketika seseorang masuk; pertanyaan kedua diajukan berulang-ulang, untuk setiap tindakan. HTTP memiliki kode status tersendiri untuk setiap kegagalan: *401 Unauthorized* berarti "saya tidak tahu siapa Anda" (walaupun namanya demikian, kode ini menyangkut autentikasi), dan *403 Forbidden* berarti "saya tahu siapa Anda, dan Anda tidak boleh melakukan ini" [7].

Untuk saat ini Profitina Laundry hanya memerlukan dua peran. *Pemilik* boleh melakukan segalanya, termasuk mengubah harga. *Kasir* boleh mendaftarkan pelanggan, menerima pesanan, mencatat pembayaran, dan memajukan pesanan, tetapi tidak boleh menyentuh daftar harga. Para sopir akan masuk sebagai kasir dari ponsel mereka.

10.2.2 Sesi untuk peramban, token untuk program

Peramban dan aplikasi ponsel membuktikan identitasnya dengan cara berbeda, yang diperlihatkan berdampingan pada **Gambar 10.1**.



Gambar 10.1 Dua cara membuktikan siapa yang bertanya: peramban masuk sekali lalu membawa *cookie* sesi; program menukar kata sandi dengan token lalu mengirimnya bersama setiap permintaan; keduanya berakhir di pemeriksaan otorisasi yang sama.

Peramban masuk sekali, melalui formulir. Laravel memeriksa kata sandi terhadap *hash* yang tersimpan, mengingat di dalam *sesi*, di server, bahwa pengunjung ini adalah Dimas, lalu memberi peramban sebuah *cookie* yang menyebut nama sesi itu. Setiap permintaan berikutnya membawa *cookie* itu secara otomatis, dan *middleware* `auth` menemukan Dimas lagi darinya. Cara ini nyaman bagi manusia, dan Bab 6 melakukan hal yang sama di ASP.NET Core dengan *cookie* Identity.

Program tidak dapat dipercaya memegang *cookie* yang diurus peramban untuknya, dan ia juga tidak seharusnya menyimpan kata sandi sopir. Sebagai gantinya, program mengirim alamat surel dan kata sandi sekali saja ke URL khusus dan menerima sebuah *token*: *string* acak yang panjang. Server hanya menyimpan *hash* SHA-256 dari token itu di tabel `personal_access_tokens`, sehingga orang yang mencuri salinan basis data pun tidak dapat memakai token di dalamnya [4]. Program mengirim token itu dalam *header* `Authorization: Bearer ...` bersama setiap permintaan, cara baku membawa token semacam itu di HTTP [8], dan *middleware* `auth:sanctum` menemukan penggunanya dari token itu. Satu token milik satu perangkat: ketika seorang sopir kehilangan ponselnya, pemilik menghapus token yang satu itu dan ponsel tersebut terkunci, sedangkan kata sandi sopir dan perangkatnya yang lain tetap berfungsi.

10.2.3 Resource: menentukan isi sebuah jawaban

Mengembalikan model Eloquent dari *controller* memang menghasilkan JSON, karena model tahu cara mengubah dirinya menjadi larik. Namun itu tetap kekeliruan untuk sebuah API. JSON-nya lalu memuat setiap kolom, termasuk yang tidak perlu dilihat ponsel, dalam bentuk basis data itu sendiri: `quantity` sebagai teks `"4.50"`, status sebagai apa pun yang disimpan basis data, *foreign key* alih-alih nama. Dan pada hari sebuah kolom diganti namanya, setiap aplikasi yang membaca JSON itu rusak.

API resource adalah kelas yang menyatakan jawabannya secara eksplisit [5]. Metode `toArray`-nya mendaftar isian yang dikirim, namanya, dan cara menghitung isian yang bukan kolom, seperti `paid` dan `balance`. Ia dapat memuat *resource* lain di dalamnya, sehingga sebuah pesanan membawa nama dan telepon pelanggannya, dan hanya menyertakan relasi bila *controller* sudah memuatnya. Basis data dapat berubah di balik *resource* sementara JSON-nya tetap sama.

10.2.4 Uji: sepasang tangan kedua yang tidak kenal lelah

Uji (*test*) adalah program kecil yang memakai aplikasi seperti yang dilakukan manusia lalu memeriksa hasilnya. *Uji unit* memeriksa satu kelas saja, misalnya `PriceCalculator`, tanpa basis data dan tanpa HTTP; ia berjalan dalam hitungan milidetik. *Uji fitur* mengirim permintaan pura-pura melalui seluruh aplikasi, yaitu perutean, *middleware*, *form request*, *controller*, dan basis data sungguhan, lalu memeriksa respons dan basis data sesudahnya [6]. Laravel memberi setiap uji fitur basis data yang baru dan kosong: *trait* `RefreshDatabase` menjalankan migrasi sekali dan membungkus setiap uji dalam transaksi yang dibatalkan di akhir, sehingga uji tidak pernah melihat data uji yang lain [9]. Pekerjaan yang dulu memakan dua puluh menit klik Dimas kini hanya beberapa detik, dan dikerjakan dengan cara yang sama setiap kali.

10.3 Pengguna dan Peran

Proyek Laravel yang baru sudah memiliki tabel `users`, model `User`, dan `UserFactory`. Bab ini memberi setiap pengguna sebuah peran. **Kode 10.1** mendaftar kedua peran sebagai *backed enum*, seperti `OrderStatus` di Bab 7.

Kode 10.1 `app/Enums/Role.php`: dua peran di toko.

```php
<?php

namespace App\Enums;

enum Role: string
{
    case Owner = 'owner';        // Bu Wati: everything, including prices
    case Cashier = 'cashier';    // the counter staff: orders and payments
}
```

Kode 10.2 menambahkan kolomnya. Setiap pengguna yang sudah ada menjadi kasir secara bawaan, dan itulah pilihan yang aman: akun yang seharusnya memiliki hak lebih mudah dinaikkan, sedangkan akun yang hak-haknya terlalu banyak dapat berbuat kerusakan sebelum ada yang menyadarinya.

Kode 10.2 `2026_10_04_000001_add_role_to_users_table.php`: setiap pengguna mendapat peran, kasir kecuali dinyatakan lain.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::table('users', function (Blueprint $table) {
            $table->string('role', 10)->default('cashier')->after('email');
        });
    }

    public function down(): void
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dropColumn('role');
        });
    }
};
```

Kode 10.3 adalah model `User`. Model ini memakai atribut `#[Fillable]` yang sama dengan model-model lain di buku ini, dan `#[Hidden]` agar *hash* kata sandi dan token "ingat saya" tidak ikut ke larik atau JSON mana pun yang dibuat dari seorang pengguna. Tiga *cast* melakukan pekerjaan penting. `'password' => 'hashed'` mengubah kata sandi polos yang diberikan kepada model menjadi *hash* sebelum disimpan, dengan algoritme yang diatur di `config/hashing.php`, yaitu bcrypt kecuali diubah [10]. `'role' => Role::class` mengubah teks yang tersimpan menjadi *enum*. Dan *trait* `HasApiTokens` dari Sanctum memberi setiap pengguna metode `createToken` dan relasi `tokens` yang dipakai di Subbab 10.5.

Kode 10.3 `app/Models/User.php`: pengguna dengan peran, kata sandi yang di-*hash*, dan token API.

```php
<?php

namespace App\Models;

use App\Enums\Role;
use Illuminate\Database\Eloquent\Attributes\Fillable;
use Illuminate\Database\Eloquent\Attributes\Hidden;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;

#[Fillable(['name', 'email', 'password', 'role'])]
#[Hidden(['password', 'remember_token'])]
class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;

    protected function casts(): array
    {
        return [
            'email_verified_at' => 'datetime',
            'password' => 'hashed',          // stored as a hash, never plain
            'role' => Role::class,
        ];
    }

    public function isOwner(): bool
    {
        return $this->role === Role::Owner;
    }
}
```

`UserFactory` bawaan proyek baru mendapat peran dan *state* `owner()`, pada **Kode 10.4**. Kata sandi polos "password" di-*hash* oleh *cast* model, sehingga setiap pengguna rekaan dapat masuk dengan kata sandi itu dalam uji.

Kode 10.4 `database/factories/UserFactory.php`: kasir secara bawaan, dan *state* `owner()`.

```php
<?php

namespace Database\Factories;

use App\Enums\Role;
use Illuminate\Database\Eloquent\Factories\Factory;
use Illuminate\Support\Str;

/**
 * @extends Factory<\App\Models\User>
 */
class UserFactory extends Factory
{
    public function definition(): array
    {
        return [
            'name' => fake()->name(),
            'email' => fake()->unique()->safeEmail(),
            'email_verified_at' => now(),
            'password' => 'password',           // hashed by the model's cast
            'role' => Role::Cashier,
            'remember_token' => Str::random(10),
        ];
    }

    public function owner(): static
    {
        return $this->state(fn (array $attributes) => ['role' => Role::Owner]);
    }
}
```

Terakhir, `DatabaseSeeder` membuat satu pemilik dan satu kasir di komputer pengembang, di samping data demo Bab 9 (**Kode 10.5**). Kedua akun ini memiliki kata sandi yang diketahui umum dan hanya ada untuk mencoba aplikasi; akun-akun sungguhan toko dibuat di komputer toko, satu untuk setiap orang, dengan kata sandi yang tidak diketahui orang lain.

Kode 10.5 `database/seeders/DatabaseSeeder.php`: dua akun demo, hanya di komputer pengembang.

```php
<?php

namespace Database\Seeders;

use App\Models\User;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    public function run(): void
    {
        $this->call(PriceListSeeder::class);      // real data: always

        if (app()->environment('local', 'testing')) {
            User::factory()->owner()->create(['name' => 'Bu Wati',
                'email' => 'owner@laundry.test']);
            User::factory()->create(['name' => 'Dimas',
                'email' => 'cashier@laundry.test']);
            $this->call([DemoSeeder::class, BusyMonthSeeder::class]);
        }
    }
}
```

10.4 Masuk ke Halaman Staf

10.4.1 Controller untuk masuk

Laravel menawarkan *starter kit* siap pakai yang menghasilkan layar masuk, pendaftaran, dan atur ulang kata sandi yang lengkap [2]. *Starter kit* itu sangat baik untuk produk baru, tetapi juga menghasilkan banyak kode sekaligus, sedangkan toko yang stafnya dibuat oleh pemilik tidak memerlukan pendaftaran terbuka. Bab ini menulis ketiga tindakan itu dengan tangan, pada **Kode 10.6**, yang sekaligus memperlihatkan dengan tepat apa yang dilakukan *starter kit* di dalamnya.

Kode 10.6 `app/Http/Controllers/Auth/LoginController.php`: menampilkan formulir, masuk, dan keluar.

```php
<?php

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Validation\ValidationException;

class LoginController extends Controller
{
    public function create()
    {
        return view('auth.login');
    }

    public function store(Request $request)
    {
        $credentials = $request->validate([
            'email' => ['required', 'email'],
            'password' => ['required', 'string'],
        ]);

        if (! Auth::attempt($credentials, $request->boolean('remember'))) {
            throw ValidationException::withMessages([
                'email' => 'These credentials do not match our records.',
            ]);
        }

        $request->session()->regenerate();      // a new session id after login

        return redirect()->intended(route('counter'));
    }

    public function destroy(Request $request)
    {
        Auth::logout();
        $request->session()->invalidate();
        $request->session()->regenerateToken();

        return redirect()->route('home');
    }
}
```

`store` lebih dahulu memvalidasi kedua isian, lalu memanggil `Auth::attempt`. Satu panggilan itu mencari pengguna berdasarkan surel, memeriksa kata sandi yang diketik terhadap *hash* yang tersimpan dengan perbandingan yang memakan waktu sama, baik kata sandinya hampir benar maupun sama sekali salah, lalu, bila cocok, mengingat pengguna itu dalam sesi; dengan `true`

sebagai argumen kedua ia juga memasang *cookie* "ingat saya" yang berumur panjang. Ketika percobaan gagal, metode ini melempar `ValidationException` dengan pesan pada isian `email`, sehingga formulir kembali persis seperti pada galat validasi lainnya. Pesan itu sengaja tidak menyebutkan apakah surel atau kata sandinya yang salah: memberi tahu orang asing "surel ini terdaftar" membantunya menebak [11].

Setelah berhasil masuk, `$request->session()->regenerate()` memberi sesi itu pengenal baru. Ini melindungi dari *session fixation*, serangan ketika seseorang menanamkan id sesi di peramban korban sebelum korban masuk, lalu memakai id yang sama sesudahnya. `redirect()->intended(...)` mengembalikan pengguna ke halaman yang semula ingin ia buka, jika *middleware* `auth` yang mengirimnya ke halaman masuk, atau ke meja kasir bila tidak. `destroy` melakukan kebalikannya: melupakan pengguna, membuang seluruh sesi, dan membuat token pelindung formulir yang baru, sehingga tidak ada bagian sesi lama yang dapat dipakai lagi. *View* `auth/login.blade.php` di folder bab ini adalah formulir Blade biasa dengan kedua isian, kotak "ingat saya", dan `@csrf`.

10.4.2 Menjaga rute

Kode 10.7 adalah berkas rute bab ini. Berkas ini memilah rute menjadi empat kelompok: halaman publik yang boleh dilihat siapa saja, rute masuk yang hanya boleh dipakai tamu, halaman staf di belakang `auth`, dan di dalamnya daftar harga di belakang aturan tambahan `can:manage-prices`.

Kode 10.7 `routes/web.php`: halaman publik, halaman masuk khusus tamu, halaman staf di belakang `auth`, dan daftar harga untuk pemilik.

```php
<?php

use App\Http\Controllers\Auth\LoginController;
use App\Http\Controllers\CounterController;
use App\Http\Controllers\CustomerController;
use App\Http\Controllers\HomeController;
use App\Http\Controllers\OrderController;
use App\Http\Controllers\OutletController;
use App\Http\Controllers\PaymentController;
use App\Http\Controllers\PriceController;
use App\Http\Controllers\ReportController;
use App\Http\Controllers\ServiceController;
use Illuminate\Support\Facades\Route;

// The public pages: anyone may see them.
Route::get('/', HomeController::class)->name('home');
Route::get('services', [ServiceController::class, 'index'])
    ->name('services.index');
Route::get('services/{service}', [ServiceController::class, 'show'])
    ->name('services.show');
Route::get('outlets', [OutletController::class, 'index'])
    ->name('outlets.index');

// Signing in and out.
Route::middleware('guest')->group(function () {
    Route::get('login', [LoginController::class, 'create'])->name('login');
    Route::post('login', [LoginController::class, 'store'])
        ->middleware('throttle:5,1');           // five tries a minute
});
Route::post('logout', [LoginController::class, 'destroy'])
    ->middleware('auth')->name('logout');

// The staff pages: only for signed-in staff.
Route::middleware('auth')->group(function () {
    Route::get('counter', CounterController::class)->name('counter');
    Route::resource('customers', CustomerController::class);
    Route::resource('orders', OrderController::class)
        ->only(['index', 'create', 'store', 'show']);
    Route::get('reports/daily', [ReportController::class, 'daily'])
        ->name('reports.daily');
```

Kode 10.7 (Lanjutan) *routes/web.php: halaman publik, halaman masuk khusus tamu, halaman staf di belakang auth, dan daftar harga untuk pemilik.*

```php
    Route::middleware('order.open')->group(function () {
        Route::post('orders/{order}/advance',
            [OrderController::class, 'advance'])->name('orders.advance');
        Route::post('orders/{order}/payments',
            [PaymentController::class, 'store'])
            ->name('orders.payments.store');
    });

    // The price list: the owner only.
    Route::resource('prices', PriceController::class)
        ->parameters(['prices' => 'service'])
        ->except(['show', 'destroy'])
        ->middleware('can:manage-prices');
});
```

Middleware `guest` mengirim pengguna yang sudah masuk menjauh dari formulir masuk. `throttle:5,1` pada rute POST mengizinkan lima percobaan masuk per menit dari satu pengguna atau alamat, dan menjawab percobaan keenam dengan 429 Too Many Requests; itu mengubah penebakan kata sandi dari ribuan percobaan per menit menjadi lima. Keluar adalah rute POST,

tidak pernah GET: tautan yang mengeluarkan orang dapat dipicu oleh halaman web mana pun yang menyematkannya sebagai gambar.

Middleware `auth` perlu tahu ke mana tamu dikirim. **Kode 10.8** memberitahunya dengan `redirectGuestsTo('/login')`, dan juga menambahkan berkas rute API yang ditulis Subbab 10.5, dengan *middleware* pengukur waktu yang sama seperti halaman web.

Kode 10.8 `bootstrap/app.php`: rute API, pengukur waktu untuk kedua grup, dan halaman masuk bagi tamu.

```
<?php

use App\Http\Middleware\AddServerTiming;
use App\Http\Middleware\EnsureOrderIsOpen;
use Illuminate\Foundation\Application;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;

return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__.'/../routes/web.php',
        api: __DIR__.'/../routes/api.php',
        commands: __DIR__.'/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware): void {
        $middleware->web(append: [AddServerTiming::class]);
        $middleware->api(append: [AddServerTiming::class]);
        $middleware->alias([
            'order.open' => EnsureOrderIsOpen::class,
        ]);
        $middleware->redirectGuestsTo('/login');
    })
    ->withExceptions(function (Exceptions $exceptions): void {
        //
    })->create();
```

10.4.3 Gate untuk daftar harga

`can:manage-prices` bertanya kepada *gate* bernama `manage-prices`. *Gate* didefinisikan sekali, biasanya di metode `boot` pada `AppServiceProvider`, sebagai fungsi yang menerima pengguna yang sedang masuk dan mengembalikan benar atau salah [3]. **Kode 10.9** mendefinisikan *gate* kita: hanya pemilik yang boleh mengelola harga.

Kode 10.9 `app/Providers/AppServiceProvider.php`: satu *gate*, `manage-prices`, hanya untuk pemilik.

```php
<?php

namespace App\Providers;

use App\Models\User;
use Illuminate\Support\Facades\Gate;
use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
    public function register(): void
    {
        //
    }

    public function boot(): void
    {
        // Only the owner changes the price list.
        Gate::define('manage-prices', fn (User $user) => $user->isOwner());
    }
}
```

Gate yang sama dipakai di tiga tempat, dan di situlah kekuatannya. Di rute, `->middleware('can:manage-prices')` menjawab 403 kepada kasir yang mengetik URL daftar harga dengan tangan. Di Blade, `@can('manage-prices') ... @endcan` menyembunyikan tautan **Prices** dari kasir, dan `layouts/app.blade.php` bab ini melakukan persis hal itu, bersama tombol **Sign out** yang menampilkan nama pengguna di dalam blok `@auth`. Dan di kode PHP mana pun, `Gate::allows('manage-prices')` atau `$user->can('manage-prices')` mengajukan pertanyaan yang sama. Menyembunyikan tautan hanyalah kesopanan; *middleware* rute adalah kuncinya. Aturan yang berlaku untuk satu jenis model, seperti "hanya pemilik yang boleh menghapus pesanan yang belum dikerjakan", lebih cocok ditempatkan di kelas *policy*: kelas dengan satu metode per tindakan, yang ditemukan Laravel berdasarkan nama modelnya [3]. Latihan 10.4 menanyakan tempat masing-masing jenis aturan.

Tabel 10.1 mengumpulkan jawaban yang dapat diberikan halaman staf dan API bab ini ketika sesuatu ditolak, beserta bagian aplikasi yang memberikannya.

Tabel 10.1 Penolakan dalam aplikasi bab ini: kode status, artinya, dan bagian yang menjawab.

Status	Arti	Yang menjawab	Contoh
302 ke `/login`	tamu meminta halaman staf	*middleware* `auth` (peramban)	membuka `/counter` tanpa masuk
401 Unauthorized	tidak ada sesi atau token yang sah	`auth` / `auth:sanctum` (JSON)	memanggil API tanpa token
403 Forbidden	sudah masuk, tetapi tidak diizinkan	*middleware* `can:`, *gate*, atau *form request*	kasir membuka `/prices`
409 Conflict	pesanan sudah ditutup	`order.open` (Bab 9)	membayar pesanan yang sudah pergi lewat API
419 Page Expired	token CSRF formulir hilang atau kedaluwarsa	perlindungan pemalsuan permintaan	formulir yang dibiarkan terbuka semalaman
422 Unprocessable	masukan melanggar aturan	validasi, *form request*	membayar melebihi sisa tagihan
429 Too Many Requests	terlalu banyak percobaan	`throttle`	percobaan masuk keenam dalam semenit

10.5 API untuk Program

10.5.1 Memasang Sanctum

Proyek Laravel 13 yang baru tidak memiliki berkas rute API. Satu perintah menambahkan semua yang diperlukan API: `php artisan install:api` memasang Laravel Sanctum dengan Composer, menerbitkan migrasinya untuk tabel `personal_access_tokens`, membuat `routes/api.php`, dan mendaftarkannya [4]. Model `User` kita sudah memakai *trait* `HasApiTokens` yang diminta perintah itu, dan `bootstrap/app.php` kita sudah mendaftar berkas rute `api`. Setiap rute di berkas itu mendapat awalan `/api` dan grup *middleware* `api`, yang tidak memiliki sesi dan *cookie*: permintaan API bersifat *stateless* dan harus membuktikan dirinya pada setiap panggilan.

10.5.2 Menerbitkan dan mencabut token

Kode 10.10 adalah seluruh loket token: satu metode untuk menerbitkan token dan satu untuk mencabut token yang dipakai permintaan saat ini.

Kode 10.10 `app/Http/Controllers/Api/TokenController.php`: kata sandi masuk, token keluar; dan cara mengembalikannya.

```php
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Illuminate\Validation\ValidationException;

class TokenController extends Controller
{
    // POST /api/tokens: e-mail + password in, a token out (shown once).
    public function store(Request $request)
    {
        $data = $request->validate([
            'email' => ['required', 'email'],
            'password' => ['required', 'string'],
            'device' => ['required', 'string', 'max:60'],
        ]);

        $user = User::where('email', $data['email'])->first();
        if (! $user || ! Hash::check($data['password'], $user->password)) {
            throw ValidationException::withMessages([
                'email' => 'These credentials do not match our records.',
            ]);
        }

        return response()->json([
            'token' => $user->createToken($data['device'])->plainTextToken,
            'user' => ['name' => $user->name, 'role' => $user->role->value],
        ], 201);
    }

    // DELETE /api/tokens/current: sign this device out.
    public function destroy(Request $request)
    {
        $request->user()->currentAccessToken()->delete();

        return response()->noContent();
    }
}
```

Pemeriksaan kata sandinya sama dengan di formulir masuk, tetapi dilakukan secara eksplisit dengan `Hash::check`, karena tidak ada sesi yang terlibat. `createToken($data['device'])` membuat token acak baru, menyimpan *hash*-nya bersama nama perangkat yang dikirim aplikasi, misalnya “driver-phone-rina”, lalu mengembalikan objek yang `plainTextToken`-nya adalah satu-satunya salinan teks polos token itu yang akan pernah ada. API mengirimnya sekali, dengan status 201 Created; setelah itu bahkan server pun tidak dapat menampilkannya lagi, karena ia hanya memiliki *hash*-nya. `destroy` menghapus token yang dipakai untuk membuat permintaan saat ini: `currentAccessToken()` adalah token yang dicocokkan Sanctum. Aplikasi memanggilnya ketika sopir keluar; pemilik dapat menghapus token ponsel yang hilang dari basis data, atau menghapus semua token seorang pengguna dengan `$user->tokens()->delete()`.

10.5.3 Rute API

Kode 10.11 mendaftar API-nya. Rute token bersifat publik, dengan pembatas laju yang sama seperti formulir masuk; semua yang lain memerlukan `auth:sanctum`, dan dua rute yang mengubah pesanan dijaga oleh `order.open` Bab 9 persis seperti rute web.

Kode 10.11 `routes/api.php`: token, pelanggan, pesanan, dan pembayaran, di belakang `auth:sanctum`.

```php
<?php

use App\Http\Controllers\Api\CustomerController;
use App\Http\Controllers\Api\OrderController;
use App\Http\Controllers\Api\PaymentController;
use App\Http\Controllers\Api\TokenController;
use Illuminate\Support\Facades\Route;

// Every URL here starts with /api (Laravel adds the prefix).
Route::post('tokens', [TokenController::class, 'store'])
    ->middleware('throttle:5,1');

Route::middleware('auth:sanctum')->group(function () {
    Route::delete('tokens/current', [TokenController::class, 'destroy']);

    Route::get('customers', [CustomerController::class, 'index']);
    Route::post('customers', [CustomerController::class, 'store']);

    Route::get('orders', [OrderController::class, 'index']);
    Route::post('orders', [OrderController::class, 'store']);
    Route::get('orders/{order}', [OrderController::class, 'show']);

    Route::middleware('order.open')->group(function () {
        Route::patch('orders/{order}/status',
            [OrderController::class, 'updateStatus']);
        Route::post('orders/{order}/payments',
            [PaymentController::class, 'store']);
    });
});
```

Karena permintaan API mengirim `Accept: application/json`, setiap penolakan di **Tabel 10.1** tiba sebagai JSON: `{"message": "Unauthenticated."}` dengan 401, `{"message": "..."}` dengan 409 dari *middleware* kita, dan untuk galat validasi, badan 422 dengan `message` dan objek `errors` yang mendaftar pesan per isian. Dengan demikian aplikasi ponsel dapat menampilkan pesan yang tepat di bawah isian yang tepat, seperti yang dilakukan formulir Blade.

10.5.4 Satu cara membuat pesanan

Meja kasir web dan API sama-sama membuat pesanan. Jika setiap *controller* memiliki salinan sendiri dari "cari layanan, hitung total, simpan sebagai *queued*", kedua salinan itu akan menyimpang begitu salah satunya diubah. **Kode 10.12** memindahkan pekerjaan itu ke kelas *action*, yaitu kelas biasa dengan satu metode publik, yang diterima kedua *controller* dari *service container* Laravel.

Kode 10.12 `app/Actions/CreateOrder.php`: satu-satunya tempat masukan yang sudah diperiksa menjadi pesanan berharga.

```php
<?php

namespace App\Actions;

use App\Domain\PriceCalculator;
use App\Enums\OrderStatus;
use App\Models\Order;
use App\Models\Service;

// One place that turns checked input into a priced, saved order.
// The web page and the API both call it, so they can never disagree.
class CreateOrder
{
    public function handle(array $data): Order
    {
        $service = Service::findOrFail($data['service_id']);
        $pickup = (bool) ($data['pickup'] ?? false);

        return Order::create([
            ...$data,
            'pickup' => $pickup,
            'total' => PriceCalculator::total($service->per_kg,
                $service->price, (string) $data['quantity'], $pickup),
            'status' => OrderStatus::Queued,
            'received_at' => now(),
        ]);
    }
}
```

`OrderController` web di bab ini kini memanggil `$createOrder->handle($request->validated())` di metode `store`-nya alih-alih menghitung harga pesanan sendiri; bagian lain *controller* itu tidak berubah dari Bab 9. **Kode 10.13** adalah *controller* API untuk pesanan.

Kode 10.13 `app/Http/Controllers/Api/OrderController.php`: mendaftar, membuat, menampilkan, dan memajukan pesanan, menjawab dengan *resource*.

```php
<?php

namespace App\Http\Controllers\Api;

use App\Actions\CreateOrder;
use App\Enums\OrderStatus;
use App\Http\Controllers\Controller;
use App\Http\Requests\StoreOrderRequest;
use App\Http\Requests\UpdateOrderStatusRequest;
use App\Http\Resources\OrderResource;
use App\Models\Order;
use Illuminate\Http\Request;
use Illuminate\Validation\Rule;

class OrderController extends Controller
{
    // GET /api/orders?status=finished&page=2
    public function index(Request $request)
    {
        $request->validate([
            'status' => ['nullable', Rule::enum(OrderStatus::class)],
        ]);

        $orders = Order::with(['customer', 'service'])
            ->withSum('payments', 'amount')
            ->when($request->query('status'),
                fn ($q, $status) => $q->where('status', $status))
            ->orderByDesc('received_at')->orderByDesc('id')
            ->paginate(20)
            ->withQueryString();

        return OrderResource::collection($orders);
    }

    public function store(StoreOrderRequest $request, CreateOrder $createOrder)
    {
        $order = $createOrder->handle($request->validated());

        return (new OrderResource($order->load(['customer', 'service'])))
            ->response()->setStatusCode(201);
    }
```

Kode 10.13 (Lanjutan) *app/Http/Controllers/Api/OrderController.php: mendaftar, membuat, menampilkan, dan memajukan pesanan, menjawab dengan resource.*

```
    public function show(Order $order)
    {
        return new OrderResource($order->load(['customer', 'service',
            'payments' => fn ($q) => $q->orderBy('paid_at')]));
    }

    // PATCH /api/orders/5/status   {"status": "finished"}
    public function updateStatus(UpdateOrderStatusRequest $request,
        Order $order)
    {
        $next = OrderStatus::from($request->validated('status'));
        $order->status = $next;
        if ($next === OrderStatus::Finished) {
            $order->finished_at = now();
        } else {
            $order->departed_at = now();
        }
        $order->save();

        return new OrderResource($order->load(['customer', 'service']));
    }
}
```

`index` memvalidasi satu-satunya parameter kuerinya, lalu membangun kueri yang sejenis dengan daftar web Bab 8: pelanggan dan layanan dimuat lebih awal, jumlah pembayaran ditambahkan oleh `withSum`, penyaring status yang opsional, dua puluh per halaman. `->withQueryString()` mempertahankan `?status=finished` di tautan ke halaman berikutnya. `OrderResource::collection($orders)` mengubah halaman itu menjadi JSON dengan tiga bagian: `data`, yaitu pesanannya; `links`, yaitu URL halaman pertama, sebelumnya, berikutnya, dan terakhir; dan `meta`, dengan angka-angka seperti `current_page`, `per_page`, dan `total` [5]. `store` memakai ulang `StoreOrderRequest` Bab 9, sehingga web dan API menerima masukan yang persis sama, dan menjawab 201 Created. `show` juga memuat pembayaran, diurutkan menurut waktu.

`updateStatus` adalah cara API memajukan pesanan. Peramban menekan tombol yang berarti "langkah berikutnya, apa pun itu"; program sebaiknya menyebutkan langkah berikutnya yang ia yakini, `{"status": "finished"}`, agar dua sopir yang menekan pada saat bersamaan tidak memajukan pesanan dua langkah. **Kode 10.14** memeriksa permintaan itu dengan `OrderWorkflow` Bab 8.

Kode 10.14 `app/Http/Requests/UpdateOrderStatusRequest.php`: status yang diminta harus status berikutnya yang diizinkan alur kerja.

```php
<?php

namespace App\Http\Requests;

use App\Domain\OrderWorkflow;
use App\Enums\OrderStatus;
use DomainException;
use Illuminate\Foundation\Http\FormRequest;
use Illuminate\Validation\Rule;
use Illuminate\Validation\Validator;

class UpdateOrderStatusRequest extends FormRequest
{
    public function authorize(): bool
    {
        return true;
    }

    public function rules(): array
    {
        return ['status' => ['required', Rule::enum(OrderStatus::class)]];
    }

    // The workflow of Chapter 8 decides which step is allowed.
    public function after(): array
    {
        return [
            function (Validator $validator) {
                if ($validator->errors()->isNotEmpty()) {
                    return;
                }
                $order = $this->route('order')->load('payments');
                try {
                    $next = OrderWorkflow::advance($order->status,
                        $order->balance());
                } catch (DomainException $e) {
                    $validator->errors()->add('status', $e->getMessage());
                    return;
                }
                if ($this->input('status') !== $next->value) {
                    $validator->errors()->add('status', "An order that is "
                        . "{$order->status->value} can only become "
                        . "{$next->value}.");
                }
            },
        ];
```

Kode 10.14 (Lanjutan) *app/Http/Requests/UpdateOrderStatusRequest.php: status yang diminta harus status berikutnya yang diizinkan alur kerja.*

```
    }
}
```

Aturan `Rule::enum(OrderStatus::class)` menolak kata-kata yang sama sekali bukan status. *Hook* `after` kemudian menanyakan status berikutnya kepada alur kerja, berdasarkan sisa tagihan pesanan. Jika alur kerja menolak, karena cucian akan pergi sebelum lunas, pesannya sendiri menjadi galat: “Collect Rp 34.000 before handing over the laundry.” Jika alur kerja mengizinkan satu langkah tetapi bukan langkah yang diminta, galatnya menyebutkan langkah mana yang diizinkan. Pesanan yang sudah pergi tidak pernah sampai ke *request* ini, karena `order.open` menghentikannya lebih dahulu dengan 409.

API pembayaran (**Kode 10.15**) memakai ulang `StorePaymentRequest` Bab 9, dengan pemeriksaannya terhadap sisa tagihan, dan menjawab dengan seluruh pesanan beserta pembayarannya, sehingga aplikasi dapat langsung menampilkan sisa tagihan yang baru.

Kode 10.15 `app/Http/Controllers/Api/PaymentController.php`: mencatat pembayaran, menjawab dengan pesanan yang sudah diperbarui.

```php
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use App\Http\Requests\StorePaymentRequest;
use App\Http\Resources\OrderResource;
use App\Models\Order;

class PaymentController extends Controller
{
    // POST /api/orders/5/payments   {"amount": 34000, "method": "qris"}
    public function store(StorePaymentRequest $request, Order $order)
    {
        $order->payments()->create([...$request->validated(),
            'paid_at' => now()]);

        return (new OrderResource($order->load(['customer', 'service',
            'payments' => fn ($q) => $q->orderBy('paid_at')])))
            ->response()->setStatusCode(201);
    }
}
```

Para sopir juga bertemu pelanggan baru di depan pintu. **Kode 10.16** memeriksa pelanggan baru: nomor telepon lebih dahulu disederhanakan menjadi angka-angkanya saja, sehingga “0812-3456-7890” dan “0812 3456 7890” dianggap nomor yang sama, lalu nomor itu harus tampak seperti nomor seluler Indonesia dan belum dimiliki pelanggan lain. `CustomerController` API, di folder bab ini, memiliki `index` yang mencari pelanggan berdasarkan telepon dan `store` yang memakai *request* ini serta menjawab 201.

Kode 10.16 `app/Http/Requests/StoreCustomerRequest.php`: pelanggan baru, dengan telepon yang disederhanakan menjadi angka dan diperiksa agar tidak ganda.

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class StoreCustomerRequest extends FormRequest
{
    public function authorize(): bool
    {
        return true;
    }

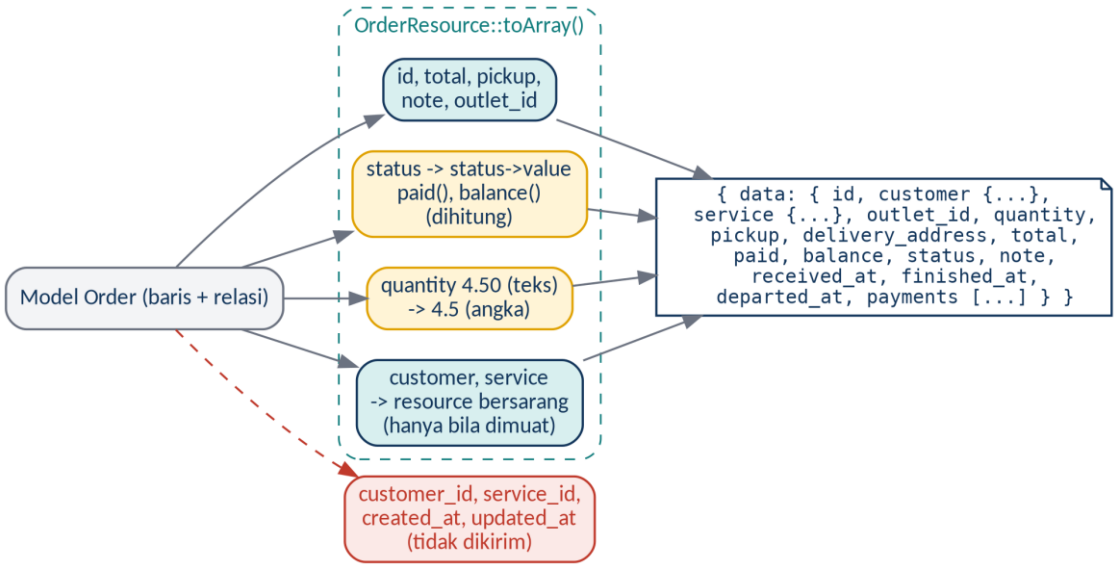
    // "0812-3456-7890" and "0812 3456 7890" are the same number.
    protected function prepareForValidation(): void
    {
        $this->merge([
            'phone' => preg_replace('/[^0-9]/', '', (string) $this->phone),
        ]);
    }

    public function rules(): array
    {
        return [
            'name' => ['required', 'string', 'min:2', 'max:80'],
            'phone' => ['required', 'regex:/^08[0-9]{8,11}$/',
                'unique:customers,phone'],
            'address' => ['nullable', 'string', 'max:200'],
        ];
    }

    public function messages(): array
    {
        return ['phone.regex' => 'Write the phone as 08..., 10 to 13 digits.'];
    }
}
```

10.6 API Resource

Gambar 10.2 memperlihatkan apa yang dilakukan `OrderResource` terhadap sebuah pesanan: sebagian isian diteruskan tanpa perubahan, sebagian diubah bentuknya, sebagian dihitung, relasi menjadi objek bersarang bila sudah dimuat, dan sisanya tidak pernah meninggalkan server.



Gambar 10.2 Apa yang dilakukan `OrderResource` terhadap sebuah pesanan: isian diteruskan, diubah bentuknya, atau dihitung, relasi disarangkan bila dimuat, dan kolom internal ditinggalkan.

Kode 10.17 adalah kelasnya, dibuat dengan `php artisan make:resource OrderResource`.

Kode 10.17 `app/Http/Resources/OrderResource.php`: bentuk JSON sebuah pesanan.

```php
<?php

namespace App\Http\Resources;

use Illuminate\Http\Request;
use Illuminate\Http\Resources\Json\JsonResource;

class OrderResource extends JsonResource
{
    public function toArray(Request $request): array
    {
        return [
            'id' => $this->id,
            'customer' => new CustomerResource($this->whenLoaded('customer')),
            'service' => new ServiceResource($this->whenLoaded('service')),
            'outlet_id' => $this->outlet_id,
            'quantity' => (float) $this->quantity,
            'pickup' => $this->pickup,
            'delivery_address' => $this->delivery_address,
            'total' => $this->total,
            'paid' => $this->paid(),
            'balance' => $this->balance(),
            'status' => $this->status->value,
            'note' => $this->note,
            'received_at' => $this->received_at,
            'finished_at' => $this->finished_at,
            'departed_at' => $this->departed_at,
            'payments' => PaymentResource::collection(
                $this->whenLoaded('payments')),
        ];
    }
}
```

Di dalam `toArray`, `$this` mewakili pesanan itu, sehingga `$this->total` membaca total pesanan [5]. `'quantity' => (float) $this->quantity` mengubah teks `"4.50"` yang dihasilkan *cast* `decimal:2` menjadi angka `4.5`, yang memang diharapkan program dari sebuah jumlah. `paid()` dan `balance()` berasal dari model; `Order::paid()` bab ini memakai `payments_sum_amount` yang

ditambahkan `withSum` bila kueri menyediakannya, dan pembayaran yang sudah dimuat bila tidak, sehingga daftar dua puluh pesanan hanya memerlukan satu kueri tambahan, bukan dua puluh. `$this->status->value` mengirim kata `"finished"`, bukan objek *enum*-nya. `whenLoaded('customer')` hanya menyertakan pelanggan bila *controller* sudah memuatnya; bila tidak, kuncinya ditinggalkan sama sekali, dan tidak ada kueri tersembunyi yang berjalan. `created_at`, `updated_at`, dan *foreign key* mentah tidak didaftar, sehingga tidak dikirim.

Ketiga *resource* kecil sama sederhananya. `CustomerResource` mengirim `id`, `name`, `phone`, dan `address`. `ServiceResource` (**Kode 10.18**) mengirim kode, nama dan satuan yang diturunkan model `Service` Bab 7 dari deskripsinya, serta harganya; deskripsinya sendiri tetap di dalam.

Kode 10.18 `app/Http/Resources/ServiceResource.php`: satu baris daftar harga, sesuai kebutuhan ponsel.

```php
<?php

namespace App\Http\Resources;

use Illuminate\Http\Request;
use Illuminate\Http\Resources\Json\JsonResource;

class ServiceResource extends JsonResource
{
    public function toArray(Request $request): array
    {
        return [
            'id' => $this->id,
            'name' => $this->name,            // worked out from description
            'unit' => $this->unit,            // kg, item or pair
            'price' => $this->price,
        ];
    }
}
```

Resource tunggal dibungkus dalam kunci `data`: ponsel menerima `{"data": {"id": 12, "customer": {...}, ...}}`. Pembungkus ini menyisakan ruang untuk kunci tingkat atas lainnya, seperti `links` dan `meta` sebuah halaman, dan bentuknya sama untuk pesanan tunggal maupun daftar, sehingga kode aplikasinya lebih sederhana. Waktu-waktunya, `received_at` dan yang lain, dikirim di bab ini sesuai cara bawaan Laravel menyerialkannya, dan Bab 12 memeriksanya dengan teliti ketika API yang sama dibangun untuk kedua kalinya dengan Spring Boot.

10.7 Pengujian

10.7.1 Uji unit untuk aturan harga

Kode 10.19 memeriksa `PriceCalculator` saja. Kelas uji ini menurunkan `TestCase` milik PHPUnit sendiri, bukan milik Laravel, karena tidak memerlukan apa pun dari kerangka kerja: tanpa basis data, tanpa konfigurasi, tanpa HTTP.

Kode 10.19 `tests/Unit/PriceCalculatorTest.php`: enam kasus harga, satu metode uji.

```php
<?php

namespace Tests\Unit;

use App\Domain\PriceCalculator;
use PHPUnit\Framework\Attributes\DataProvider;
use PHPUnit\Framework\TestCase;

// No framework, no database: the rule of Chapter 3, checked on its own.
class PriceCalculatorTest extends TestCase
{
    public static function cases(): array
    {
        return [
            'per kg' => [true, 8000, '4.5', false, 36000],
            'minimum 3 kg' => [true, 8000, '2.5', false, 24000],
            'minimum 3 kg, pickup' => [true, 8000, '2.5', true, 34000],
            'items counted whole' => [false, 35000, '1.5', false, 70000],
            'free pickup from 100k' => [false, 65000, '2', true, 130000],
            'two decimals' => [true, 15000, '3.35', false, 50250],
        ];
    }

    #[DataProvider('cases')]
    public function test_total(bool $perKg, int $price, string $quantity,
        bool $pickup, int $expected): void
    {
        $this->assertSame($expected,
            PriceCalculator::total($perKg, $price, $quantity, $pickup));
    }
}
```

Atribut `#[DataProvider('cases')]` menjalankan `test_total` sekali untuk setiap baris `cases()`, dan kegagalan menyebutkan nama barisnya, misalnya “minimum 3 kg, pickup”, sehingga Anda langsung tahu aturan mana yang rusak. Keenam baris itu mencakup aturan Bab 3: harga per kilogram, minimum tiga kilogram, biaya jemput-antar, satuan utuh, jemput-antar gratis mulai Rp 100.000, dan berat dengan dua angka desimal. Total yang diharapkan bukan tebakan: semuanya dihitung dengan `PriceCalculator` Bab 8 dalam PHP biasa, dan sesuai dengan `PriceCalculator` ASP.NET Core di Bab 5, seperti yang dibuktikan pemeriksaan Bab 8.

10.7.2 Uji fitur untuk proses masuk

Kode 10.20 memeriksa proses masuk, hak akses, dan token melalui permintaan HTTP pura-pura.

Kode 10.20 `tests/Feature/AuthTest.php`: tamu, proses masuk, kata sandi yang salah, daftar harga milik pemilik, dan token API.

```php
<?php

namespace Tests\Feature;

use App\Models\User;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Tests\TestCase;

class AuthTest extends TestCase
{
    use RefreshDatabase;

    public function test_guests_are_sent_to_the_sign_in_page(): void
    {
        $this->get('/counter')->assertRedirect('/login');
    }

    public function test_the_public_pages_stay_public(): void
    {
        $this->get('/outlets')->assertOk();
    }

    public function test_staff_sign_in_and_reach_the_counter(): void
    {
        $user = User::factory()->create();

        $this->post('/login', ['email' => $user->email,
            'password' => 'password'])->assertRedirect('/counter');

        $this->assertAuthenticatedAs($user);
    }

    public function test_a_wrong_password_is_refused(): void
    {
        $user = User::factory()->create();

        $this->from('/login')->post('/login', ['email' => $user->email,
            'password' => 'wrong'])
            ->assertRedirect('/login')->assertSessionHasErrors('email');
        $this->assertGuest();
    }
```

Kode 10.20 (Lanjutan) *tests/Feature/AuthTest.php: tamu, proses masuk, kata sandi yang salah, daftar harga milik pemilik, dan token API.*

```
    public function test_only_the_owner_sees_the_price_list(): void
    {
        $this->actingAs(User::factory()->create())
            ->get('/prices')->assertForbidden();
        $this->actingAs(User::factory()->owner()->create())
            ->get('/prices')->assertOk();
    }

    public function test_a_token_opens_the_api_and_can_be_revoked(): void
    {
        $user = User::factory()->create();

        $token = $this->postJson('/api/tokens', ['email' => $user->email,
            'password' => 'password', 'device' => 'counter-tablet'])
            ->assertCreated()->json('token');

        $this->withToken($token)->getJson('/api/orders')->assertOk();
        $this->withToken($token)->deleteJson('/api/tokens/current')
            ->assertNoContent();
        $this->assertDatabaseCount('personal_access_tokens', 0);
    }

    public function test_the_api_refuses_requests_without_a_token(): void
    {
        $this->getJson('/api/orders')->assertUnauthorized();
    }
}
```

Setiap metode adalah satu cerita pendek dengan akhir yang diharapkan. `$this->get('/counter')` mengirim permintaan GET melalui seluruh aplikasi tanpa jaringan; `assertRedirect('/login')` memeriksa status dan *header* `Location`. `actingAs($user)` memasukkan seorang pengguna untuk permintaan berikutnya tanpa melalui formulir, sehingga uji tentang hak akses tetap pendek. `from('/login')` menetapkan halaman asal permintaan pura-pura itu, agar validasi yang gagal mengalihkan kembali ke sana. `postJson` dan `getJson` mengirim JSON dan meminta JSON, seperti ponsel; `withToken($token)` menambahkan *header* `Authorization: Bearer`. Baris-baris terakhir `test_a_token_opens_the_api_and_can_be_revoked` memeriksa basis data secara langsung: setelah dicabut, tidak ada token yang tersisa.

`RefreshDatabase` memberi setiap uji basis data kosong dengan semua migrasi diterapkan [9]. `phpunit.xml` pada proyek Laravel baru menjalankan uji di basis data SQLite dalam memori, sehingga uji tidak pernah menyentuh basis data yang Anda pakai untuk mencoba aplikasi, dan berjalan cepat.

10.7.3 Uji fitur untuk API

Kode 10.21 memeriksa API sebagaimana aplikasi para sopir akan memakainya.

Kode 10.21 `tests/Feature/OrderApiTest.php`: harga, validasi, alur kerja, pesanan tertutup, pembayaran, dan penyaring, melalui API.

```php
<?php

namespace Tests\Feature;

use App\Models\Customer;
use App\Models\Order;
use App\Models\User;
use Database\Seeders\PriceListSeeder;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Laravel\Sanctum\Sanctum;
use Tests\TestCase;

class OrderApiTest extends TestCase
{
    use RefreshDatabase;

    protected function setUp(): void
    {
        parent::setUp();
        $this->seed(PriceListSeeder::class);        // services and outlets
        Sanctum::actingAs(User::factory()->create());
    }

    private function newOrder(): array
    {
        return $this->postJson('/api/orders', [
            'customer_id' => Customer::factory()->create()->id,
            'outlet_id' => 'O01', 'service_id' => 'S0001',
            'quantity' => '2,5', 'pickup' => true,
            'delivery_address' => 'Jl. Kenanga 5, Surabaya',
        ])->assertCreated()->json('data');
    }

    public function test_an_order_is_priced_by_the_rule(): void
    {
        $order = $this->newOrder();

        $this->assertSame(34000, $order['total']);   // 3 kg x 8,000 + 10,000
        $this->assertSame(34000, $order['balance']);
        $this->assertSame('queued', $order['status']);
    }
```

Kode 10.21 (Lanjutan) *tests/Feature/OrderApiTest.php: harga, validasi, alur kerja, pesanan tertutup, pembayaran, dan penyaring, melalui API.*

```
    public function test_bad_input_gets_422_with_every_error(): void
    {
        $this->postJson('/api/orders', ['quantity' => 0, 'pickup' => true])
            ->assertUnprocessable()
            ->assertJsonValidationErrors(['customer_id', 'outlet_id',
                'service_id', 'quantity', 'delivery_address']);
    }

    public function test_the_steps_follow_the_workflow(): void
    {
        $url = '/api/orders/' . $this->newOrder()['id'];

        $this->patchJson("$url/status", ['status' => 'departed'])
            ->assertUnprocessable();                    // skips a step
        $this->patchJson("$url/status", ['status' => 'finished'])
            ->assertOk()->assertJsonPath('data.status', 'finished');
        $this->patchJson("$url/status", ['status' => 'departed'])
            ->assertUnprocessable()                     // not paid yet
            ->assertJsonValidationErrors('status');

        $this->postJson("$url/payments", ['amount' => 34000,
            'method' => 'qris'])->assertCreated()
            ->assertJsonPath('data.balance', 0);
        $this->patchJson("$url/status", ['status' => 'departed'])
            ->assertOk();
    }

    public function test_a_departed_order_is_closed(): void
    {
        $order = Order::factory()->departed()->create();

        $this->postJson("/api/orders/{$order->id}/payments",
            ['amount' => 1000, 'method' => 'cash'])
            ->assertStatus(409);
    }

    public function test_a_payment_cannot_exceed_the_balance(): void
    {
        $url = '/api/orders/' . $this->newOrder()['id'];

        $this->postJson("$url/payments", ['amount' => 40000,
            'method' => 'cash'])
            ->assertUnprocessable()->assertJsonValidationErrors('amount');
    }
```

Kode 10.21 (Lanjutan) *tests/Feature/OrderApiTest.php: harga, validasi, alur kerja, pesanan tertutup, pembayaran, dan penyaring, melalui API.*

```
    public function test_the_list_can_be_filtered_by_status(): void
    {
        Order::factory()->count(3)->create();
        Order::factory()->finished()->count(2)->create();

        $this->getJson('/api/orders?status=finished')
            ->assertOk()
            ->assertJsonCount(2, 'data')
            ->assertJsonPath('meta.total', 2);
    }
}
```

`setUp` berjalan sebelum setiap uji: ia mengisi daftar harga dan gerai yang sungguhan, yang diperlukan pesanan, dan `Sanctum::actingAs` memasukkan seorang kasir ke API tanpa membuat

token sungguhan [4]. Metode pembantu `newOrder` membuat pesanan jemput-antar Budi dari Subbab 9.8 melalui API, dengan koma Indonesia pada "2,5", dan mengembalikan bagian `data` dari jawabannya. Uji-uji kemudian terbaca seperti aturan toko. Pesanan 2,5 kg dengan jemput-antar berharga Rp 34.000. Permintaan yang semuanya salah dijawab 422 dengan galat untuk setiap isian yang memerlukannya. Pesanan dapat berpindah dari *queued* ke *finished*, tidak dapat melompat ke *departed*, tidak dapat pergi sebelum lunas, dan pergi setelah dibayar. Pesanan yang sudah pergi menolak pembayaran dengan 409. Pembayaran di atas sisa tagihan ditolak. Dan penyaring status menghitung dengan benar, yang dikonfirmasi oleh `meta.total`.

10.7.4 Menjalankan uji

```
php artisan test
```

menjalankan setiap uji di `tests/Unit` dan `tests/Feature` dan mencetak satu baris untuk setiap uji beserta ringkasannya. `php artisan test --filter=OrderApiTest` hanya menjalankan satu kelas, berguna ketika Anda sedang mengerjakannya. Ketika sebuah uji gagal, laporannya menyebutkan nama metode uji, baris *assertion* yang gagal, serta apa yang diharapkan dibandingkan dengan apa yang diterima, misalnya status 422 yang diharapkan dibandingkan dengan 201 yang diterima. Bacalah baris itu lebih dahulu: baris itu memberi tahu apakah aplikasi melakukan sesuatu yang baru (aturannya dilanggar) atau apakah uji itu sendiri keliru menggambarkan aturannya. Untuk *assertion* JSON yang gagal, `->dump()` yang ditempatkan tepat sebelum *assertion* mencetak seluruh badan respons, sehingga Anda dapat melihat apa yang sebenarnya dikirim aplikasi. Dan ketika Anda memperbaiki sebuah kekeliruan, jalankan seluruh rangkaian uji, bukan hanya uji yang sedang Anda perhatikan: gunanya uji otomatis justru karena uji itu juga mengawasi aturan-aturan yang tidak sedang Anda pikirkan. Kebiasaan yang baik, yang kini diikuti Dimas, adalah menjalankan `php artisan test` sebelum setiap perubahan disalin ke komputer toko, dan tidak menyalin apa pun selama masih ada satu uji yang merah. Proyek Laravel juga dapat memakai Pest, perangkat pengujian dengan sintaksis yang lebih ringkas yang berjalan di atas PHPUnit dan menjalankan kelas uji PHPUnit seperti milik kita tanpa perubahan [12]. `README.md` bab ini mendaftar jumlah uji di setiap berkas. Uji-uji ini tidak dapat dijalankan di tempat buku ini ditulis, karena Laravel tidak dapat dipasang di sana; semuanya ditulis mengikuti dokumentasi pengujian Laravel 13, dan total harga yang diharapkan sudah dihitung dan diperiksa dalam PHP biasa.

10.8 Menjalankannya

Dengan proyek Bab 9 sudah tersedia, jalankan `php artisan install:api` sekali, salin berkas-berkas bab ini ke atas proyek, jalankan `php artisan migrate:fresh --seed`, dan jalankan server di porta 4100, seperti yang dijelaskan `README.md`. Lalu cobalah aplikasi sebagai setiap tokoh dalam cerita.

1. Buka http://localhost:4100/counter. Anda tiba di halaman masuk. Masuklah sebagai `cashier@laundry.test` dengan kata sandi `password`: Anda sampai di meja kasir, dan menunya tidak memiliki tautan **Prices**. Ketik `/prices` di bilah alamat: jawabannya 403.
2. Keluar, lalu masuk sebagai `owner@laundry.test`: tautan **Prices** ada dan berfungsi.
3. Ketik kata sandi yang salah enam kali dalam semenit: percobaan keenam ditolak dengan 429 bahkan sebelum kata sandinya diperiksa.
4. Di terminal, mintalah token seperti yang akan dilakukan sopir, dengan perintah `curl` pertama di `README.md`. Jawabannya adalah objek JSON dengan `token` serta nama dan peran pengguna.

Pakailah token itu dengan perintah kedua untuk mendaftar pesanan, dan tambahkan `?status=finished` untuk melihat pesanan yang sudah selesai saja.

5. Cobalah daftar yang sama tanpa *header* `Authorization`: jawabannya 401 dengan `{"message":"Unauthenticated."}`.

Di Windows, `curl` pada PowerShell adalah program yang berbeda, sehingga ketiklah `curl.exe` seperti yang disebutkan `README.md`; di macOS dan Linux `curl` sudah merupakan program yang tepat. Ekstensi REST Client di VS Code adalah alternatif yang ramah yang menyimpan permintaan-permintaan di dalam berkas.

10.9 Membuatnya Lebih Baik

Kemampuan token. Sebuah token dapat membawa *abilities* (kemampuan), yaitu daftar kata yang dipilih ketika token dibuat: `createToken('driver-phone', ['orders:read', 'payments:write'])`. Rute kemudian dapat menuntutnya dengan *middleware* `abilities` dan `ability` milik Sanctum, dan kode dapat bertanya `$request->user()->tokenCan('payments:write')` [4]. Ponsel sopir yang hanya perlu membaca pesanan dan mencatat pembayaran tidak seharusnya membawa token yang juga dapat membuat pelanggan.

Token yang kedaluwarsa. Secara bawaan token Sanctum tidak kedaluwarsa. Pengaturan `expiration` di `config/sanctum.php` memberi setiap token masa berlaku dalam menit, dan `php artisan sanctum:prune-expired` menghapus token yang sudah mati [4]. Masa berlaku beberapa minggu, dengan kewajiban masuk ulang sesudahnya, adalah bawaan yang wajar untuk ponsel para sopir.

Batas laju untuk API. API secara keseluruhan juga layak dibatasi. `throttle:api` memakai pembatas bernama yang dapat mengizinkan, misalnya, 60 permintaan per menit per pengguna, dan melindungi komputer kasir dari aplikasi bermasalah yang menanyakan hal yang sama berulang-ulang.

Policy ketika aturan bertambah. Satu *gate* cukup untuk satu aturan. Ketika aturannya menjadi "kasir boleh menyunting pelanggan yang ia daftarkan hari ini, pemilik boleh menyunting pelanggan mana pun", tulislah `OrderPolicy` atau `CustomerPolicy` dengan satu metode per tindakan; `can:update,customer` di rute lalu meneruskan model yang terikat kepada *policy* itu [3].

JSON:API. Laravel 13 menambahkan *resource* yang mengikuti spesifikasi JSON:API, dengan nama baku untuk relasi, rekaman yang disertakan, dan *sparse fieldset* [1]. Untuk API publik yang dipakai banyak klien berbeda, standar yang sudah diterbitkan menghemat upaya menulis dan menjelaskan konvensi sendiri; untuk aplikasi ponsel milik toko sendiri, *resource* biasa bab ini lebih sederhana.

10.10 Memakainya Sungguhan

- **Satu akun untuk satu orang, tidak pernah dipakai bersama.** Akun "kasir" bersama membuat setiap baris log tidak berguna: pertanyaan "siapa yang mengubah harga ini?" tidak memiliki jawaban. Buatlah akun untuk setiap kasir dan sopir, dan hapuslah pada hari mereka berhenti bekerja.
- **Kata sandi yang hanya diketahui pemiliknya.** Kata sandi "password" pada akun demo tidak boleh pernah ada di komputer toko. Biarkan setiap orang memilih kata sandi yang panjang, dan jangan pernah memintanya lewat WhatsApp. Laravel hanya menyimpan *hash* bcrypt, tetapi kata sandi yang lemah tidak perlu dipecahkan, cukup ditebak [11].

- **HTTPS di mana pun selain laptop Anda sendiri.** *Cookie* sesi dan token *bearer* sama berharganya dengan kata sandi selama dalam perjalanan. Di server sungguhan, sajikan aplikasi hanya melalui HTTPS dan tetapkan `APP_DEBUG=false`, agar halaman galat tidak menampilkan kode, pengaturan, atau data kepada orang asing.
- **Satu token untuk satu perangkat, dicabut ketika perangkatnya hilang.** Ketika ponsel hilang atau seorang sopir berhenti, hapuslah tokennya saat itu juga.
- **Data pribadi dilindungi undang-undang.** Nama, nomor telepon, dan alamat pelanggan adalah data pribadi menurut UU PDP 27/2022. Hanya mengirim isian yang diperlukan di setiap *resource*, menjaga akun tetap pribadi, dan segera mencabut akses adalah bagian dari memperlakukan data itu dengan cermat.
- **Uji untuk setiap aturan yang pernah merugikan uang.** Harga yang salah di Subbab 10.1 akan tertangkap oleh uji di **Kode 10.20** yang melarang kasir membuka daftar harga. Setiap kali sebuah kekeliruan sampai ke toko, tulislah uji yang seharusnya menangkapnya sebelum Anda memperbaikinya.

10.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Memeriksa peran hanya di *view*.** Menyembunyikan tautan **Prices** tidak melindungi `/prices`; *middleware* rute, *gate*, atau *policy* yang harus menolak permintaannya.
- **Memberi tahu orang asing bagian mana yang salah.** “Tidak ada pengguna dengan surel ini” atau “Kata sandi salah” membantu orang menebak akun. Berikan satu pesan untuk keduanya.
- **Lupa `session()->regenerate()` setelah masuk.** Pintu terbuka bagi *session fixation*. Selalu perbarui sesi setelah berhasil masuk, dan batalkan sesi ketika keluar.
- **Keluar dengan GET.** Sebuah tautan atau gambar lalu dapat mengeluarkan orang dari halaman mana pun. Pakailah formulir POST dengan `@csrf`.
- **Mengembalikan model alih-alih *resource* dari API.** JSON-nya lalu mengikuti basis data dan membocorkan kolom. Kembalikan *resource*, dan daftarlah setiap isian dengan sengaja.
- **Memakai `whenLoaded` tanpa memuat.** *Resource* yang memakai `whenLoaded('customer')` tidak mengirim pelanggan bila *controller* lupa `with('customer')`; *resource* yang membaca `$this->customer->name` secara langsung menjalankan satu kueri untuk setiap pesanan. Muatlah di *controller* apa yang diperlukan *resource*.
- **Menyimpan atau mencatat token polos.** `plainTextToken` harus ditampilkan sekali kepada perangkatnya dan tidak pernah ditulis ke log, kolom basis data, atau laporan galat.
- **Uji yang saling bergantung atau bergantung pada data Anda.** Uji yang menganggap “ada 3 pesanan” rusak ketika *seeder* demo berubah. Pakailah `RefreshDatabase` dan buatlah apa yang diperlukan setiap uji di dalam uji itu sendiri.
- **Hanya menguji jalur yang mulus.** Sebagian besar uji di bab ini memeriksa penolakan: 401, 403, 409, 422. Itulah aturan yang melindungi toko, dan aturan itulah yang rusak diam-diam.

10.12 Ringkasan Bab

- Autentikasi menjawab “siapa Anda?” dan otorisasi menjawab “bolehkah Anda melakukan ini?”; kegagalannya masing-masing adalah 401 dan 403, dan tamu yang memakai peramban dialihkan ke halaman masuk.
- Pengguna memiliki peran, pemilik atau kasir, yang disimpan sebagai *backed enum*; kata sandi hanya disimpan sebagai *hash*, berkat *cast* `hashed`.

- `LoginController` yang ditulis dengan tangan memasukkan staf dengan `Auth::attempt`, memperbarui sesi, dan mengeluarkan mereka dengan membatalkan sesi. Rute dikelompokkan menjadi publik, khusus tamu, `auth`, dan khusus pemilik dengan `can:manage-prices`, sebuah *gate* yang didefinisikan di `AppServiceProvider`. `throttle` membatasi percobaan masuk.
- Laravel Sanctum menerbitkan token API yang disimpan sebagai *hash*, dikirim sebagai token *bearer*, dan dicabut per perangkat. Rute API berada di `routes/api.php` di belakang `auth:sanctum` dan memakai ulang *form request* dan *middleware* `order.open` Bab 9.
- Kelas *action* `CreateOrder` membuat meja kasir web dan API membuat pesanan dengan satu cara saja. `UpdateOrderStatusRequest` memungkinkan program memajukan pesanan ke status yang disebutkan, diperiksa oleh alur kerja Bab 8.
- *API resource* menentukan JSON-nya: isian mana, dengan nama apa, diubah bentuknya atau dihitung, relasi disarangkan hanya bila dimuat, dan daftar yang dibagi per halaman dengan `links` dan `meta`.
- Uji unit memeriksa kelas biasa seperti `PriceCalculator`; uji fitur mengirim permintaan melalui seluruh aplikasi dengan `RefreshDatabase`, `actingAs`, dan `Sanctum::actingAs`, lalu memeriksa kode status, jalur JSON, dan basis data. `php artisan test` menjalankan semuanya dalam hitungan detik.

10.13 Soal Latihan

1. Untuk setiap permintaan berikut, sebutkan kode status jawabannya dan bagian aplikasi yang memberikannya: (a) tamu membuka `/counter` di peramban; (b) program memanggil `GET /api/orders` tanpa token; (c) kasir membuka `/prices`; (d) percobaan masuk yang salah untuk keenam kalinya dalam satu menit; (e) kasir yang sudah masuk mengirim formulir pembayaran untuk pesanan yang baru saja ditandai sudah diambil oleh kasir lain.
2. Tulislah kelas uji fitur yang membuktikan bahwa kasir tidak dapat membuka `/prices/create` maupun mengirim formulirnya ke `POST /prices`, bahwa pemilik dapat membuka formulir itu, dan bahwa tamu diminta masuk lebih dahulu.
3. Ubahlah penerbitan token sehingga token pemilik boleh melakukan segalanya sedangkan token kasir hanya membawa kemampuan `orders:write` dan `payments:write`, dan lindungi rute pelanggan pada API dengan kemampuan baru `customers:write`. Baris mana yang berubah, dan apa yang diterima ponsel kasir ketika mencoba membuat pelanggan?
4. Aturan baru: “kasir boleh membetulkan catatan pesanan hanya pada hari pesanan itu diterima; pemilik kapan saja”. Apakah aturan ini sebaiknya berupa *gate*, *policy*, atau *form request*? Buatlah sketsa tempat kodenya dan cara rute memakainya.
5. Jelaskan mengapa `TokenController::store` hanya dapat menampilkan token sekali, apa tepatnya yang disimpan tabel `personal_access_tokens`, dan apa yang harus dilakukan pemilik ketika ponsel seorang sopir dicuri.
6. Panggillah `GET /api/orders/{id}` untuk pesanan Sari dan perhatikan `received_at`. Laravel mengirimnya sesuai cara bawaannya menyerialkan tanggal. Apa arti huruf `Z` dan angka-angka setelah detik, dan mengapa jamnya berbeda dari jam di dinding meja kasir di Surabaya?

10.14 Penyelesaian Soal Latihan

1. (a) Pengalihan 302 ke `/login`, dari *middleware* `auth`, dengan tujuan yang ditetapkan oleh `redirectGuestsTo`. (b) 401 dengan `{"message":"Unauthenticated."}`, dari `auth:sanctum`; tidak ada pengalihan karena permintaannya meminta JSON. (c) 403, dari *middleware* `can:manage-`

`prices`, yang bertanya kepada *gate*. (d) 429, dari `throttle:5,1` pada rute POST; *controller* bahkan tidak berjalan, sehingga kata sandinya tidak diperiksa. (e) Pengalihan kembali ke halaman pesanan dengan catatan merah "Order #… has left the shop and is closed.", dari *middleware* `order.open` Bab 9, yang menjawab manusia dengan catatan, bukan dengan 409; pembayarannya tidak dicatat.

2. Kode 10.22 adalah salah satu penyelesaiannya. Uji ini mengisi daftar harga karena formulir harga mendaftar layanan yang sudah ada, dan memeriksa basis data selain statusnya: jawaban 403 yang tetap menyimpan layanan baru adalah kekeliruan. Data yang dikirim ke `POST /prices` sah menurut aturan Bab 8, sehingga kegagalan hanya dapat berasal dari hak akses, bukan dari validasi.

Kode 10.22 `tests/Feature/PriceListAccessTest.php`: daftar harga milik pemilik, di setiap rute.

```php
<?php

namespace Tests\Feature;

use App\Models\User;
use Database\Seeders\PriceListSeeder;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Tests\TestCase;

// Exercise 10.2: the price list belongs to the owner, on every route.
class PriceListAccessTest extends TestCase
{
    use RefreshDatabase;

    protected function setUp(): void
    {
        parent::setUp();
        $this->seed(PriceListSeeder::class);
    }

    public function test_a_cashier_cannot_open_or_send_the_price_form(): void
    {
        $cashier = User::factory()->create();

        $this->actingAs($cashier)->get('/prices/create')->assertForbidden();
        $this->actingAs($cashier)->post('/prices', [
            'description' => 'Carpet (per kg)', 'price' => 20000,
        ])->assertForbidden();
        $this->assertDatabaseMissing('services',
            ['description' => 'Carpet (per kg)']);
    }

    public function test_the_owner_can_open_the_price_form(): void
    {
        $this->actingAs(User::factory()->owner()->create())
            ->get('/prices/create')->assertOk();
    }

    public function test_a_guest_is_sent_to_sign_in_first(): void
    {
        $this->get('/prices/create')->assertRedirect('/login');
    }
}
```

3. Di `TokenController::store`, pilihlah kemampuan berdasarkan peran: `$abilities = $user->isOwner() ? ['*'] : ['orders:write', 'payments:write'];` lalu buatlah token dengan `$user->createToken($data['device'], $abilities)`. Di `bootstrap/app.php`, daftarkan kedua

middleware Sanctum dengan nama lazimnya dengan menambahkan `'abilities' => \Laravel\Sanctum\Http\Middleware\CheckAbilities::class` dan `'ability' => \Laravel\Sanctum\Http\Middleware\CheckForAnyAbility::class` ke larik `alias` [4]. Di `routes/api.php`, tambahkan `->middleware('abilities:customers:write')` pada rute `POST customers`. Ponsel kasir yang mencoba membuat pelanggan kemudian menerima 403, karena tokennya tidak memiliki kemampuan itu, sedangkan token pemilik, dengan `*`, memiliki semua kemampuan. Token yang diterbitkan sebelum perubahan tetap membawa kemampuan yang dimilikinya saat dibuat, jadi mintalah para sopir untuk masuk ulang.

4. Aturan ini menyangkut satu jenis model, yaitu pesanan, dan bergantung pada pengguna serta pesanan itu, sehingga tempatnya di *policy*: `OrderPolicy` dengan metode `updateNote(User $user, Order $order): bool` yang mengembalikan `$user->isOwner() || $order->received_at->isToday()`. Laravel menemukan `App\Policies\OrderPolicy` untuk `App\Models\Order` berdasarkan namanya. Rute yang menyimpan catatan memakai `->middleware('can:updateNote,order')`, dan *request* formulir catatan juga dapat memanggil `$this->user()->can('updateNote', $this->route('order'))` di `authorize()`. *Gate* juga dapat dipakai tetapi akan mengumpulkan aturan-aturan yang tidak saling berkaitan di satu *provider*; *form request* saja hanya akan melindungi rute yang memakai *request* itu.

5. `createToken` membangkitkan *string* acak, hanya menyimpan *hash* SHA-256-nya di `personal_access_tokens`, bersama id pengguna, nama perangkat, kemampuan, serta waktu pembuatan dan pemakaian terakhir, lalu mengembalikan *string* polosnya sekali. Karena hanya *hash*-nya yang disimpan, server tidak dapat menampilkan token itu lagi, dan salinan basis data tidak memberi siapa pun token yang dapat dipakai [4]. Ketika sebuah ponsel dicuri, pemilik menghapus baris perangkat itu (misalnya dengan `$user->tokens()->where('name', 'driver-phone-rina')->delete()` di `php artisan tinker`), yang langsung mengunci ponsel itu, dan meminta sopir mengganti kata sandinya bila kata sandi itu mungkin tersimpan di ponselnya.

6. Laravel menyerialkan tanggal dalam format ISO 8601 pada zona waktu UTC, dengan mikrodetik: misalnya `2026-09-25T01:15:00.000000Z`. Huruf `Z` (“Zulu”) berarti UTC, dan `.000000` adalah mikrodetiknya. Surabaya berada pada Waktu Indonesia Barat, tujuh jam lebih awal dari UTC, sehingga pesanan yang diterima pukul 08.15 menurut jam meja kasir dikirim sebagai 01.15 UTC. Nilainya benar, tetapi program harus mengonversinya, dan kerangka kerja lain dapat memilih format berbeda untuk saat yang sama. Bab 12 menjadikan persis hal ini sebuah aturan dalam kontrak API dan mengubah `OrderResource` sesuai dengan aturan itu.

Daftar Pustaka

[1] Laravel, “Release Notes,” *Laravel 13.x Documentation*, Mar. 17, 2026. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/releases

[2] Laravel, “Authentication,” *Laravel 13.x Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/authentication

[3] Laravel, “Authorization,” *Laravel 13.x Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/authorization

[4] Laravel, “Laravel Sanctum,” *Laravel 13.x Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/sanctum

[5] Laravel, “Eloquent: API Resources,” *Laravel 13.x Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/eloquent-resources

[6] Laravel, “HTTP Tests,” *Laravel 13.x Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/http-tests

[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] M. Jones and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage," RFC 6750, IETF, Oct. 2012. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6750

[9] Laravel, "Database Testing," *Laravel 13.x Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/database-testing

[10] Laravel, "Hashing," *Laravel 13.x Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/hashing

[11] OWASP Foundation, "Authentication Cheat Sheet," *OWASP Cheat Sheet Series*. Accessed: Sep. 25, 2026. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

[12] Pest, "Pest: The elegant PHP testing framework," pestphp.com. Accessed: Sep. 25, 2026. [Online]. Available: https://pestphp.com/

Bab 11

Spring Boot I: Initializr, REST Controller, dan Spring Data JPA

Profitina Laundry kini berjalan di atas dua kerangka kerja: meja kasir ASP.NET Core dari Bab 3 sampai Bab 6 dan meja kasir serta API Laravel dari Bab 7 sampai Bab 10. Bab ini membangun toko itu untuk ketiga kalinya, dalam Java dengan Spring Boot, kerangka kerja yang dipakai untuk menulis sangat banyak perangkat lunak bisnis di dunia, dari bank hingga perusahaan logistik. Tujuannya bukan sekadar salinan ketiga. Melihat pelanggan, layanan, pesanan, dan pembayaran yang sama dalam kerangka kerja ketiga memisahkan apa yang hakiki dari sebuah aplikasi web dari apa yang hanya kebiasaan satu kerangka kerja. Di akhir bab ini Anda akan mampu membuat proyek Spring Boot 4.1 dengan Spring Initializr dan menjelaskan setiap bagiannya; menggambarkan apa yang dikerjakan untuk Anda oleh *container* Spring, injeksi dependensi, dan *auto-configuration* Boot; memetakan empat tabel toko ke entitas JPA, termasuk kunci primer berupa teks, tabel yang namanya adalah kata kunci SQL, dan *enum* yang disimpan sebagai kata berhuruf kecil; memperoleh kueri dari *repository* Spring Data JPA tanpa menulis implementasinya; membangun REST *controller* yang memvalidasi masukan JSON dan menjawab JSON dengan nama isian yang sama dengan API Laravel; serta memeriksa aturan toko dalam Java biasa, dengan keluaran yang identik dengan pemeriksaan PHP Bab 8.

11.1 Pertanyaan Raka

Raka adalah keponakan Dimas, yang sedang menulis aplikasi ponsel untuk para sopir. Ia mahasiswa jurusan informatika, dan di kampusnya bahasa untuk sisi *back end* adalah Java. Ketika membaca API Laravel Bab 10 untuk menulis aplikasinya, ia mengajukan pertanyaan yang cepat atau lambat diajukan setiap pengembang: “Apakah ini akan tampak sama di Spring Boot?” Bu Wati memiliki versi pertanyaannya sendiri. Sebuah kantor akuntan yang mengurus pembukuan beberapa usaha *laundry* di Surabaya menawarkan untuk menghubungkan sistemnya dengan sistem Bu Wati, dan sistem kantor itu ditulis dalam Java.

Maka bab ini menjawab pertanyaan Raka dengan membangun API Profitina Laundry di Spring Boot, dengan empat tabel yang sama, daftar harga yang sama yang dibaca dari `Service.csv` yang sama, aturan yang sama (minimum tiga kilogram, satuan utuh, biaya jemput-antar dan gratis mulai Rp 100.000, serta cucian yang hanya boleh pergi bila sudah lunas), dan nama isian JSON yang sama. Bab 12 kemudian membandingkan kedua API itu isian demi isian, dan Bab 13 serta Bab 14 menumbuhkan versi Spring menjadi aplikasi lengkap dengan halaman, keamanan, dan pengujian.

Spring Boot 4 adalah generasi yang berlaku saat ini. Versi 4.0 dirilis pada November 2025 [1], versi 4.1 pada Juni 2026, dan versi yang dipakai buku ini, 4.1.1, pada Agustus 2026 [2]. Spring Boot berjalan di Java 17 atau yang lebih baru dan diuji hingga Java 26; buku ini memakai Java 25, rilis JDK dengan dukungan jangka panjang yang berlaku saat ini [3], [4]. Di sisi web ia menyematkan Tomcat 11, dan di bawah akses datanya ada Hibernate ORM 7, implementasi standar Jakarta Persistence yang paling banyak dipakai [3], [5].

11.2 Gagasannya dengan Kata-Kata Sederhana

11.2.1 Container yang membangun objek Anda

Bab 2 menjelaskan injeksi dependensi: alih-alih sebuah kelas membangun objek yang diperlukannya dengan `new`, kelas itu menyatakannya sebagai parameter konstruktor, dan pihak lain membangunnya lalu menyerahkannya. Di ASP.NET Core pihak itu adalah *service container* di `Program.cs`; di Laravel pihak itu adalah *service container* yang membangun *controller* dan *form request*. Di Spring pihak itu adalah *application context*, yang tertua dan paling terkenal di antara semuanya [6].

Ketika aplikasi Spring Boot mulai berjalan, *context* memindai paket-paket aplikasi untuk mencari kelas yang ditandai anotasi seperti `@RestController`, `@Component`, atau `@Service`, serta antarmuka yang menurunkan `JpaRepository` milik Spring Data. Masing-masing menjadi sebuah *bean*: satu objek, dibangun sekali, yang dikelola *context*. Ketika membangun `OrderController`, *context* melihat bahwa konstruktornya memerlukan `OrderRepository`, `CustomerRepository`, dan `ServiceRepository`, membangun atau menemukan *bean-bean* itu, lalu menyerahkannya. Kode Anda tidak pernah memanggil `new OrderController(...)`; ia hanya menyatakan apa yang diperlukannya.

11.2.2 Boot: bawaan yang wajar, dinyalakan oleh dependensi Anda

Spring biasa dapat melakukan hampir apa saja, tetapi dulu memerlukan sangat banyak konfigurasi untuk itu. *Spring Boot* menambahkan tiga hal yang menghilangkan sebagian besar pekerjaan itu. *Starter* adalah dependensi yang membawa sekumpulan pustaka yang serasi dalam satu baris: `spring-boot-starter-webmvc` membawa Spring MVC, Jackson untuk JSON, dan Tomcat tertanam; `spring-boot-starter-data-jpa` membawa Hibernate dan Spring Data JPA. *Auto-configuration* melihat apa yang ada di *class path* lalu mengonfigurasinya dengan bawaan yang wajar: bila Hibernate dan *driver* basis data H2 tersedia dan tidak ada basis data yang dikonfigurasi, Boot membuat basis data H2 di memori dan menghubungkan Hibernate dengannya. Dan *server tertanam* berarti aplikasinya adalah program Java biasa dengan metode `main`, dijalankan dengan `mvn spring-boot:run` atau `java -jar`, tanpa server terpisah yang harus dipasang.

Spring Boot 4 memecah *auto-configuration*-nya menjadi modul-modul yang lebih kecil, satu untuk setiap teknologi, sehingga aplikasi hanya memuat apa yang dipakainya; *starter* web kini bernama `spring-boot-starter-webmvc`, dan setiap *starter* utama memiliki *starter* uji pasangannya [7]. Proyek-proyek di buku ini memakai nama-nama yang baru.

11.2.3 Entitas, repository, dan SQL yang tidak Anda tulis

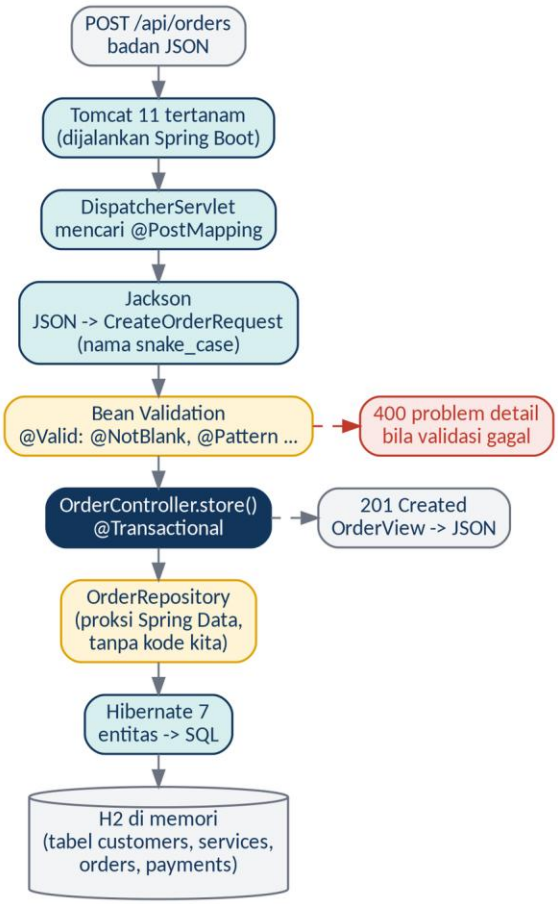
Jakarta Persistence (JPA) adalah standar Java untuk memetakan kelas ke tabel [8]. *Entitas* adalah kelas yang ditandai `@Entity`, dengan satu *field* untuk setiap kolom dan anotasi untuk rincian yang tidak dapat ditebak: nama tabel, kunci primer, panjang kolom teks, relasi ke entitas lain. Hibernate membaca anotasi itu lalu menulis SQL-nya: `INSERT` ketika Anda menyimpan objek baru, `UPDATE` ketika Anda mengubahnya di dalam transaksi, `SELECT` dengan *join* yang tepat ketika Anda memintanya.

Spring Data JPA melangkah satu tahap lebih jauh. Anda menyatakan antarmuka seperti `OrderRepository extends JpaRepository<Order, Long>`, dan Spring membuat implementasinya ketika aplikasi mulai berjalan: `findById`, `findAll`, `save`, `count`, `deleteById`, dan

belasan lainnya tersedia begitu saja. Lebih baik lagi, metode yang namanya mengikuti pola, seperti `findByPhone(String phone)` atau `findByStatusOrderByReceivedAtDescIdDesc(...)`, diubah menjadi kueri berdasarkan namanya [9]. Gagasannya sama dengan `Customer::where('phone', $phone)->first()` di Eloquent, hanya ditulis sebagai tanda tangan metode alih-alih rangkaian panggilan.

11.2.4 Controller yang berbicara JSON

Kelas yang ditandai `@RestController` menjawab permintaan HTTP dengan data, bukan halaman. Setiap metode dipetakan ke sebuah URL dan metode HTTP dengan anotasi seperti `@GetMapping("/{id}")`; parameternya diisi dari URL (`@PathVariable`), *query string* (`@RequestParam`), dan badan JSON (`@RequestBody`); dan apa pun yang dikembalikannya diubah menjadi JSON oleh Jackson [10]. `@Valid` di depan parameter badan menjalankan Jakarta Validation terhadapnya sebelum metode dipanggil, memakai anotasi seperti `@NotBlank` dan `@DecimalMin` pada *field* kelas permintaan [11]. **Gambar 11.1** mengikuti satu permintaan, yaitu pembuatan pesanan, melalui bagian-bagian ini.



Gambar 11.1 Satu `POST /api/orders` melalui aplikasi Spring Boot: Tomcat tertanam, DispatcherServlet, Jackson, Bean Validation, *controller*, *repository* Spring Data, Hibernate, dan basis data.

Tabel 11.1 menyandingkan bagian-bagian Spring Boot dengan bagian-bagian Laravel dari Bab 7 sampai Bab 10 yang memainkan peran yang sama. Sebagian besar bab ini adalah belajar mengenali kawan lama dalam pakaian baru.

Tabel 11.1 Pekerjaan yang sama di Laravel dan di Spring Boot.

Pekerjaan	Laravel (Bab 7 sampai 10)	Spring Boot (bab ini)
memulai proyek	`composer create-project laravel/laravel`	Spring Initializr, https://start.spring.io
dependensi	`composer.json`	`pom.xml` (Maven)
pengaturan	`.env`, `config/*.php`	`application.properties`
model	model Eloquent	`@Entity` JPA
kueri	`Order::where(...)->get()`	metode *repository* Spring Data
skema	migrasi	`ddl-auto` Hibernate di sini; Flyway atau Liquibase kemudian
rute	`routes/api.php`	`@GetMapping`, `@PostMapping` pada metode
aturan masukan	*form request*	*record* dengan anotasi Jakarta Validation
bentuk JSON	*API resource*	*record* respons (Jackson)
data isian	*seeder*	*bean* `CommandLineRunner`
menjalankan	`php artisan serve`	`mvn spring-boot:run`

11.3 Membuat Proyek

11.3.1 Spring Initializr

Setiap proyek Spring Boot dimulai dengan cara yang sama, di Spring Initializr [12]. Halaman web https://start.spring.io, perintah "Spring Initializr" di paket ekstensi Spring Boot untuk VS Code, dan fungsi yang sama di penyunting lain semuanya mengajukan pertanyaan yang sama: Maven atau Gradle (pilih Maven), Java, versi Spring Boot (4.1.x), *group* `com.profitina`, *artifact* `laundry`, Java 25, dan dependensinya. Pilihlah empat: **Spring Web**, **Spring Data JPA**, **Validation**, dan **H2 Database**. Initializr menjawab dengan berkas zip berisi proyek lengkap yang siap dijalankan; folder `spring` bab ini adalah proyek itu yang sudah ditambahi kode Profitina Laundry. Bab 1 sudah memasang JDK 25 dan Maven yang diperlukannya, dan `README.md` bab ini mengulang perintah-perintahnya untuk Windows, macOS, dan Linux.

11.3.2 Berkas build

Kode 11.1 adalah `pom.xml` proyek, berkas yang dibaca Maven untuk mengetahui proyek apa ini dan apa yang diperlukannya.

Kode 11.1 `pom.xml`: Spring Boot 4.1.1 sebagai induk, Java 25, dan lima dependensi.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>4.1.1</version>
    <relativePath/>
  </parent>
  <groupId>com.profitina</groupId>
  <artifactId>laundry</artifactId>
  <version>11.0.0</version>
  <properties>
    <java.version>25</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-webmvc</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
```

Kode 11.1 (Lanjutan) *pom.xml: Spring Boot 4.1.1 sebagai induk, Java 25, dan lima dependensi.*

```
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

`<parent>` adalah bagian terpenting. Dengan mewarisi `spring-boot-starter-parent` versi 4.1.1, proyek ini mewarisi daftar versi yang sudah teruji untuk ratusan pustaka, dan karena itulah tidak satu pun dependensi di bawahnya memiliki `<version>`: Spring Boot sudah memilih versi Hibernate, Jackson, Tomcat, dan *driver* H2 yang dapat bekerja sama. `<java.version>25</java.version>` membuat Maven mengompilasi untuk Java 25. *Driver* H2 memiliki `<scope>runtime</scope>` karena kode kita tidak pernah merujuknya secara langsung, dan *starter* uji memiliki `<scope>test</scope>` karena hanya diperlukan oleh uji. `spring-boot-`

`maven-plugin` menambahkan *goal* `spring-boot:run` dan mengemas aplikasi menjadi satu berkas jar yang dapat dijalankan dengan `mvn package`.

11.3.3 Tata letak dan titik masuk

Proyek ini mengikuti tata letak baku Maven. Kode Java berada di `src/main/java/com/profitina/laundry`, dalam tiga paket: `domain` untuk entitas dan aturan, `repository` untuk antarmuka Spring Data, dan `web` untuk *controller* serta *record* yang dibaca dan ditulisnya. Pengaturan dan berkas data berada di `src/main/resources`, dan uji di `src/test/java`. **Kode 11.2** adalah kelas yang memulai segalanya.

Kode 11.2 `LaundryApplication.java`: seluruh aplikasi dimulai di sini.

```
package com.profitina.laundry;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class LaundryApplication {

    public static void main(String[] args) {
        SpringApplication.run(LaundryApplication.class, args);
    }
}
```

`@SpringBootApplication` menyalakan tiga hal sekaligus: kelas ini adalah sumber konfigurasi, *auto-configuration* diaktifkan, dan pemindaian komponen dimulai di paket kelas ini, `com.profitina.laundry`, serta mencakup setiap paket di bawahnya. Karena itulah ketiga paket tadi harus berada *di bawah* kelas aplikasi; *controller* yang ditempatkan di `com.profitina.other` tidak akan pernah ditemukan. `SpringApplication.run` membuat *application context*, membangun setiap *bean*, menjalankan Tomcat tertanam di porta 8080, lalu menunggu permintaan.

11.3.4 Pengaturan

Kode 11.3 memuat pengaturan yang berbeda dari bawaan Boot.

Kode 11.3 `application.properties`: basis data di memori, JSON snake_case, dan *problem details* untuk galat.

```
spring.application.name=profitina-laundry
server.port=8080

# An in-memory H2 database: empty at every start, filled by DemoData.
spring.datasource.url=jdbc:h2:mem:laundry
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.open-in-view=false

# JSON names in snake_case (customer_id), the same as the Laravel API.
spring.jackson.property-naming-strategy=SNAKE_CASE
# Errors as RFC 9457 problem details.
spring.mvc.problemdetails.enabled=true
```

`jdbc:h2:mem:laundry` adalah basis data H2 [13] yang hidup di memori dan lenyap ketika aplikasi berhenti. Itu ideal untuk perkenalan pertama: setiap kali dijalankan, tokonya bersih, lalu diisi oleh kode awal di Subbab 11.7. `ddl-auto=create-drop` membuat Hibernate membangun tabel dari entitas ketika aplikasi mulai dan membuangnya di akhir; itu wajar untuk basis data di memori dan keliru untuk basis data sungguhan, yang tabel-tabelnya harus dikelola dengan migrasi,

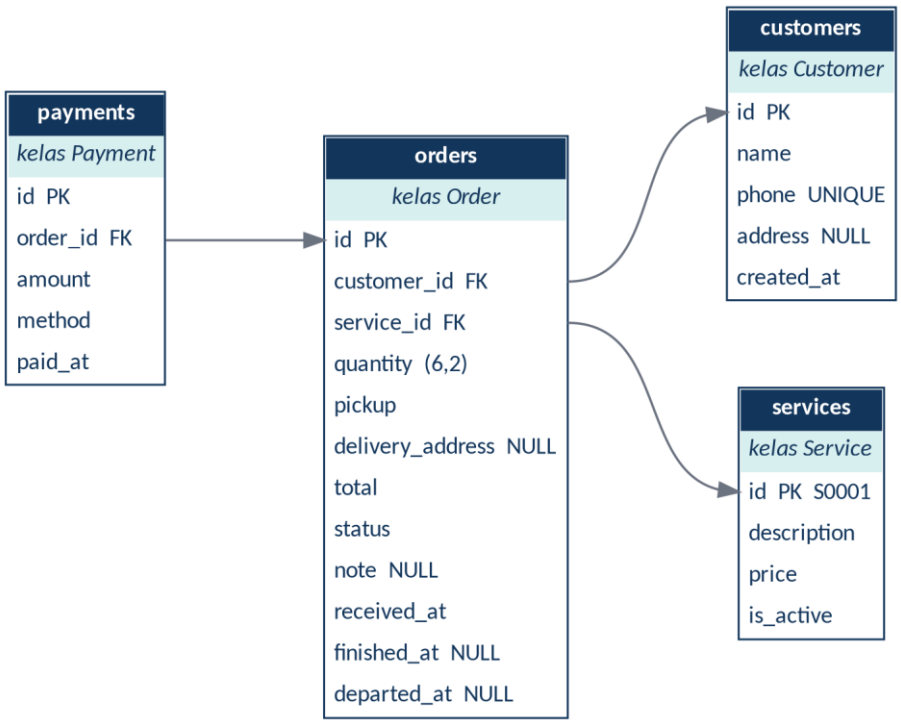
seperti yang diperlihatkan Bab 9. `open-in-view=false` mematikan kemudahan Spring yang membiarkan sesi basis data tetap terbuka selama JSON ditulis; dengan kemudahan itu dimatikan, setiap metode *controller* harus memuat apa yang diperlukannya di dalam transaksinya, sehingga kueri-kuerinya terlihat, bukan tersembunyi.

Dua pengaturan terakhir menyangkut dunia luar. `property-naming-strategy=SNAKE_CASE` membuat Jackson menulis dan membaca nama isian JSON dalam snake_case: *field* Java `deliveryAddress` menjadi `delivery_address`, persis seperti di API Laravel, sehingga aplikasi ponsel yang sama dapat berbicara dengan keduanya [14]. `problemdetails.enabled=true` membuat Spring menjawab galat dalam format baku *problem details* dari RFC 9457, yaitu objek JSON dengan `type`, `title`, `status`, dan `detail` [15].

11.4 Domain: Entitas

11.4.1 Empat tabel

Gambar 11.2 memperlihatkan keempat tabel beserta kelas Java-nya. Tabel-tabel itu adalah tabel `LaundryDbContext` Bab 5 dan migrasi Bab 7, kolom demi kolom; satu-satunya tabel yang ditinggalkan adalah `outlets`, yang belum diperlukan versi Spring.



Gambar 11.2 Empat tabel Profitina Laundry dan kelas entitas yang memetakannya; panah mengarah dari *foreign key* ke kunci primer yang dirujuknya.

11.4.2 Customer

Kode 11.4 adalah entitas yang paling sederhana.

Kode 11.4 `Customer.java`: seorang pelanggan, dengan nomor telepon yang unik.

```
package com.profitina.laundry.domain;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import java.time.LocalDateTime;

@Entity
@Table(name = "customers")
public class Customer {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, length = 80)
    private String name;

    @Column(nullable = false, length = 20, unique = true)
    private String phone;

    @Column(length = 200)
    private String address;

    @Column(nullable = false)
    private LocalDateTime createdAt;

    protected Customer() {                // for JPA only
    }

    public Customer(String name, String phone, String address,
                    LocalDateTime createdAt) {
        this.name = name;
        this.phone = phone;
        this.address = address;
        this.createdAt = createdAt;
    }
```

Kode 11.4 (Lanjutan) *Customer.java: seorang pelanggan, dengan nomor telepon yang unik.*

```
    public Long getId() { return id; }
    public String getName() { return name; }
    public String getPhone() { return phone; }
    public String getAddress() { return address; }
    public LocalDateTime getCreatedAt() { return createdAt; }
}
```

`@Entity` menandai kelas ini untuk Hibernate dan `@Table(name = "customers")` memberi nama tabelnya; tanpa anotasi itu tabelnya akan bernama `customer`. `@Id` dengan `@GeneratedValue(strategy = GenerationType.IDENTITY)` menjadikan `id` kunci primer yang dinomori oleh basis data sendiri, seperti `$table->id()` di Laravel. `@Column` memberikan rincian yang diberikan migrasi Bab 7: nama wajib diisi dan paling banyak 80 karakter, telepon paling banyak 20 dan unik, alamat paling banyak 200 dan opsional. Strategi penamaan Spring Boot mengubah *field* `createdAt` menjadi kolom `created_at` secara otomatis, sehingga nama kolomnya sama dengan Laravel tanpa anotasi tambahan.

JPA mewajibkan konstruktor tanpa argumen, agar Hibernate dapat membuat objek kosong lalu mengisinya dari sebuah baris. Menjadikannya `protected` memungkinkan Hibernate memakainya sambil mencegah kode kita sendiri membuat pelanggan tanpa nama. Konstruktor publik

menerima semua yang diperlukan pelanggan baru. Ada *getter* tetapi tidak ada *setter*: pelanggan diubah melalui metode yang menyatakan apa yang dilakukannya, yang ditambahkan Bab 13, bukan dengan mengisi *field* satu per satu.

11.4.3 Service, dan nama yang sudah dipakai

Kode 11.5 memetakan daftar harga. Kunci primernya bukan angka, melainkan kode dari Bab 3, seperti `S0001`, sehingga tidak ada `@GeneratedValue`: kode itu milik kita, dan kitalah yang mengisinya.

Kode 11.5 `Service.java`: satu baris daftar harga, dengan kodenya sendiri sebagai kunci, serta nama dan satuan yang diturunkan dari deskripsinya.

```
package com.profitina.laundry.domain;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

/** A line of the price list, e.g. S0001 "Wash, dry & fold (per kg)". */
@Entity
@Table(name = "services")
public class Service {

    @Id
    @Column(length = 5)
    private String id;                      // S0001: our code, not counted

    @Column(nullable = false, length = 60)
    private String description;

    @Column(nullable = false)
    private long price;                     // rupiah per kg, item or pair

    @Column(name = "is_active", nullable = false)
    private boolean active = true;

    protected Service() {
    }

    public Service(String id, String description, long price) {
        this.id = id;
        this.description = description;
        this.price = price;
    }

    /** "Wash, dry & fold (per kg)" -> "Wash, dry & fold". */
    public String name() {
        int cut = description.lastIndexOf(" (");
        return cut < 0 ? description : description.substring(0, cut);
    }

    public boolean perKg() {
        return description.endsWith("(per kg)");
    }
```

Kode 11.5 (Lanjutan) *Service.java: satu baris daftar harga, dengan kodenya sendiri sebagai kunci, serta nama dan satuan yang diturunkan dari deskripsinya.*

```
    public String unit() {
        return perKg() ? "kg"
                : description.endsWith("(per pair)") ? "pair" : "item";
    }

    public String getId() { return id; }
    public String getDescription() { return description; }
    public long getPrice() { return price; }
    public boolean isActive() { return active; }
}
```

Field `active` disimpan di kolom `is_active`, nama yang dipakai migrasi Bab 7, berkat `@Column(name = "is_active")`. Metode `name()`, `perKg()`, dan `unit()` melakukan persis apa yang dilakukan properti `Name`, `PerKg`, dan `Unit` pada `Service` C# Bab 5 serta *accessor* pada `Service` Eloquent Bab 7: "Wash, dry & fold (per kg)" memiliki nama "Wash, dry & fold" dan satuan "kg"; "Shoes deep clean (per pair)" memiliki satuan "pair"; selebihnya dihitung per *item*. Aturannya berada di satu tempat untuk setiap kerangka kerja dan sama di ketiganya.

Kelas ini bernama `Service`, seperti modelnya di dua kerangka kerja lain, dan itu menimbulkan satu masalah kecil di Spring: Spring juga memiliki anotasi bernama `@Service`, untuk kelas-kelas lapisan layanan yang diperkenalkan Bab 13. Java tidak dapat mengimpor dua hal dengan nama pendek yang sama ke dalam satu berkas, sehingga berkas yang memakai keduanya harus menulis salah satunya secara lengkap, `@org.springframework.stereotype.Service`. Bab 13 melakukan persis hal itu. Mempertahankan nama `Service` untuk daftar harga sepadan dengan biaya kecil itu, karena kata yang sama lalu berarti hal yang sama di ketiga versi toko.

11.4.4 Order dan Payment

Kode 11.6 adalah jantung domain. Dua hal menonjol bahkan sebelum membaca kodenya: tabelnya bernama `orders`, karena `ORDER` adalah kata tercadang dalam SQL (`ORDER BY`) dan tabel bernama `order` harus diberi tanda kutip di setiap kueri; dan kelas ini dibuat dengan konstruktor yang menghitung harga pesanan, sehingga tidak ada pesanan yang dapat ada tanpa totalnya.

Kode 11.6 `Order.java`: sebuah pesanan, diberi harga ketika dibuat, dimajukan oleh alur kerja toko.

```java
package com.profitina.laundry.domain;

import jakarta.persistence.CascadeType;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.FetchType;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.OneToMany;
import jakarta.persistence.Table;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import org.hibernate.annotations.BatchSize;

@Entity
@Table(name = "orders")                    // ORDER is a word of SQL
public class Order {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "customer_id")
    private Customer customer;

    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "service_id")
    private Service service;

    @Column(nullable = false, precision = 6, scale = 2)
    private BigDecimal quantity;

    @Column(nullable = false)
    private boolean pickup;

    @Column(length = 200)
    private String deliveryAddress;
```

Kode 11.6 (Lanjutan) *Order.java: sebuah pesanan, diberi harga ketika dibuat, dimajukan oleh alur kerja toko.*

```
    @Column(nullable = false)
    private long total;                     // rupiah, fixed at creation

    @Column(nullable = false, length = 12)
    private OrderStatus status = OrderStatus.QUEUED;

    @Column(length = 200)
    private String note;

    @Column(nullable = false)
    private LocalDateTime receivedAt;

    private LocalDateTime finishedAt;

    private LocalDateTime departedAt;

    @OneToMany(mappedBy = "order", cascade = CascadeType.ALL,
               orphanRemoval = true)
    @BatchSize(size = 50)                   // payments of 50 orders at once
    private List<Payment> payments = new ArrayList<>();

    protected Order() {
    }

    /** A new order, priced by the shop's rule the moment it is made. */
    public Order(Customer customer, Service service, BigDecimal quantity,
                 boolean pickup, String deliveryAddress, String note,
                 LocalDateTime receivedAt) {
        this.customer = customer;
        this.service = service;
        this.quantity = quantity.setScale(2, RoundingMode.UNNECESSARY);
        this.pickup = pickup;
        this.deliveryAddress = deliveryAddress;
        this.note = note;
        this.receivedAt = receivedAt;
        this.total = LaundryRules.total(service.perKg(), service.getPrice(),
                quantity, pickup);
    }
```

Kode 11.6 (Lanjutan) *Order.java: sebuah pesanan, diberi harga ketika dibuat, dimajukan oleh alur kerja toko.*

```
    /** One step along the workflow, stamped with the time it happened. */
    public void advance(LocalDateTime when) {
        status = LaundryRules.advance(status, balance());
        if (status == OrderStatus.FINISHED) {
            finishedAt = when;
        } else {
            departedAt = when;
        }
    }

    public long paid() {
        return payments.stream().mapToLong(Payment::getAmount).sum();
    }

    public long balance() {
        return total - paid();
    }

    public Long getId() { return id; }
    public Customer getCustomer() { return customer; }
    public Service getService() { return service; }
    public BigDecimal getQuantity() { return quantity; }
    public boolean isPickup() { return pickup; }
    public String getDeliveryAddress() { return deliveryAddress; }
    public long getTotal() { return total; }
    public OrderStatus getStatus() { return status; }
    public String getNote() { return note; }
    public LocalDateTime getReceivedAt() { return receivedAt; }
    public LocalDateTime getFinishedAt() { return finishedAt; }
    public LocalDateTime getDepartedAt() { return departedAt; }
    public List<Payment> getPayments() { return payments; }
}
```

`@ManyToOne(fetch = FetchType.LAZY, optional = false)` menyatakan bahwa banyak pesanan dimiliki satu pelanggan, bahwa setiap pesanan harus memiliki satu pelanggan, dan bahwa pelanggan itu tidak dimuat sampai ada yang memintanya. `@JoinColumn(name = "customer_id")` memberi nama kolom *foreign key*-nya. Bawaan `@ManyToOne` adalah pemuatan *eager*, yang mengambil pelanggan bersama setiap pesanan, diperlukan atau tidak; *lazy* adalah bawaan yang lebih baik, dan Subbab 11.5 memperlihatkan cara memuat relasinya ketika diperlukan. `@OneToMany(mappedBy = "order", ...)` adalah sisi lain dari relasi `Payment`: `mappedBy` menyatakan bahwa *foreign key*-nya berada di tabel `payments`, pada *field* `order` di `Payment`. `cascade = CascadeType.ALL` ikut menyimpan pembayaran baru yang ditambahkan ke daftar ketika pesanan disimpan, dan `orphanRemoval = true` menghapus pembayaran yang dikeluarkan dari daftar. `@BatchSize(size = 50)`, dari Hibernate, membuat pembayaran hingga lima puluh pesanan dimuat dalam satu kueri ketika sebuah daftar pesanan memerlukannya, alih-alih satu kueri per pesanan.

Jumlahnya berupa `BigDecimal` dengan presisi 6 dan skala 2, sama dengan `decimal(6,2)` di migrasi Bab 7. `setScale(2, RoundingMode.UNNECESSARY)` menyimpan "3" sebagai 3.00 dan melempar *exception* bila jumlah dengan lebih dari dua angka desimal sempat lolos dari validasi, alih-alih membulatkannya diam-diam. Uang berupa `long` dalam rupiah utuh; tidak ada bilangan titik mengambang di dekat harga mana pun.

Konstruktor meminta total kepada `LaundryRules`, dan `advance` meminta status berikutnya kepadanya, lalu mencap waktunya: `finishedAt` untuk *finished*, `departedAt` untuk *departed*.

`paid()` dan `balance()` menghitung apa yang dihitung properti `Paid` dan `Balance` Bab 5. **Kode 11.7** adalah pembayaran, entitas terakhir dari keempatnya.

Kode 11.7 `Payment.java`: uang yang diterima untuk sebuah pesanan; pesanan boleh dibayar bertahap.

```
package com.profitina.laundry.domain;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.FetchType;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Table;
import java.time.LocalDateTime;

/** Money received for an order; an order may be paid in parts. */
@Entity
@Table(name = "payments")
public class Payment {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "order_id")
    private Order order;

    @Column(nullable = false)
    private long amount;

    @Column(nullable = false, length = 10)
    private PaymentMethod method;

    @Column(nullable = false)
    private LocalDateTime paidAt;

    protected Payment() {
    }

    public Payment(Order order, long amount, PaymentMethod method,
                   LocalDateTime paidAt) {
        this.order = order;
        this.amount = amount;
        this.method = method;
        this.paidAt = paidAt;
    }
```

Kode 11.7 (Lanjutan) *Payment.java: uang yang diterima untuk sebuah pesanan; pesanan boleh dibayar bertahap.*

```
    public Long getId() { return id; }
    public Order getOrder() { return order; }
    public long getAmount() { return amount; }
    public PaymentMethod getMethod() { return method; }
    public LocalDateTime getPaidAt() { return paidAt; }
}
```

11.4.5 Status yang disimpan seperti di kerangka kerja lain

Enum Java lazimnya dinamai dengan huruf kapital, `QUEUED`, sedangkan basis data Laravel Bab 7 menyimpan kata `queued`, `finished`, dan `departed`. Pilihan JPA yang lazim, `@Enumerated(EnumType.STRING)`, akan menyimpan `QUEUED`, dan kedua aplikasi tidak dapat berbagi basis data atau bahkan bersepakat dalam JSON-nya. **Kode 11.8** memberi setiap konstanta *enum* katanya, dan **Kode 11.9** memberi tahu JPA untuk menyimpan kata itu.

Kode 11.8 `OrderStatus.java`: tiga status, masing-masing dengan kata yang dipakai basis data dan JSON.

```
package com.profitina.laundry.domain;

/** The life of an order; value() is the word stored and sent as JSON. */
public enum OrderStatus {
    QUEUED("queued"),
    FINISHED("finished"),
    DEPARTED("departed");

    private final String value;

    OrderStatus(String value) {
        this.value = value;
    }

    public String value() {
        return value;
    }

    /** The status that follows this one, or null when the order has left. */
    public OrderStatus next() {
        return switch (this) {
            case QUEUED -> FINISHED;
            case FINISHED -> DEPARTED;
            case DEPARTED -> null;
        };
    }

    /** "finished" -> FINISHED; an unknown word gives an exception. */
    public static OrderStatus fromValue(String value) {
        for (OrderStatus s : values()) {
            if (s.value.equals(value)) {
                return s;
            }
        }
        throw new IllegalArgumentException("Unknown status: " + value);
    }
}
```

Kode 11.9 `OrderStatusConverter.java`: menyimpan `queued`, bukan `QUEUED`, di setiap kolom status.

```java
package com.profitina.laundry.domain;

import jakarta.persistence.AttributeConverter;
import jakarta.persistence.Converter;

/** Stores "queued", not "QUEUED": the same text as the Laravel app. */
@Converter(autoApply = true)
public class OrderStatusConverter
        implements AttributeConverter<OrderStatus, String> {

    @Override
    public String convertToDatabaseColumn(OrderStatus status) {
        return status == null ? null : status.value();
    }

    @Override
    public OrderStatus convertToEntityAttribute(String value) {
        return value == null ? null : OrderStatus.fromValue(value);
    }
}
```

`AttributeConverter` memiliki dua metode, satu untuk setiap arah, dan `@Converter(autoApply = true)` menerapkannya pada setiap *field* bertipe `OrderStatus` di setiap entitas, sehingga `Order` tidak memerlukan anotasi pada *field* `status`-nya [8]. `PaymentMethod` beserta *converter*-nya, di folder bab ini, melakukan hal yang sama untuk `cash`, `qris`, dan `transfer`. `next()` adalah kembaran Java dari `next()` pada *enum* PHP di Bab 7: *queued* ke *finished*, *finished* ke *departed*, dan `null` sesudahnya.

11.4.6 Aturannya, dalam Java biasa

Kode 11.10 adalah versi Spring dari `PriceCalculator` dan `OrderWorkflow` Bab 8, dalam satu kelas yang tidak tahu apa pun tentang Spring, JPA, atau HTTP.

Kode 11.10 `LaundryRules.java`: aturan harga dan alur kerja, dengan aritmetika bilangan bulat yang sama dengan versi PHP.

```
package com.profitina.laundry.domain;

import java.math.BigDecimal;
import java.util.Locale;

/**
 * The shop's rules as plain Java: no Spring, no database. The same
 * arithmetic as PriceCalculator and OrderWorkflow in the Laravel app.
 */
public final class LaundryRules {
    public static final long MINIMUM_KG = 3;
    public static final long PICKUP_FEE = 10_000;
    public static final long FREE_PICKUP_FROM = 100_000;

    private LaundryRules() {
    }

    /** 4.5 -> 450: the quantity in hundredths, exact. */
    public static long hundredths(BigDecimal quantity) {
        return quantity.movePointRight(2).longValueExact();
    }

    /** At least 3 kg for per-kg services; whole items otherwise. */
    public static long total(boolean perKg, long price, BigDecimal quantity,
                             boolean pickup) {
        long h = hundredths(quantity);
        long billed = perKg ? Math.max(h, MINIMUM_KG * 100)
                            : (h + 99) / 100 * 100;
        long subtotal = (price * billed + 50) / 100;       // half up
        long fee = pickup && subtotal < FREE_PICKUP_FROM ? PICKUP_FEE : 0;
        return subtotal + fee;
    }

    /** The next status, or an exception that says why there is none. */
    public static OrderStatus advance(OrderStatus status, long balance) {
        OrderStatus next = status.next();
        if (next == null) {
            throw new IllegalStateException(
                    "This order has already left the shop.");
        }
        if (next == OrderStatus.DEPARTED && balance > 0) {
            throw new IllegalStateException("Collect " + rupiah(balance)
                    + " before handing over the laundry.");
        }
        return next;
    }
```

Kode 11.10 (Lanjutan) *LaundryRules.java: aturan harga dan alur kerja, dengan aritmetika bilangan bulat yang sama dengan versi PHP.*

```
    /** 36000 -> "Rp 36.000", as Money::rp does in PHP. */
    public static String rupiah(long amount) {
        return "Rp " + String.format(Locale.ROOT, "%,d", amount)
                .replace(',', '.');
    }
}
```

Aritmetikanya adalah aritmetika Bab 8, langkah demi langkah. Jumlah diubah menjadi perseratusan: 4,5 menjadi 450, tepat, karena `movePointRight(2)` pada `BigDecimal` menggeser titik desimal tanpa pembulatan apa pun, dan `longValueExact` lebih suka melempar *exception* daripada membuang satu angka. Layanan per kilogram menagih sedikitnya 300 perseratusan, yaitu tiga kilogram; layanan lain dibulatkan ke atas menjadi satuan utuh dengan `(h + 99) / 100 * 100`. Subtotalnya adalah harga dikali perseratusan yang ditagih lalu dibagi 100, dibulatkan

setengah ke atas dengan menambahkan 50 sebelum pembagian bilangan bulat. Jemput-antar berbiaya Rp 10.000 bila subtotal di bawah Rp 100.000 dan gratis mulai dari angka itu. `advance` menolak memajukan pesanan yang sudah pergi dan menolak membiarkan cucian pergi bila masih ada tagihan, dengan pesan yang sama dengan `OrderWorkflow` PHP, dan `rupiah` menulis "Rp 36.000" seperti yang dilakukan `Money::rp`.

Karena kelas ini hanya memerlukan JDK, ia dapat dikompilasi dan dijalankan tanpa Maven, Spring, atau jaringan. `check/RulesCheck.java` bab ini menjalankan sembilan kasus harga dan empat langkah alur kerja dari `tools/core_check.php` Bab 8:

```
javac -d out -sourcepath src/main/java check/RulesCheck.java
java -cp out RulesCheck
```

```
Prices (Problem 8.1)
  Wash, dry & fold             4.50 kg            36000
  Wash, dry & fold             2.00 kg            24000
  Wash, dry & fold             4.02 kg            32160
  Ironing only                 3.35 kg            20100
  Curtain                      4.40 kg   pickup   76000
  Bed cover, any size          2.00 item pickup   80000
  Dry cleaning - suit 2 pieces 2.00 item pickup  130000
  Dry cleaning - shirt/blouse  5.00 item pickup  100000
  Shoes deep clean             1.50 pair          90000
Workflow (Problem 8.2)
  Queued   owing  25000 -> Finished
  Finished owing  25000 -> Collect Rp 25.000 before handing over the laundry.
  Finished owing      0 -> Departed
  Departed owing      0 -> This order has already left the shop.
```

Keluaran ini dihasilkan oleh program tersebut, lalu dibandingkan bait demi bait dengan keluaran `php tools/core_check.php` dari Bab 8: keduanya identik. Tiga kerangka kerja, dua bahasa, dan satu daftar harga. Apa pun perbedaan lain di antara versi-versi toko ini, seorang pelanggan membayar jumlah yang sama di semuanya.

11.5 Repository

Kode 11.11 adalah sebuah *repository* utuh: antarmuka dengan satu metode tambahan, dan tidak ada lagi yang perlu ditulis.

Kode 11.11 `CustomerRepository.java`: setiap operasi baku untuk pelanggan, ditambah dua metode yang diturunkan dari namanya.

```java
package com.profitina.laundry.repository;

import com.profitina.laundry.domain.Customer;
import java.util.Optional;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CustomerRepository extends JpaRepository<Customer, Long> {

    Optional<Customer> findByPhone(String phone);

    boolean existsByPhone(String phone);
}
```

`JpaRepository<Customer, Long>` menyatakan bahwa *repository* ini mengelola entitas `Customer` yang id-nya bertipe `Long`. Spring Data membaca nama `findByPhone`: "find" berarti kueri, "By" memulai syarat, "Phone" adalah properti `phone`. Kueri yang dibangkitkan adalah `select c from Customer c where c.phone = ?1`, dan tipe kembalian `Optional` berarti "satu pelanggan atau tidak ada". `existsByPhone` mengajukan pertanyaan yang sama dengan jawaban `boolean`, yang

lebih murah bila pelanggannya sendiri tidak diperlukan [9]. `ServiceRepository` menurunkan `JpaRepository<Service, String>`, karena id layanan adalah kodenya, dan menambahkan `findByActiveTrueOrderByIdAsc`, yaitu daftar harga yang aktif menurut urutan kode. `PaymentRepository` belum memerlukan metode tambahan.

Kode 11.12 adalah *repository* yang dipakai *endpoint* pesanan.

Kode 11.12 `OrderRepository.java`: halaman-halaman pesanan, disaring menurut status, dan satu pesanan dengan semua yang diperlukannya.

```
package com.profitina.laundry.repository;

import com.profitina.laundry.domain.Order;
import com.profitina.laundry.domain.OrderStatus;
import java.util.Optional;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.EntityGraph;
import org.springframework.data.jpa.repository.JpaRepository;

public interface OrderRepository extends JpaRepository<Order, Long> {

    @EntityGraph(attributePaths = {"customer", "service"})
    Page<Order> findAllByOrderByReceivedAtDescIdDesc(Pageable pageable);

    @EntityGraph(attributePaths = {"customer", "service"})
    Page<Order> findByStatusOrderByReceivedAtDescIdDesc(OrderStatus status,
                                                        Pageable pageable);

    @EntityGraph(attributePaths = {"customer", "service", "payments"})
    Optional<Order> findWithDetailsById(Long id);
}
```

Nama-namanya panjang karena menyatakan segalanya. `findByStatusOrderByReceivedAtDescIdDesc(OrderStatus status, Pageable pageable)` berarti: pesanan dengan status ini, yang terbaru lebih dahulu, dan pesanan yang diterima pada detik yang sama diurutkan menurut id, satu halaman setiap kali. Parameter `Pageable` membawa nomor dan ukuran halaman, dan hasil `Page` membawa pesanan di halaman itu bersama jumlah keseluruhannya, yang darinya jumlah halaman dapat dihitung.

`@EntityGraph(attributePaths = {"customer", "service"})` adalah cara Spring Data mengatakan apa yang dikatakan Eloquent dengan `Order::with(['customer', 'service'])`: muatlah relasi *lazy* ini bersama pesanannya, dalam kueri yang sama. Tanpa itu, menulis JSON untuk satu halaman berisi dua puluh pesanan akan memuat pelanggan dan layanan setiap pesanan dengan kueri tersendiri: masalah "N+1 kueri" yang ditemui Bab 8 di Laravel. `findWithDetailsById` juga memuat pembayarannya, untuk halaman sebuah pesanan tunggal.

11.6 REST Controller

11.6.1 Record untuk yang keluar

Entitas bukan jawaban yang baik untuk sebuah API, dengan alasan yang sama seperti model Laravel di Bab 10: ia mengikuti basis data, ia akan membuka segalanya, dan menyerialkan relasi *lazy*-nya di luar transaksi akan gagal. Karena itu versi Spring menjawab dengan *record*, kelas data Java yang ringkas dan tidak dapat diubah, yang memainkan peran *API resource* Laravel. **Kode 11.13** adalah pesanannya.

Kode 11.13 `OrderView.java`: JSON satu pesanan, dibangun dari entitasnya.

```
package com.profitina.laundry.web;

import com.profitina.laundry.domain.Order;
import java.math.BigDecimal;
import java.time.LocalDateTime;

/** The JSON of one order: the same field names as Laravel's resource. */
public record OrderView(
        Long id,
        CustomerView customer,
        ServiceView service,
        BigDecimal quantity,
        boolean pickup,
        String deliveryAddress,
        long total,
        long paid,
        long balance,
        String status,
        String note,
        LocalDateTime receivedAt,
        LocalDateTime finishedAt,
        LocalDateTime departedAt) {

    public static OrderView of(Order o) {
        return new OrderView(o.getId(), CustomerView.of(o.getCustomer()),
                ServiceView.of(o.getService()), o.getQuantity(),
                o.isPickup(), o.getDeliveryAddress(), o.getTotal(), o.paid(),
                o.balance(), o.getStatus().value(), o.getNote(),
                o.getReceivedAt(), o.getFinishedAt(), o.getDepartedAt());
    }
}
```

Setiap komponen *record* menjadi satu isian JSON, dan strategi penamaan snake_case di **Kode 11.3** mengubah `deliveryAddress` menjadi `delivery_address` dan `receivedAt` menjadi `received_at`: nama yang sama dengan *resource* Laravel. Metode statis `of` membangun *record* dari sebuah `Order`, dengan membaca pelanggan dan layanannya, yang karena itu harus sudah dimuat; *entity graph* pada *repository* memastikannya. `CustomerView` dan `ServiceView`, di folder bab ini, adalah dua *record* lain yang sejenis, dan `ServiceView` memakai `name()` dan `unit()` dari entitasnya. Satu halaman pesanan dibungkus dalam `PageView` (**Kode 11.14**), yang memberi halaman itu bagian `data` dan `meta` yang sama dengan *resource* berhalaman di Laravel, dengan halaman dihitung mulai dari 1.

Kode 11.14 `PageView.java`: satu halaman sebagai `data` dan `meta`, dengan halaman dihitung mulai dari 1 seperti di Laravel.

```java
package com.profitina.laundry.web;

import java.util.List;
import java.util.function.Function;
import org.springframework.data.domain.Page;

/** {data: [...], meta: {...}}, with pages counted from 1 as in Laravel. */
public record PageView<T>(List<T> data, Meta meta) {

    public record Meta(int currentPage, int perPage, long total,
                       int lastPage) {
    }

    public static <E, T> PageView<T> of(Page<E> page, Function<E, T> map) {
        return new PageView<>(page.getContent().stream().map(map).toList(),
                new Meta(page.getNumber() + 1, page.getSize(),
                        page.getTotalElements(),
                        Math.max(1, page.getTotalPages())));
    }
}
```

Spring Data menghitung halaman mulai dari 0, dan sebuah `Page` juga memiliki `getTotalPages()` bernilai 0 bila tidak ada hasil; `PageView.of` mengubah keduanya menjadi apa yang diharapkan manusia dan API Laravel.

11.6.2 Record untuk yang masuk

Kode 11.15 adalah masukan untuk pesanan baru, beserta aturannya.

Kode 11.15 `CreateOrderRequest.java`: isian pesanan baru dan aturannya, termasuk satu aturan tentang dua isian.

```java
package com.profitina.laundry.web;

import jakarta.validation.constraints.AssertTrue;
import jakarta.validation.constraints.DecimalMax;
import jakarta.validation.constraints.DecimalMin;
import jakarta.validation.constraints.Digits;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Size;
import java.math.BigDecimal;

public record CreateOrderRequest(
        @NotNull Long customerId,
        @NotBlank String serviceId,
        @NotNull @DecimalMin("0.1") @DecimalMax("100")
        @Digits(integer = 3, fraction = 2) BigDecimal quantity,
        boolean pickup,
        @Size(max = 200) String deliveryAddress,
        @Size(max = 200) String note) {

    /** A rule about two fields at once: pickup needs an address. */
    @AssertTrue(message = "Where should the laundry be delivered?")
    public boolean isAddressGivenForPickup() {
        return !pickup || (deliveryAddress != null
                && !deliveryAddress.isBlank());
    }
}
```

Jackson mengisi *record* dari badan JSON, `customer_id` ke `customerId` dan seterusnya, lalu `@Valid` di *controller* memeriksa anotasinya [11]. `@NotNull` dan `@NotBlank` membuat isian wajib; `@DecimalMin("0.1")`, `@DecimalMax("100")`, dan `@Digits(integer = 3, fraction = 2)` adalah

ejaan Java untuk `min:0.1`, `max:100`, dan `decimal:0,2` di Laravel. Aturan "jemput-antar memerlukan alamat" melibatkan dua isian, sehingga tidak dapat menjadi anotasi pada salah satunya. Metode yang ditandai `@AssertTrue` diperiksa seperti isian yang harus bernilai benar, dan metode itu dapat melihat seluruh *record*: Jakarta Validation memperlakukan `isAddressGivenForPickup()` sebagai properti bernama `addressGivenForPickup`. `CreateCustomerRequest` di folder bab ini memeriksa pelanggan baru, dan *compact constructor*-nya, yaitu blok tanpa parameter yang berjalan ketika *record* dibangun, menyederhanakan telepon menjadi angka-angka sebelum aturan mana pun melihatnya, seperti yang dilakukan `prepareForValidation` di Bab 10.

11.6.3 Controller-nya

Kode 11.16 adalah *controller* terkecil, daftar harga.

Kode 11.16 `ServiceController.java`: daftar harga yang aktif, dan satu layanan menurut kodenya.

```
package com.profitina.laundry.web;

import com.profitina.laundry.repository.ServiceRepository;
import java.util.List;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.server.ResponseStatusException;

@RestController
@RequestMapping("/api/services")
public class ServiceController {

    private final ServiceRepository services;

    public ServiceController(ServiceRepository services) {
        this.services = services;          // handed in by Spring
    }

    @GetMapping
    public List<ServiceView> index() {
        return services.findByActiveTrueOrderByIdAsc().stream()
                .map(ServiceView::of).toList();
    }

    @GetMapping("/{id}")
    public ServiceView show(@PathVariable String id) {
        return services.findById(id).map(ServiceView::of)
                .orElseThrow(() -> new ResponseStatusException(
                        HttpStatus.NOT_FOUND, "No service " + id + "."));
    }
}
```

`@RequestMapping("/api/services")` menetapkan awal URL yang sama; `@GetMapping` tanpa argumen menjawab `GET /api/services`, dan `@GetMapping("/{id}")` menjawab `GET /api/services/S0001`, dengan `@PathVariable String id` menerima "S0001". *Repository*-nya tiba melalui konstruktor: Spring melihat satu-satunya konstruktor itu dan menyuntikkan *bean*-nya, tanpa perlu anotasi. Ketika sebuah layanan tidak ada, `ResponseStatusException` dengan `HttpStatus.NOT_FOUND` membuat Spring menjawab 404, dalam format *problem details*. `CustomerController`, di folder bab ini, mendaftar pelanggan, mencari satu pelanggan dengan

`?phone=`, dan membuat pelanggan dengan `POST /api/customers`, menjawab 409 Conflict bila teleponnya sudah terdaftar.

Kode 11.17 adalah *controller* untuk pesanan.

Kode 11.17 `OrderController.java`: mendaftar, menampilkan, membuat, dan memajukan pesanan, masing-masing di dalam transaksi.

```
package com.profitina.laundry.web;

import com.profitina.laundry.domain.Customer;
import com.profitina.laundry.domain.Order;
import com.profitina.laundry.domain.OrderStatus;
import com.profitina.laundry.domain.Service;
import com.profitina.laundry.repository.CustomerRepository;
import com.profitina.laundry.repository.OrderRepository;
import com.profitina.laundry.repository.ServiceRepository;
import jakarta.validation.Valid;
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.http.HttpStatus;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PatchMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.server.ResponseStatusException;

@RestController
@RequestMapping("/api/orders")
public class OrderController {

    private final OrderRepository orders;
    private final CustomerRepository customers;
    private final ServiceRepository services;

    public OrderController(OrderRepository orders,
                           CustomerRepository customers,
                           ServiceRepository services) {
        this.orders = orders;
        this.customers = customers;
        this.services = services;
    }
```

Kode 11.17 (Lanjutan) *OrderController.java: mendaftar, menampilkan, membuat, dan memajukan pesanan, masing-masing di dalam transaksi.*

```
    /** GET /api/orders?status=finished&page=2 (pages start at 1). */
    @GetMapping
    @Transactional(readOnly = true)
    public PageView<OrderView> index(
            @RequestParam(required = false) String status,
            @RequestParam(defaultValue = "1") int page) {
        PageRequest pageable = PageRequest.of(Math.max(page, 1) - 1, 20);
        Page<Order> result = status == null
                ? orders.findAllByOrderByReceivedAtDescIdDesc(pageable)
                : orders.findByStatusOrderByReceivedAtDescIdDesc(
                        parseStatus(status), pageable);
        return PageView.of(result, OrderView::of);
    }

    @GetMapping("/{id}")
    @Transactional(readOnly = true)
    public OrderView show(@PathVariable Long id) {
        return OrderView.of(find(id));
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    @Transactional
    public OrderView store(@Valid @RequestBody CreateOrderRequest in) {
        Customer customer = customers.findById(in.customerId())
                .orElseThrow(() -> unprocessable("No customer "
                        + in.customerId() + "."));
        Service service = services.findById(in.serviceId())
                .filter(Service::isActive)
                .orElseThrow(() -> unprocessable("The service "
                        + in.serviceId() + " is not offered."));
        Order order = new Order(customer, service, in.quantity(),
                in.pickup(), in.deliveryAddress(), in.note(), now());
        return OrderView.of(orders.save(order));
    }
```

Kode 11.17 (Lanjutan) *OrderController.java: mendaftar, menampilkan, membuat, dan memajukan pesanan, masing-masing di dalam transaksi.*

```
    /** PATCH /api/orders/5/status {"status": "finished"}: one step on. */
    @PatchMapping("/{id}/status")
    @Transactional
    public OrderView updateStatus(@PathVariable Long id,
                                  @Valid @RequestBody StatusRequest in) {
        Order order = find(id);
        OrderStatus wanted = parseStatus(in.status());
        if (wanted != order.getStatus().next()) {
            throw new ResponseStatusException(HttpStatus.CONFLICT,
                    "An order that is " + order.getStatus().value()
                    + " cannot become " + wanted.value() + ".");
        }
        try {
            order.advance(now());
        } catch (IllegalStateException e) {
            throw new ResponseStatusException(HttpStatus.CONFLICT,
                    e.getMessage());
        }
        return OrderView.of(order);        // saved when the method commits
    }

    private Order find(Long id) {
        return orders.findWithDetailsById(id)
                .orElseThrow(() -> new ResponseStatusException(
                        HttpStatus.NOT_FOUND, "No order " + id + "."));
    }

    private static OrderStatus parseStatus(String value) {
        try {
            return OrderStatus.fromValue(value);
        } catch (IllegalArgumentException e) {
            throw unprocessable(e.getMessage());
        }
    }

    private static ResponseStatusException unprocessable(String reason) {
        return new ResponseStatusException(HttpStatus.UNPROCESSABLE_CONTENT,
                reason);
    }

    private static LocalDateTime now() {
        return LocalDateTime.now().truncatedTo(ChronoUnit.SECONDS);
    }
}
```

`index` menerima dua parameter kueri, keduanya opsional: `@RequestParam(required = false) String status` bernilai `null` bila URL tidak memiliki `?status=`, dan `@RequestParam(defaultValue = "1") int page` bernilai 1 bila tidak ada `?page=`. Metode ini meminta halaman yang tepat dengan `PageRequest.of(page - 1, 20)` lalu membungkus hasilnya dengan `PageView.of(result, OrderView::of)`: `OrderView::of` adalah *method reference*, fungsi yang mengubah setiap pesanan menjadi tampilannya.

Setiap metode membawa `@Transactional`. Transaksi *read-only* untuk pembacaan memberi tahu Hibernate bahwa tidak ada yang akan berubah, sehingga ia dapat melewati sebagian pekerjaan; transaksi baca-tulis untuk perubahan data mengelompokkan semua yang dilakukan metode itu menjadi satu kesatuan yang terjadi seluruhnya atau tidak sama sekali. Transaksi itu juga menjaga sesi basis data tetap terbuka selama `OrderView.of` membaca relasi *lazy*. `store` menjawab 201 Created karena `@ResponseStatus(HttpStatus.CREATED)`. Metode ini mencari pelanggan dan layanan yang aktif, menjawab 422 bila salah satunya tidak ada, lalu membuat pesanan dengan

konstruktor yang menghitung harganya, dan menyimpannya. `updateStatus` memeriksa bahwa status yang diminta adalah status berikutnya dan membiarkan `Order.advance` menerapkan alur kerja; metode ini tidak memerlukan panggilan `save`, karena Hibernate memperhatikan perubahan pada entitas yang dimuat di dalam transaksi dan menuliskannya ketika transaksi di-*commit*. Di bab ini langkah yang ditolak dijawab dengan 409 Conflict. Bab 12 membandingkan pilihan itu dengan pilihan API Laravel, yang menjawab 422, lalu memutuskannya.

Menaruh `@Transactional` pada metode *controller* masih dapat diterima untuk API pertama yang kecil. Bab 13 memindahkan pekerjaannya ke *service layer* (lapisan layanan), yaitu kelas-kelas yang ditandai `@org.springframework.stereotype.Service` yang memiliki transaksi dan pemakaian *repository*, sehingga *controller* hanya menerjemahkan antara HTTP dan domain.

11.7 Memuat Daftar Harga

Laravel mengisi basis datanya dengan *seeder*; aplikasi Spring Boot biasanya melakukan hal yang sama dengan *bean* yang berjalan sekali setelah aplikasi dimulai. Setiap *bean* yang mengimplementasikan `CommandLineRunner` dipanggil metode `run`-nya ketika *application context* sudah siap. **Kode 11.18** adalah milik kita.

Kode 11.18 `DemoData.java`: daftar harga sungguhan dari `Service.csv`, serta tiga pelanggan Bab 5 dan Bab 7 beserta pesanannya.

```
package com.profitina.laundry;

import com.profitina.laundry.domain.Customer;
import com.profitina.laundry.domain.Order;
import com.profitina.laundry.domain.Payment;
import com.profitina.laundry.domain.PaymentMethod;
import com.profitina.laundry.domain.Service;
import com.profitina.laundry.repository.CustomerRepository;
import com.profitina.laundry.repository.OrderRepository;
import com.profitina.laundry.repository.ServiceRepository;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigDecimal;
import java.nio.charset.StandardCharsets;
import java.time.LocalDate;
import java.time.LocalDateTime;
import org.springframework.boot.CommandLineRunner;
import org.springframework.core.io.ClassPathResource;
import org.springframework.stereotype.Component;

/** Spring's counterpart of Laravel's seeders: runs once at start-up. */
@Component
public class DemoData implements CommandLineRunner {

    private final ServiceRepository services;
    private final CustomerRepository customers;
    private final OrderRepository orders;

    public DemoData(ServiceRepository services, CustomerRepository customers,
                    OrderRepository orders) {
        this.services = services;
        this.customers = customers;
        this.orders = orders;
    }

    @Override
    public void run(String... args) throws IOException {
        if (services.count() > 0) {
            return;                          // already filled: do nothing
        }
        loadPriceList();
```

Kode 11.18 (Lanjutan) *DemoData.java: daftar harga sungguhan dari Service.csv, serta tiga pelanggan Bab 5 dan Bab 7 beserta pesanannya.*

```
        LocalDateTime today = LocalDate.now().atStartOfDay();
        Customer sari = customers.save(new Customer("Sari Wulandari",
                "081200000101", "Jl. Kertajaya 12, Surabaya", today));
        Customer budi = customers.save(new Customer("Budi Santoso",
                "081200000102", null, today));
        customers.save(new Customer("Dewi Lestari", "081200000103",
                "Jl. Manyar 7, Surabaya", today));

        Order first = new Order(sari, service("S0001"), new BigDecimal("4.5"),
                false, null, null, today.minusDays(1).plusHours(8));
        first.getPayments().add(new Payment(first, 36_000, PaymentMethod.QRIS,
                today.minusDays(1).plusHours(8)));
        first.advance(today.plusHours(9));              // queued -> finished
        orders.save(first);

        Order second = new Order(budi, service("S0002"), new BigDecimal("3"),
                false, null, null, today.plusMinutes(450));
        second.getPayments().add(new Payment(second, 20_000,
                PaymentMethod.CASH, today.plusMinutes(450)));
        orders.save(second);

        orders.save(new Order(budi, service("S0008"), new BigDecimal("2"),
                true, "Jl. Mulyosari 30, Surabaya", null,
                today.plusHours(10)));
    }

    /** data/Service.csv: the same price list as Chapters 3, 5 and 7. */
    private void loadPriceList() throws IOException {
        var csv = new ClassPathResource("data/Service.csv");
        try (var in = new BufferedReader(new InputStreamReader(
                csv.getInputStream(), StandardCharsets.UTF_8))) {
            in.readLine();                  // the header line
            for (String line; (line = in.readLine()) != null; ) {
                int first = line.indexOf(',');
                int last = line.lastIndexOf(',');
                String description = line.substring(first + 1, last)
                        .replace("\"", "");
                long price = new BigDecimal(line.substring(last + 1))
                        .longValueExact();
                services.save(new Service(line.substring(0, first),
                        description, price));
            }
        }
    }
```

Kode 11.18 (Lanjutan) *DemoData.java: daftar harga sungguhan dari Service.csv, serta tiga pelanggan Bab 5 dan Bab 7 beserta pesanannya.*

```
    private Service service(String id) {
        return services.findById(id).orElseThrow();
    }
}
```

`@Component` menjadikan kelas ini sebuah *bean*, dan konstruktornya menerima tiga *repository*. `run` lebih dahulu memeriksa apakah daftar harga sudah ada, sehingga kelas ini tetap tidak berbahaya bila basis datanya kelak dibuat permanen. `loadPriceList` membaca `data/Service.csv` dari *class path*, berkas yang sama dengan Bab 3, 5, dan 7, disalin ke `src/main/resources/data`. Penguraian CSV-nya sengaja dibuat sederhana: id berada sebelum koma pertama, harga setelah koma terakhir, dan deskripsinya, yang dapat memuat koma dan karena itu tanda kutip, adalah semua yang ada di antaranya. Lalu Sari, Budi, dan Dewi dibuat dengan pesanan yang mereka miliki di

seeder Bab 7. Pesanan Sari diberi harga oleh konstruktor sebesar Rp 36.000, dibayar dengan QRIS, dan dimajukan ke *finished*; pesanan kilat Budi seberat 3 kg memiliki pembayaran pertama Rp 20.000; dan dua *bed cover*-nya dengan jemput-antar berharga Rp 80.000.

11.8 Menjalankannya

Dengan JDK 25 dan Maven terpasang seperti di Bab 1, bukalah terminal di folder `spring` bab ini dan jalankan:

```
mvn spring-boot:run
```

Jalan pertama mengunduh Spring Boot beserta pustakanya dari Maven Central, yang memakan beberapa menit; jalan berikutnya dimulai dalam hitungan detik. Ketika log menunjukkan bahwa Tomcat sudah berjalan di porta 8080, bukalah http://localhost:8080/api/services di peramban: kedua belas layanan dari `Service.csv` muncul sebagai JSON, masing-masing dengan `id`, `name`, `unit`, dan `price`. http://localhost:8080/api/orders menampilkan tiga pesanan demo sebagai satu halaman dengan `data` dan `meta`. Untuk membuat pesanan, pakailah perintah `curl` di `README.md` bab ini; perintah itu mendaftarkan pesanan pertama Dewi, 2,5 kg "Wash, dry & fold" dengan jemput-antar, dan jawabannya membawa `total` sebesar 34000, yaitu tiga kilogram seharga Rp 8.000 ditambah biaya jemput-antar, persis seperti yang dihitung API Laravel di Bab 10. Kirimkan permintaan yang sama dengan `"quantity": 0` dan tanpa alamat: jawabannya 400 dengan badan *problem details*. Hentikan aplikasi dengan Ctrl+C; karena basis datanya di memori, jalan berikutnya kembali menjadi toko yang bersih.

Di VS Code, dengan Extension Pack for Java dan Spring Boot Extension Pack, bukalah folder `spring`, tunggu sampai proyek Java-nya dimuat, lalu tekan tautan **Run** di atas `main` pada `LaundryApplication.java`; Spring Boot Dashboard di bilah samping juga dapat menjalankan dan menghentikan aplikasinya.

Lingkungan tempat buku ini ditulis tidak dapat mengunduh paket Maven, sehingga aplikasinya sendiri tidak dijalankan di sana. Semua yang ada di `domain/LaundryRules.java` dan `check/RulesCheck.java` sudah dikompilasi dan dijalankan, dengan keluaran yang diperlihatkan di Subbab 11.4.6; berkas Java lainnya dikompilasi terhadap tiruan kecil tipe-tipe Spring, Jakarta, dan JUnit yang dipakainya, tanpa galat maupun peringatan. `README.md` bab ini menjelaskan dengan tepat apa yang dijalankan dan bagaimana caranya.

11.9 Menguji Aturannya

Kode 11.19 adalah uji pertama proyek Spring. Seperti `PriceCalculatorTest` di Bab 10, uji ini memeriksa aturannya saja, tanpa menjalankan Spring.

Kode 11.19 `LaundryRulesTest.java`: aturan harga dan alur kerja, diuji tanpa Spring.

```
package com.profitina.laundry.domain;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;

import java.math.BigDecimal;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;

/** No Spring, no database: the rules alone, fast. */
class LaundryRulesTest {

    @ParameterizedTest
    @CsvSource({
        "true, 8000, 4.5, false, 36000",
        "true, 8000, 2.5, false, 24000",
        "true, 8000, 2.5, true, 34000",
        "false, 35000, 1.5, false, 70000",
        "false, 65000, 2, true, 130000",
        "true, 15000, 3.35, false, 50250",
    })
    void totalFollowsTheRule(boolean perKg, long price, String quantity,
                             boolean pickup, long expected) {
        assertEquals(expected, LaundryRules.total(perKg, price,
                new BigDecimal(quantity), pickup));
    }

    @Test
    void laundryLeavesOnlyWhenPaid() {
        assertEquals(OrderStatus.FINISHED,
                LaundryRules.advance(OrderStatus.QUEUED, 25_000));
        assertThrows(IllegalStateException.class,
                () -> LaundryRules.advance(OrderStatus.FINISHED, 25_000));
        assertEquals(OrderStatus.DEPARTED,
                LaundryRules.advance(OrderStatus.FINISHED, 0));
    }
}
```

`@ParameterizedTest` dengan `@CsvSource` menjalankan `totalFollowsTheRule` sekali untuk setiap baris, mengubah setiap nilai yang dipisahkan koma menjadi tipe parameternya; keenam baris itu adalah keenam kasus uji PHP di Bab 10, dengan total yang diharapkan yang sama. `assertThrows` memeriksa bahwa memajukan pesanan *finished* yang belum lunas melempar *exception* yang diubah *controller* menjadi galat. `mvn test` menjalankan uji ini dengan JUnit, yang disediakan *starter* uji. Bab 14 menambahkan uji yang menjalankan seluruh aplikasi dan mengirim permintaan HTTP kepadanya, seperti yang dilakukan Bab 10 untuk Laravel.

11.10 Membuatnya Lebih Baik

Basis data yang menetap. Mengganti URL H2 dengan basis data berbasis berkas atau server, SQLite seperti di Bab 13, atau MySQL atau PostgreSQL di toko dengan beberapa komputer, membuat data tetap ada di antara jalan-jalan aplikasi. Pada saat yang sama `ddl-auto` sebaiknya menjadi `validate` atau `none`, dan skemanya dikelola dengan migrasi melalui Flyway atau Liquibase, yang didukung Spring Boot 4 melalui *starter* masing-masing [7].

Profil. `application-dev.properties` dan `application-prod.properties` memuat pengaturan yang berbeda antara laptop dan toko, dan `--spring.profiles.active=prod` memilih salah satunya; data demo dapat dibatasi hanya untuk profil `dev` dengan `@Profile("dev")` pada `DemoData`, versi Spring dari pemeriksaan lingkungan di Bab 9.

Perangkat pengembang. Dependensi `spring-boot-devtools` menjalankan ulang aplikasi secara otomatis ketika sebuah kelas berubah, sehingga siklus sunting-dan-coba hampir secepat PHP.

Kesehatan dan metrik. `spring-boot-starter-actuator` menambahkan *endpoint* seperti `/actuator/health` yang dapat ditanyai server atau layanan pemantauan apakah aplikasinya masih hidup.

Bentuk galat yang sama dengan Laravel. *Problem details* adalah standar yang baik, tetapi bukan bentuk yang dikirim API Laravel. Bab 12 menulis satu kelas yang membuat setiap galat API Spring tampak seperti galat Laravel, sehingga aplikasi ponsel hanya memerlukan satu cara membaca galat.

11.11 Memakainya Sungguhan

- **Mulailah dari Initializr, bukan dari salinan.** Proyek salinan membawa versi lama dan pengaturan yang terlupakan. Initializr selalu memberikan versi Boot terkini dan sekumpulan pustaka yang serasi.
- **Jauhkan aturan dari kerangka kerja.** `LaundryRules` dapat dikompilasi dan dijalankan hanya dengan JDK, dan keluarannya dapat dibandingkan dengan versi PHP. Bagi usaha mana pun yang aturannya harus sama di beberapa sistem, seperti perhitungan bunga sebuah koperasi atau nomor antrean sebuah klinik, kelas biasa ditambah satu tabel kasus uji bersama adalah jaminan yang paling murah.
- **Samakan nama di tepi sistem.** Sistem kedua yang berbicara dengan sistem pertama, seperti milik kantor akuntan tadi, paling mudah dihubungkan bila nama isian JSON dan kata-kata yang disimpan sama. Strategi snake_case dan *converter enum* hanya memerlukan beberapa baris di sini dan menghemat satu lapisan penerjemahan kelak.
- **Anggaplah `ddl-auto` alat purwarupa.** Biarkan Hibernate membuat tabel selama modelnya masih berubah setiap jam; beralihlah ke migrasi sebelum pesanan sungguhan pertama disimpan.
- **Data pribadi tetap data pribadi dalam bahasa apa pun.** API Spring menyimpan nama, nomor telepon, dan alamat pelanggan yang sama dengan API Laravel, dan kewajiban UU PDP 27/2022 berlaku terhadapnya dengan cara yang sama. Sampai Bab 14 menambahkan Spring Security, jalankan API ini hanya di komputer Anda sendiri.

11.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Menempatkan kelas di luar paket aplikasi.** Pemindaian komponen dimulai di paket `LaundryApplication`; *controller* di paket yang bersebelahan tidak pernah ditemukan, dan URL-nya menjawab 404.
- **Menamai tabel `order`.** `ORDER` adalah kata kunci SQL. Namailah tabelnya `orders` dengan `@Table`.
- **Melupakan konstruktor tanpa argumen.** Hibernate memerlukannya untuk membuat entitas dari baris. Jadikan `protected`, agar kode Anda sendiri tidak memakainya secara tidak sengaja.
- **Relasi *eager* di mana-mana.** `@ManyToOne` memuat secara *eager* secara bawaan. Jadikan relasinya *lazy* dan muatlah dengan `@EntityGraph` di tempat yang memerlukannya.
- **Menyerialkan entitas secara langsung.** Jackson dapat mengikuti relasi *lazy* di luar transaksi lalu gagal, atau mengikuti `order.payments[0].order` berputar-putar. Jawablah dengan *record*.

- **Menyimpan *enum* menurut urutan atau menurut nama Java.** `EnumType.ORDINAL` menyimpan 0, 1, 2 dan rusak ketika sebuah konstanta disisipkan; `EnumType.STRING` menyimpan `QUEUED`, yang tidak diharapkan program lain. Simpanlah kata yang disepakati dengan *converter*.
- **Uang dalam `double`.** 0.1 + 0.2 tidak sama dengan 0.3 dalam bilangan titik mengambang biner. Pakailah `long` untuk rupiah utuh dan `BigDecimal` untuk jumlah.
- **Halaman mulai dari 0 di URL.** Spring Data menghitung halaman mulai dari 0, sedangkan manusia dan API Laravel mulai dari 1. Konversikan sekali, di satu tempat.
- **Lupa `@Transactional` pada metode yang membaca relasi *lazy*.** Dengan `open-in-view=false`, membaca `order.getCustomer().getName()` di luar transaksi melempar `LazyInitializationException`. Muatlah yang Anda perlukan dengan *entity graph*, di dalam transaksi.
- **Mengandalkan `ddl-auto=update` untuk basis data sungguhan.** Pengaturan itu menambahkan kolom tetapi tidak pernah membuang atau mengganti nama kolom, dan tidak dapat memindahkan data. Pakailah migrasi.

11.13 Ringkasan Bab

- Spring Boot 4.1 berjalan di Java 17 atau yang lebih baru (buku ini memakai Java 25) dan membawa *starter*, *auto-configuration*, dan Tomcat 11 tertanam; proyek dimulai di Spring Initializr, dan `pom.xml` mewarisi versi pustaka yang sudah teruji dari `spring-boot-starter-parent`.
- *Application context* Spring memindai paket-paket di bawah `@SpringBootApplication`, membuat *bean*, dan menyuntikkannya melalui konstruktor.
- Entitas JPA memetakan keempat tabel toko: kunci primer yang dibangkitkan dan yang berupa teks, relasi `@ManyToOne` dan `@OneToMany`, pemuatan *lazy*, tabel `orders`, dan *converter* yang menyimpan kata berhuruf kecil yang sama dengan Laravel.
- `LaundryRules` memuat aturan harga dan alur kerja dalam Java biasa; pemeriksaannya mencetak persis apa yang dicetak pemeriksaan PHP Bab 8.
- *Repository* Spring Data JPA adalah antarmuka; kuerinya berasal dari `JpaRepository` dan dari nama metode, dan `@EntityGraph` memuat relasi bersama kueri utamanya.
- Kelas `@RestController` memetakan URL ke metode, membaca variabel jalur, parameter kueri, dan badan JSON, memvalidasi *record* dengan Jakarta Validation, dan menjawab dengan *record* sebagai JSON snake_case dengan nama isian yang sama dengan API Laravel.
- *Bean* `CommandLineRunner` memuat daftar harga dari `Service.csv` dan pelanggan demo ketika aplikasi mulai, seperti yang dilakukan *seeder* Laravel.

11.14 Soal Latihan

1. Tulislah nama metode Spring Data untuk kueri-kueri berikut, dan sebutkan apa yang dikembalikan masing-masing: (a) pelanggan dengan telepon tertentu; (b) semua pesanan seorang pelanggan, yang terbaru lebih dahulu; (c) jumlah pesanan yang sudah *finished*; (d) apakah ada pesanan yang memakai layanan tertentu; (e) layanan aktif yang deskripsinya memuat kata tertentu, tanpa membedakan huruf besar dan kecil.
2. Tambahkan `GET /api/customers/{id}/orders`, yang mengembalikan pesanan seorang pelanggan, yang terbaru lebih dahulu, sebagai daftar `OrderView`, dan 404 bila pelanggannya tidak ada. Tulislah *controller* dan metode *repository* yang diperlukannya.

3. `OrderRepository.findAllByOrderByReceivedAtDescIdDesc` membawa `@EntityGraph(attributePaths = {"customer", "service"})`. Berapa kueri SQL yang dikirim `GET /api/orders` untuk satu halaman berisi dua puluh pesanan, dengan dan tanpa *entity graph*, termasuk pembayaran yang diperlukan `OrderView.of` melalui `paid()`? Apa yang diubah oleh `@BatchSize(size = 50)` pada `payments`?
4. Hitunglah dengan tangan, memakai `LaundryRules.total`, total untuk: 1,2 kg "Ironing only" (Rp 6.000 per kg) tanpa jemput-antar; 3 kemeja "Dry cleaning - shirt/blouse" (Rp 20.000 per *item*) dengan jemput-antar; 7,25 kg "Curtain" (Rp 15.000 per kg) dengan jemput-antar. Tunjukkan perseratusan, jumlah yang ditagih, subtotal, dan biayanya.
5. Jelaskan langkah demi langkah apa yang terjadi ketika JSON `{"customer_id": 1, "service_id": "S0001", "quantity": 2.555}` dikirim ke `POST /api/orders`: bagian mana yang membacanya, pemeriksaan mana yang gagal, dan apa jawabannya.
6. Untuk setiap bagian Laravel berikut, sebutkan bagian Spring Boot di bab ini yang mengerjakan hal yang sama, beserta satu perbedaannya: migrasi, *seeder*, *form request*, *API resource*, `Order::with('customer')`.

11.15 Penyelesaian Soal Latihan

1. (a) `Optional<Customer> findByPhone(String phone)`, satu pelanggan atau tidak ada. (b) `List<Order> findByCustomerIdOrderByReceivedAtDesc(Long customerId)`; `CustomerId` berjalan dari `Order.customer` ke `id`-nya, dan daftarnya boleh kosong. (c) `long countByStatus(OrderStatus status)`, sebuah angka. (d) `boolean existsByServiceId(String serviceId)`. (e) `List<Service> findByActiveTrueAndDescriptionContainingIgnoreCase(String word)`; `Containing` menambahkan *wildcard* `%` di kedua sisi dan `IgnoreCase` membandingkan dalam satu jenis huruf [9].

2. Tambahkan ke `OrderRepository` metode dari 1(b) dengan *entity graph*, agar tampilannya dapat membaca pelanggan dan layanan: `@EntityGraph(attributePaths = {"customer", "service"}) List<Order> findByCustomerIdOrderByReceivedAtDesc(Long customerId);`. **Kode 11.20** adalah *controller*-nya. *Controller* ini lebih dahulu memeriksa bahwa pelanggannya ada, karena daftar kosong akan menyembunyikan id yang salah ketik, dan membaca di dalam transaksi *read-only* karena `OrderView.of` membaca pembayaran melalui `paid()`.

Kode 11.20 `CustomerOrdersController.java`: pesanan seorang pelanggan, yang terbaru lebih dahulu, atau 404.

```
package com.profitina.laundry.web;

import com.profitina.laundry.repository.CustomerRepository;
import com.profitina.laundry.repository.OrderRepository;
import java.util.List;
import org.springframework.http.HttpStatus;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.server.ResponseStatusException;

/** Exercise 11.2: GET /api/customers/{id}/orders, newest first. */
@RestController
public class CustomerOrdersController {

    private final CustomerRepository customers;
    private final OrderRepository orders;

    public CustomerOrdersController(CustomerRepository customers,
                                    OrderRepository orders) {
        this.customers = customers;
        this.orders = orders;
    }

    @GetMapping("/api/customers/{id}/orders")
    @Transactional(readOnly = true)
    public List<OrderView> index(@PathVariable Long id) {
        if (!customers.existsById(id)) {
            throw new ResponseStatusException(HttpStatus.NOT_FOUND,
                    "No customer " + id + ".");
        }
        return orders.findByCustomerIdOrderByReceivedAtDesc(id).stream()
                .map(OrderView::of).toList();
    }
}
```

3. Tanpa *entity graph*: satu kueri untuk halaman pesanan, satu untuk menghitung semua pesanan bagi `meta.total`, lalu untuk setiap dari dua puluh pesanan satu kueri untuk pelanggannya, satu untuk layanannya, dan satu untuk pembayarannya, 2 + 60 = 62 kueri (lebih sedikit bila beberapa pesanan berbagi pelanggan atau layanan yang sudah dimuat dalam transaksi yang sama). Dengan *entity graph*, pelanggan dan layanan datang bersama pesanannya, sehingga tersisa 2 + 20 = 22. `@BatchSize(size = 50)` membuat Hibernate memuat pembayaran hingga lima puluh pesanan dalam satu kueri ketika salah satunya pertama kali diperlukan, sehingga dua puluh kueri pembayaran menjadi satu: seluruhnya 3 kueri.

4. *Ironing*: 1,2 kg adalah 120 perseratusan, di bawah minimum, sehingga yang ditagih 300; $6000 \times 300 = 1\,800\,000$, ditambah 50, dibagi 100 menghasilkan 18 000; tanpa jemput-antar; total Rp 18.000. Kemeja: 3 *item* adalah 300 perseratusan, sudah utuh; $20000 \times 300/100 = 60\,000$; jemput-antar di bawah Rp 100.000 menambah 10 000; total Rp 70.000. *Curtain*: 7,25 kg adalah 725 perseratusan, di atas minimum; $15000 \times 725 = 10\,875\,000$, ditambah 50 dan dibagi 100 menghasilkan 108 750 (sisa 50 dibuang oleh pembagian bilangan bulat); subtotalnya sedikitnya Rp 100.000, sehingga jemput-antar gratis; total Rp 108.750.

5. Tomcat menerima permintaan itu dan menyerahkannya kepada DispatcherServlet, yang menemukan `OrderController.store` berdasarkan URL dan metodenya. Jackson membaca badannya ke `CreateOrderRequest`: `customer_id` menjadi `customerId`, `quantity` menjadi `BigDecimal` 2.555, `pickup` tidak ada dan menjadi `false`. `@Valid` lalu memeriksa anotasinya: `@Digits(integer = 3, fraction = 2)` gagal, karena 2.555 memiliki tiga angka desimal; aturan

lainnya lolos, dan `isAddressGivenForPickup()` bernilai benar karena tidak ada jemput-antar. Spring melempar `MethodArgumentNotValidException` sebelum isi metode berjalan, dan menjawab 400 Bad Request dengan badan *problem details*. Tidak ada pesanan yang dibuat. (Bab 12 mengubah jawabannya menjadi 422 dengan objek `errors` bergaya Laravel.)

6. Migrasi: `ddl-auto` Hibernate di bab ini; berbeda dengan migrasi, ia tidak menyimpan riwayat dan tidak dapat memindahkan data, sehingga proyek sungguhan memakai Flyway atau Liquibase. *Seeder*: `CommandLineRunner` `DemoData`; ia berjalan setiap kali aplikasi dimulai, bukan atas permintaan, sehingga lebih dahulu memeriksa apakah datanya sudah ada. *Form request*: *record* dengan anotasi Jakarta Validation dan `@Valid`; pemeriksaan yang memerlukan basis data ditulis di *controller* (Bab 12 mengumpulkannya di satu tempat). *API resource*: *record* respons seperti `OrderView`; nama isiannya berasal dari strategi penamaan, bukan ditulis sebagai kunci larik. `Order::with('customer')`: `@EntityGraph(attributePaths = {"customer"})` pada metode *repository*; ia melekat pada metodenya, bukan dipilih pada setiap panggilan.

Daftar Pustaka

[1] Spring, “Spring Boot 4.0.0 available now,” spring.io blog, Nov. 20, 2025. Accessed: Sep. 25, 2026. [Online]. Available: https://spring.io/blog/2025/11/20/spring-boot-4-0-0-available-now/

[2] Spring, “Spring Boot 4.1.1 available now,” spring.io blog, Aug. 20, 2026. Accessed: Sep. 25, 2026. [Online]. Available: https://spring.io/blog/2026/08/20/spring-boot-4-1-1-available-now/

[3] Spring, “System Requirements,” *Spring Boot 4.1.1 Reference Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.spring.io/spring-boot/system-requirements.html

[4] OpenJDK, “JDK 25,” openjdk.org. Accessed: Sep. 25, 2026. [Online]. Available: https://openjdk.org/projects/jdk/25/

[5] Hibernate, “Hibernate ORM documentation.” Accessed: Sep. 25, 2026. [Online]. Available: https://hibernate.org/orm/documentation/

[6] Spring, “The IoC Container,” *Spring Framework Reference Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.spring.io/spring-framework/reference/core/beans.html

[7] Spring, “Modularizing Spring Boot,” spring.io blog, Oct. 28, 2025. Accessed: Sep. 25, 2026. [Online]. Available: https://spring.io/blog/2025/10/28/modularizing-spring-boot/

[8] Eclipse Foundation, “Jakarta Persistence 3.2,” *Jakarta EE Specifications*. Accessed: Sep. 25, 2026. [Online]. Available: https://jakarta.ee/specifications/persistence/3.2/

[9] Spring, “JPA Query Methods,” *Spring Data JPA Reference Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html

[10] Spring, “Annotated Controllers,” *Spring Framework Reference Documentation: Web on Servlet Stack*. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-controller.html

[11] Eclipse Foundation, “Jakarta Validation 3.1,” *Jakarta EE Specifications*. Accessed: Sep. 25, 2026. [Online]. Available: https://jakarta.ee/specifications/bean-validation/3.1/

[12] Spring, “Spring Initializr.” Accessed: Sep. 25, 2026. [Online]. Available: https://start.spring.io

[13] H2 Database Engine, “H2 Database Engine.” Accessed: Sep. 25, 2026. [Online]. Available: https://www.h2database.com/

[14] Spring, “JSON,” *Spring Boot Reference Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.spring.io/spring-boot/reference/features/json.html

[15] M. Nottingham, E. Wilde, and S. Dalal, “Problem Details for HTTP APIs,” RFC 9457, IETF, Jul. 2023. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9457

Bab 12

Evaluasi dan Pembahasan III: Satu REST API di Laravel dan Spring Boot

Bab 9 sampai Bab 11 membangun dua API untuk *laundry* yang sama. API Laravel di Bab 10 memiliki token masuk, *form request*, *resource*, dan uji; API Spring Boot di Bab 11 memiliki entitas, *repository*, *record*, dan aturan yang sama dalam Java biasa. Aplikasi ponsel Raka seharusnya dapat berbicara dengan salah satunya tanpa perlu tahu siapa yang menjawab. Bab ini memeriksa apakah hal itu benar, menemukan tempat-tempat yang belum benar, lalu memperbaikinya, satu masalah demi satu masalah, di kedua kerangka kerja. Di akhir bab ini Anda akan mampu menulis kontrak API yang cukup tepat untuk diperiksa oleh program; mengubah setiap perbedaan antara dua implementasi menjadi keputusan yang dicatat di kontrak itu; membuat Laravel dan Spring Boot sepakat tentang amplop jawaban, format tanggal, bentuk galat, kode status penolakan, pembayaran, dan laporan harian; serta memeriksa server mana pun terhadap kontrak dengan program PHP kecil yang jalannya dapat Anda ulangi di komputer Anda sendiri.

12.1 Titik Berangkat Kita

Bab 9 memberi aplikasi Laravel migrasi yang mengubah tabel berisi data, *factory* dan *seeder* yang mengisi laptop dengan satu bulan data yang masuk akal, *middleware* `EnsureOrderIsOpen` yang menutup pesanan yang sudah pergi, dan *form request* seperti `StoreOrderRequest` dan `StorePaymentRequest`. Bab 10 menambahkan proses masuk staf, *gate* untuk daftar harga, API di belakang token Sanctum, *resource* yang membentuk JSON-nya, serta uji unit dan uji fitur. Bab 11 membangun toko yang sama di Spring Boot: entitas untuk pelanggan, layanan, pesanan, dan pembayaran; *repository* Spring Data; REST *controller* yang membaca dan menulis *record*; dan `LaundryRules`, yang pemeriksaannya mencetak persis apa yang dicetak pemeriksaan PHP Bab 8.

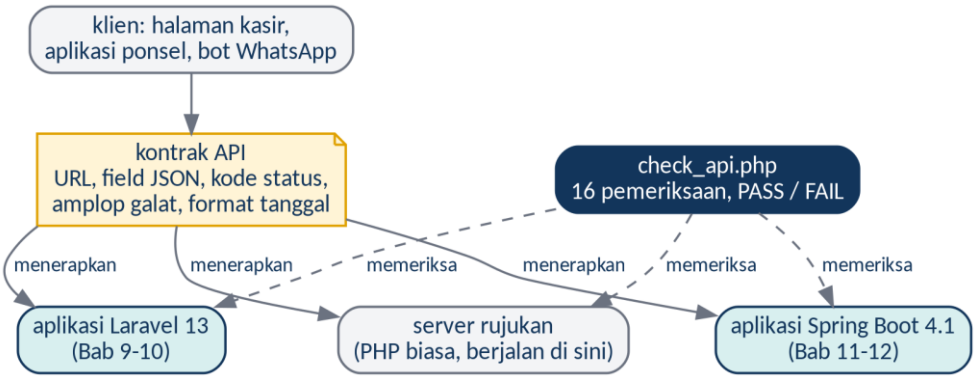
Aturannya sama di keduanya, sampai ke rupiahnya. API-nya tidak. *Resource* Bab 10 membungkus setiap jawaban dalam `data`, sedangkan *controller* Bab 11 mengembalikan objek dan daftar apa adanya. Bab 10 mengirim waktu dengan cara bawaan Laravel menyerialkannya, dalam UTC dengan mikrodetik, sedangkan Bab 11 mengirim tanggal-waktu lokal Java. Laravel menjawab aturan yang dilanggar dengan 422 dan daftar galat per isian; Spring menjawab dengan 400 dan objek *problem details*, atau dengan 409 untuk perubahan status yang ditolak. Spring tidak memiliki *endpoint* pembayaran, dan keduanya tidak memiliki laporan harian untuk program. Tidak satu pun dari ini adalah kekeliruan satu kerangka kerja. Masing-masing adalah keputusan yang belum pernah diambil siapa pun, dan aplikasi ponsel tidak dapat bekerja dengan dua jawaban untuk pertanyaan yang sama.

12.2 Cara Mendekati Masalah API

Masalah API berbeda dari masalah algoritme: bagian yang sulit jarang berupa kodenya, dan hampir selalu berupa kesepakatannya. Setiap masalah di bab ini dikerjakan dengan tujuh langkah yang sama.

1. **Nyatakan aturan kontraknya** dalam satu atau dua kalimat, dengan URL, isian, dan kode status yang tepat.
2. **Kerjakan satu contoh dengan tangan**: satu permintaan dan jawaban tepat yang dituntut kontrak, bait demi bait di bagian yang penting.
3. **Bandingkan kedua implementasi** dengan contoh itu, dan catat setiap perbedaannya.
4. **Putuskan** apa kata kontrak di tempat keduanya berbeda, dan mengapa. Utamakan jawaban yang lebih mudah bagi klien, yang mengikuti standar HTTP, dan yang dapat dihasilkan kerangka kerja lain tanpa melawan dirinya sendiri.
5. **Ubahlah setiap implementasi** sesedikit mungkin agar sesuai dengan keputusan itu.
6. **Periksalah** dengan uji di setiap kerangka kerja dan dengan pemeriksa kontrak di Subbab 12.11.
7. **Perhatikan penyelesaian keliru yang menggoda**, yaitu perubahan yang tampak memecahkan masalah tetapi merusak hal lain, dan lihat mengapa ia gagal.

Gambar 12.1 memperlihatkan susunan yang dibangun langkah-langkah ini. Kontrak berada di tengah; aplikasi Laravel, aplikasi Spring Boot, dan sebuah server rujukan kecil yang ditulis dalam PHP biasa masing-masing menerapkannya; dan satu program pemeriksa memeriksa ketiganya dengan cara yang sama.



Gambar 12.1 Satu kontrak, tiga implementasi, dan satu pemeriksa: klien hanya bergantung pada kontrak, dan `check_api.php` memeriksa setiap server terhadapnya.

Server rujukan itu layak mendapat penjelasan. Lingkungan tempat buku ini ditulis tidak dapat memasang Laravel maupun Spring Boot, sebagaimana dijelaskan `README.md` setiap bab, tetapi dapat menjalankan PHP. Maka `contract/reference/server.php` bab ini menerapkan seluruh kontrak dalam sekitar 240 baris PHP biasa dengan SQLite, memakai berkas aturan Bab 8 itu sendiri, dan keluaran pemeriksa di Subbab 12.11 adalah keluaran sesungguhnya dari jalannya terhadap server itu. Di komputer Anda sendiri, pemeriksa yang sama dijalankan terhadap aplikasi Laravel dan Spring.

12.3 Kontraknya

Tabel 12.1 mendaftar *endpoint* kontrak. Setiap URL diawali `/api`; setiap badan permintaan dan jawaban berupa JSON; setiap jawaban yang membawa satu *resource* atau satu daftar membungkusnya dalam kunci tingkat atas `data`.

Tabel 12.1 *Endpoint* kontrak API Profitina Laundry; tanda tanya menandai isian yang opsional.

Metode dan URL	Badan	Berhasil
`GET /api/services`		200, `data`: layanan aktif, menurut kode
`POST /api/customers`	`name`, `phone`, `address`?	201, `data`: pelanggannya
`GET /api/orders?status=&page=`		200, `data`: pesanan, `meta`: angka halaman
`POST /api/orders`	`customer_id`, `service_id`, `quantity`, `pickup`, `delivery_address`?, `note`?	201, `data`: pesanannya
`GET /api/orders/{id}`		200, `data`: pesanan beserta `payments`
`PATCH /api/orders/{id}/status`	`status`	200, `data`: pesanannya
`POST /api/orders/{id}/payments`	`amount`, `method`	201, `data`: pesanan beserta `payments`
`GET /api/reports/daily?date=YYYY-MM-DD`		200, `data`: angka-angka hari itu

Aturan tentang isi jawaban dibuat sesingkat mungkin, agar dapat ditaati.

- **Nama.** Nama isian berbentuk snake_case (`delivery_address`). Nilai status dan metode pembayaran berupa kata berhuruf kecil (`queued`, `finished`, `departed`; `cash`, `qris`, `transfer`).
- **Angka.** Uang adalah bilangan bulat dalam rupiah. Jumlah berupa angka, bukan teks, dengan paling banyak dua angka desimal.
- **Waktu.** Setiap tanggal-waktu adalah waktu lokal toko, ditulis `YYYY-MM-DDTHH:MM:SS`, misalnya `2026-09-25T08:15:00`, dan waktu yang belum terjadi bernilai `null`, tidak pernah hilang.
- **Sebuah pesanan** membawa sedikitnya `id`, `customer` (`id`, `name`, `phone`), `service` (`id`, `name`, `unit`, `price`), `quantity`, `pickup`, `delivery_address`, `total`, `paid`, `balance`, `status`, `note`, `received_at`, `finished_at`, dan `departed_at`, dengan sisa tagihan yang didefinisikan oleh Persamaan (12.1). Klien harus mengabaikan isian yang tidak dikenalnya, sehingga implementasi boleh mengirim lebih, seperti `outlet_id` milik Laravel.

$$balance = total - \sum_{p \in P} amount_p \tag{12.1}$$

Di sini P adalah himpunan pembayaran yang tercatat untuk pesanan itu dan $amount_p$ adalah jumlah pembayaran p. Karena tidak ada pembayaran yang boleh melebihi sisa tagihan (Masalah 12.6), sisa tagihan selalu berada di antara 0 dan totalnya.

Penolakan mengikuti **Tabel 12.2**. Badan setiap penolakan memiliki `message`; badan 422 juga memiliki `errors`, sebuah objek yang kuncinya adalah nama isian permintaan dan nilainya adalah daftar pesan. Inilah bentuk yang sudah dihasilkan validasi Laravel, dan alasan memilihnya sederhana: klien dapat menampilkan setiap pesan di bawah isian yang bersangkutan.

Tabel 12.2 Penolakan dalam kontrak, dan badan yang dibawa masing-masing.

Status	Kapan	Badan
401	tidak ada token yang sah (Laravel; Spring mulai Bab 14)	`message`
404	pesanan, layanan, atau pelanggannya tidak ada	`message`
409	pesanan sudah pergi dan ditutup	`message`
422	masukan melanggar aturan, termasuk langkah status yang ditolak	`message`, `errors`

12.4 Masalah 12.1: Daftar Harga untuk Program

12.4.1 Masalahnya

`GET /api/services` menjawab 200 dengan `{"data": [...]}`: setiap layanan aktif, diurutkan menurut kode, masing-masing dengan `id`, `name`, `unit`, dan `price`.

12.4.2 Memahaminya dengan tangan

Baris pertama `Service.csv` adalah `S0001,"Wash, dry & fold (per kg)",8000.00`. Maka unsur pertama `data` menurut kontrak adalah `{"id": "S0001", "name": "Wash, dry & fold", "unit": "kg", "price": 8000}`: nama tanpa bagian dalam kurung, satuan yang diambil dari bagian itu, dan harga sebagai bilangan bulat.

12.4.3 Membandingkan keduanya

API Laravel Bab 10 sama sekali tidak memiliki *endpoint* layanan; `ServiceResource`-nya ada, tetapi hanya di dalam pesanan. API Spring Bab 11 memilikinya, tetapi menjawab dengan daftar polos `[...]`, bukan `{"data": [...]}`. Keputusannya: *endpoint* ini ada di keduanya, dan membungkus daftarnya dalam `data`, seperti setiap jawaban lain dalam kontrak.

12.4.4 Penyelesaian di kedua kerangka kerja

Di Laravel, **Kode 12.1** adalah *controller* dua metode yang memakai ulang *resource* Bab 10; `ServiceResource::collection` membungkus daftarnya dalam `data` dengan sendirinya.

Kode 12.1 `Api/ServiceController.php` (Laravel): daftar harga yang aktif sebagai koleksi *resource*.

```php
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use App\Http\Resources\ServiceResource;
use App\Models\Service;

class ServiceController extends Controller
{
    // GET /api/services: the active price list, S0001 first.
    public function index()
    {
        return ServiceResource::collection(
            Service::where('is_active', true)->orderBy('id')->get());
    }

    public function show(Service $service)
    {
        return new ServiceResource($service);
    }
}
```

Kode 12.2 adalah berkas rute API bab ini: rute-rute Bab 10 ditambah layanan dan laporan harian dari Masalah 12.7.

Kode 12.2 `routes/api.php` (Laravel): rute Bab 10, daftar harga, dan laporan harian.

```php
<?php

use App\Http\Controllers\Api\CustomerController;
use App\Http\Controllers\Api\OrderController;
use App\Http\Controllers\Api\PaymentController;
use App\Http\Controllers\Api\ReportController;
use App\Http\Controllers\Api\ServiceController;
use App\Http\Controllers\Api\TokenController;
use Illuminate\Support\Facades\Route;

// Every URL here starts with /api (Laravel adds the prefix).
Route::post('tokens', [TokenController::class, 'store'])
    ->middleware('throttle:5,1');

Route::middleware('auth:sanctum')->group(function () {
    Route::delete('tokens/current', [TokenController::class, 'destroy']);

    Route::get('services', [ServiceController::class, 'index']);
    Route::get('services/{service}', [ServiceController::class, 'show']);

    Route::get('customers', [CustomerController::class, 'index']);
    Route::post('customers', [CustomerController::class, 'store']);

    Route::get('orders', [OrderController::class, 'index']);
    Route::post('orders', [OrderController::class, 'store']);
    Route::get('orders/{order}', [OrderController::class, 'show']);

    Route::middleware('order.open')->group(function () {
        Route::patch('orders/{order}/status',
            [OrderController::class, 'updateStatus']);
        Route::post('orders/{order}/payments',
            [PaymentController::class, 'store']);
    });

    Route::get('reports/daily', [ReportController::class, 'daily']);
});
```

Di Spring tidak ada pembungkus otomatis, sehingga **Kode 12.3** menulisnya sendiri: *record* generik dengan satu komponen, yang ditulis Jackson sebagai `{"data": ...}`.

Kode 12.3 `DataView.java` (Spring): amplop kontrak dalam satu baris.

```
package com.profitina.laundry.web;

/** The contract wraps every answer in {"data": ...}, as Laravel does. */
public record DataView<T>(T data) {
}
```

Kode 12.4 memakainya; satu-satunya perubahan dari Bab 11 adalah tipe kembalian dan `new DataView<>(...)` di sekeliling hasilnya.

Kode 12.4 `ServiceController.java` (Spring): daftar yang sama, kini di dalam amplop.

```
package com.profitina.laundry.web;

import com.profitina.laundry.repository.ServiceRepository;
import java.util.List;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.server.ResponseStatusException;

@RestController
@RequestMapping("/api/services")
public class ServiceController {

    private final ServiceRepository services;

    public ServiceController(ServiceRepository services) {
        this.services = services;
    }

    @GetMapping
    public DataView<List<ServiceView>> index() {
        return new DataView<>(services.findByActiveTrueOrderByIdAsc()
                .stream().map(ServiceView::of).toList());
    }

    @GetMapping("/{id}")
    public DataView<ServiceView> show(@PathVariable String id) {
        return new DataView<>(services.findById(id).map(ServiceView::of)
                .orElseThrow(() -> new ResponseStatusException(
                        HttpStatus.NOT_FOUND, "No service " + id + ".")));
    }
}
```

`CustomerController` dan `OrderController` pada aplikasi Spring membungkus jawabannya dalam `DataView` dengan cara yang sama; `PageView` tidak memerlukan pembungkus, karena sudah memiliki `data` dan `meta`.

12.4.5 Penyelesaian keliru yang menggoda

Mengembalikan modelnya sendiri, `Service::all()` di Laravel atau `services.findAll()` di Spring, menghasilkan jawaban hanya dengan beberapa karakter. Cara itu juga mengirim `description` dan `is_active` (dan di Laravel juga `created_at` serta `updated_at`), menghilangkan `name` dan `unit` yang merupakan hasil hitungan, dan mengikat setiap aplikasi ponsel pada kolom-

kolom basis data hari ini. Pada hari pemilik memecah deskripsi menjadi dua kolom, setiap klien rusak. Jawaban adalah janji; ia harus ditulis dengan sengaja, dalam *resource* atau *record*.

12.5 Masalah 12.2: Satu Cara Menulis Waktu

12.5.1 Masalahnya

Setiap tanggal-waktu dalam jawaban adalah waktu lokal toko, ditulis `YYYY-MM-DDTHH:MM:SS`, dan waktu yang belum terjadi bernilai `null`.

12.5.2 Memahaminya dengan tangan

Pesanan Sari diterima pada 25 September 2026 pukul 08.15 menurut jam di meja kasir di Surabaya. `received_at` menurut kontrak adalah `2026-09-25T08:15:00`. Waktu Indonesia Barat tujuh jam lebih awal dari UTC, sehingga saat yang sama dalam UTC adalah pukul 01.15.

12.5.3 Membandingkan keduanya

`OrderResource` Bab 10 meneruskan objek Carbon ke penyandi JSON, yang menulis bawaan Laravel: ISO 8601 dalam UTC dengan mikrodetik, `2026-09-25T01:15:00.000000Z` (Latihan 10.6). `OrderView` Bab 11 meneruskan `LocalDateTime` ke Jackson, yang menulis waktu lokal tanpa zona; untuk waktu dengan detik nol dan tanpa pecahan, `toString()` milik Java sendiri bahkan akan menulis `2026-09-25T08:15`, tanpa detik. Semuanya ISO 8601 yang sah, dan semuanya menggambarkan saat yang sama, tetapi klien yang membandingkan *string*, atau manusia yang membacanya, melihat tiga nilai yang berbeda.

Keputusannya adalah waktu lokal tanpa zona, dengan detik, dan tidak lebih. Hari toko, laporannya, dan struknya semuanya hidup dalam waktu Surabaya; klien di toko tidak pernah perlu mengonversi. Sistem yang mencakup beberapa zona waktu akan memilih UTC dengan *offset* yang eksplisit; yang penting adalah kontraknya menyatakan yang mana.

12.5.4 Penyelesaian di kedua kerangka kerja

Resource Laravel memformat setiap waktu sendiri. **Kode 12.5** mendefinisikan formatnya sekali, sebagai konstanta, dan memakai operator *null-safe* `?->` sehingga waktu yang belum ada menjadi `null`. Zona waktu aplikasi harus `Asia/Jakarta` di `config/app.php`, sebagaimana diminta `README.md` setiap bab, agar objek Carbon sudah berada dalam waktu toko sebelum diformat.

Kode 12.5 `OrderResource.php` (Laravel): setiap waktu dalam format kontrak.

```php
<?php

namespace App\Http\Resources;

use Illuminate\Http\Request;
use Illuminate\Http\Resources\Json\JsonResource;

class OrderResource extends JsonResource
{
    // The contract's one date-time format: 2026-09-25T08:15:00, local time.
    public const TIME = 'Y-m-d\TH:i:s';

    public function toArray(Request $request): array
    {
        return [
            'id' => $this->id,
            'customer' => new CustomerResource($this->whenLoaded('customer')),
            'service' => new ServiceResource($this->whenLoaded('service')),
            'outlet_id' => $this->outlet_id,
            'quantity' => (float) $this->quantity,
            'pickup' => $this->pickup,
            'delivery_address' => $this->delivery_address,
            'total' => $this->total,
            'paid' => $this->paid(),
            'balance' => $this->balance(),
            'status' => $this->status->value,
            'note' => $this->note,
            'received_at' => $this->received_at->format(self::TIME),
            'finished_at' => $this->finished_at?->format(self::TIME),
            'departed_at' => $this->departed_at?->format(self::TIME),
            'payments' => PaymentResource::collection(
                $this->whenLoaded('payments')),
        ];
    }
}
```

`PaymentResource` memakai konstanta yang sama untuk `paid_at`. Di Spring, format ditetapkan per isian dengan `@JsonFormat` milik Jackson, seperti diperlihatkan **Kode 12.6**; *record* yang sama juga mendapat daftar pembayaran yang diperlukan Masalah 12.6.

Kode 12.6 `OrderView.java` (Spring): waktu dengan pola tetap, dan pembayaran hanya bila diminta.

```java
package com.profitina.laundry.web;

import com.fasterxml.jackson.annotation.JsonFormat;
import com.fasterxml.jackson.annotation.JsonInclude;
import com.profitina.laundry.domain.Order;
import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.util.List;

/** The JSON of one order, exactly as the contract of Chapter 12 says. */
public record OrderView(
        Long id,
        CustomerView customer,
        ServiceView service,
        BigDecimal quantity,
        boolean pickup,
        String deliveryAddress,
        long total,
        long paid,
        long balance,
        String status,
        String note,
        @JsonFormat(pattern = TIME) LocalDateTime receivedAt,
        @JsonFormat(pattern = TIME) LocalDateTime finishedAt,
        @JsonFormat(pattern = TIME) LocalDateTime departedAt,
        @JsonInclude(JsonInclude.Include.NON_NULL)
        List<PaymentView> payments) {

    /** The contract's one date-time format: 2026-09-25T08:15:00. */
    static final String TIME = "yyyy-MM-dd'T'HH:mm:ss";

    /** Without the payment list, like Laravel when it is not loaded. */
    public static OrderView of(Order o) {
        return build(o, null);
    }

    public static OrderView withPayments(Order o) {
        return build(o, o.getPayments().stream().map(PaymentView::of)
                .toList());
    }
```

Kode 12.6 (Lanjutan) *OrderView.java (Spring): waktu dengan pola tetap, dan pembayaran hanya bila diminta.*

```java
    private static OrderView build(Order o, List<PaymentView> payments) {
        return new OrderView(o.getId(), CustomerView.of(o.getCustomer()),
                ServiceView.of(o.getService()), o.getQuantity(),
                o.isPickup(), o.getDeliveryAddress(), o.getTotal(), o.paid(),
                o.balance(), o.getStatus().value(), o.getNote(),
                o.getReceivedAt(), o.getFinishedAt(), o.getDepartedAt(),
                payments);
    }
}
```

`'T'` dalam pola itu diberi tanda kutip karena ia huruf harfiah, bukan kode untuk isian tanggal. `TIME` adalah konstanta *record* agar `PaymentView` dapat memakai pola yang sama. Karena *controller* memotong setiap waktu menjadi detik utuh sebelum disimpan, tidak ada pecahan detik yang dapat muncul.

12.5.5 Penyelesaian keliru yang menggoda

Mengubah zona waktu di `config/app.php` Laravel menjadi UTC “agar sama dengan Spring” tampak menyelaraskan keduanya, padahal justru memperburuk segalanya: laporan di Masalah

12.7 lalu akan memotong hari pada pukul 07.00 waktu Surabaya, dan setiap pesanan yang diterima antara tengah malam dan pukul tujuh pagi akan terhitung pada hari yang salah. Zona waktu dipilih untuk manusia, dan format dipilih untuk program; jangan mengubah yang pertama untuk memperbaiki yang kedua.

12.6 Masalah 12.3: Satu Bentuk untuk Setiap Galat

12.6.1 Masalahnya

Permintaan yang melanggar aturan dijawab dengan 422 dan `{"message": "...", "errors": {"field": ["..."]}}`, dengan nama isian yang sama dengan nama di permintaan, dalam snake_case. Penolakan lainnya sedikitnya memiliki `message`.

12.6.2 Memahaminya dengan tangan

`POST /api/customers` dengan `{"name": "S", "phone": "12345"}` harus dijawab dengan 422 dan objek `errors` dengan kunci `name` (terlalu pendek) dan `phone` (bukan nomor seluler). Urutan kuncinya tidak penting; namanya yang penting.

12.6.3 Membandingkan keduanya

Laravel sudah menjawab dengan cara itu: itulah format kerangka kerja itu sendiri untuk galat validasi. Aplikasi Spring Bab 11, dengan *problem details* dinyalakan, menjawab 400 dengan badan *problem details* seperti `{"type": "about:blank", "title": "Bad Request", "status": 400, "detail": "Invalid request content."}`. Keterangan tentang isian mana yang gagal sama sekali tidak ada di badannya, dan kalaupun ada, isian itu akan disebut dengan nama Java-nya, `serviceId`, bukan `service_id`. `ResponseStatusException` miliknya juga dijawab sebagai *problem details*, dengan alasannya di `detail`, bukan di `message`.

Keputusannya mengikuti Laravel. Bentuknya membawa keterangan yang diperlukan klien, dan juga bentuk yang sudah diharapkan sebagian besar klien PHP dan JavaScript. Standar HTTP mendefinisikan 422 Unprocessable Content tepat untuk kasus ini, yaitu permintaan yang bentuknya benar tetapi isinya tidak dapat diproses [1]; 400 tetap dipakai untuk badan yang sama sekali bukan JSON yang sah.

12.6.4 Penyelesaian di Spring

Dua kelas mengerjakannya. **Kode 12.7** adalah *exception* yang membawa peta galat, untuk aturan yang tidak dapat diungkapkan dengan anotasi, seperti “telepon ini sudah terdaftar”.

Kode 12.7 `ApiValidationException.java` (Spring): aturan yang dilanggar, beserta pesan per isian.

```java
package com.profitina.laundry.web;

import java.util.List;
import java.util.Map;

/** Input that breaks a rule: becomes 422 with {"errors": {...}}. */
public class ApiValidationException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    private final transient Map<String, List<String>> errors;

    public ApiValidationException(Map<String, List<String>> errors) {
        super("The given data was invalid.");
        this.errors = errors;
    }

    public ApiValidationException(String field, String message) {
        this(Map.of(field, List.of(message)));
    }

    public Map<String, List<String>> getErrors() {
        return errors;
    }
}
```

Kode 12.8 mengubah setiap galat menjadi bentuk kontrak. Kelas yang ditandai `@RestControllerAdvice` menerapkan metode `@ExceptionHandler`-nya pada *exception* yang dilempar *controller* mana pun, sebelum penanganan bawaan Spring [2].

Kode 12.8 `ApiExceptionHandler.java` (Spring): galat validasi, aturan yang dilanggar, *exception* status, dan badan yang tidak terbaca, masing-masing dalam bentuk kontrak.

```java
package com.profitina.laundry.web;

import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.http.converter.HttpMessageNotReadableException;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import org.springframework.web.server.ResponseStatusException;

/** Every error in the contract's shape: {"message": ..., "errors": ...}. */
@RestControllerAdvice
public class ApiExceptionHandler {

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Map<String, Object>> invalid(
            MethodArgumentNotValidException e) {
        Map<String, List<String>> errors = new LinkedHashMap<>();
        for (FieldError f : e.getBindingResult().getFieldErrors()) {
            errors.computeIfAbsent(snake(f.getField()),
                    k -> new ArrayList<>()).add(f.getDefaultMessage());
        }
        return unprocessable(errors);
    }

    @ExceptionHandler(ApiValidationException.class)
    public ResponseEntity<Map<String, Object>> rule(
            ApiValidationException e) {
        return unprocessable(e.getErrors());
    }

    @ExceptionHandler(ResponseStatusException.class)
    public ResponseEntity<Map<String, Object>> status(
            ResponseStatusException e) {
        String reason = e.getReason() == null ? "" : e.getReason();
        return ResponseEntity.status(e.getStatusCode())
                .body(Map.of("message", reason));
    }
```

Kode 12.8 (Lanjutan) *ApiExceptionHandler.java (Spring): galat validasi, aturan yang dilanggar, exception status, dan badan yang tidak terbaca, masing-masing dalam bentuk kontrak.*

```
    @ExceptionHandler(HttpMessageNotReadableException.class)
    public ResponseEntity<Map<String, Object>> unreadable(
            HttpMessageNotReadableException e) {
        return ResponseEntity.badRequest().body(Map.of("message",
                "The request body is not valid JSON for this endpoint."));
    }

    private static ResponseEntity<Map<String, Object>> unprocessable(
            Map<String, List<String>> errors) {
        Map<String, Object> body = new LinkedHashMap<>();
        body.put("message", "The given data was invalid.");
        body.put("errors", errors);
        return ResponseEntity.status(HttpStatus.UNPROCESSABLE_CONTENT)
                .body(body);
    }

    /** customerId -> customer_id: the name the client sent. */
    static String snake(String name) {
        return name.replaceAll("([a-z0-9])([A-Z])", "$1_$2").toLowerCase();
    }
}
```

`invalid` menangani *exception* yang dilempar `@Valid`. Metode ini menelusuri galat isian, mengubah setiap nama isian menjadi snake_case dengan ekspresi reguler di `snake`, yang menyisipkan garis bawah di antara huruf kecil atau angka dan huruf kapital sesudahnya, lalu mengumpulkan pesan per isian dalam `LinkedHashMap`, yang mempertahankan urutan ditemukannya. `rule` menangani *exception* kita sendiri, `status` memberi setiap `ResponseStatusException` badan dengan alasannya sebagai `message`, dan `unreadable` menjawab badan yang bukan JSON dengan 400. `unprocessable` membangun badan 422 sekali untuk kedua jalur validasi. Karena *advice* ini menangani *exception* itu sendiri, pengaturan *problem details* Bab 11 tidak lagi membentuk jawaban-jawaban ini; pengaturan itu tetap menyala untuk galat yang tidak ditangani kelas ini.

Controller pelanggan kini menjawab telepon ganda dengan `new ApiValidationException("phone", "The phone has already been taken.")`, pesan yang sama dengan aturan `unique` di Laravel, alih-alih 409 seperti di Bab 11.

12.6.5 Penyelesaian keliru yang menggoda

Jalan pintas yang menggoda adalah mengatur Jackson agar menulis nama Java dalam snake_case (seperti yang dilakukan Bab 11) dan berharap nama isian pada galat ikut berubah. Ternyata tidak: nama dalam `FieldError` adalah jalur properti Java yang dihasilkan validator, bukan oleh Jackson, dan tetap berbunyi `serviceId`. Konversinya harus terjadi di tempat badan galat dibangun.

12.7 Masalah 12.4: Semua Galat Sekaligus

12.7.1 Masalahnya

`POST /api/orders` dengan beberapa aturan yang dilanggar menjawab 422 dengan semuanya di `errors` sekaligus, termasuk aturan yang melibatkan dua isian: bila `pickup` bernilai benar, `delivery_address` wajib diisi.

12.7.2 Memahaminya dengan tangan

`{"quantity": 0, "pickup": true}` melanggar empat aturan: tidak ada pelanggan, tidak ada layanan, jumlah di bawah 0,1, dan jemput-antar tanpa alamat. Kontrak mewajibkan `errors` memuat sedikitnya `customer_id`, `service_id`, `quantity`, dan `delivery_address`. (Laravel juga menyebut `outlet_id`, isian wajib tambahannya; klien mengabaikan kunci yang tidak dikenalnya.)

12.7.3 Membandingkan keduanya

`StoreOrderRequest` Laravel memeriksa semua aturannya bersama-sama, termasuk `required_if_accepted:pickup`, dan melaporkan setiap kegagalan: ia sudah menaati kontrak. Versi Spring Bab 11 gagal dengan dua cara. Aturan dua isiannya berupa metode `@AssertTrue`, `isAddressGivenForPickup()`, dan validator melaporkannya dengan nama properti `addressGivenForPickup`, yang menjadi `address_given_for_pickup`, kunci yang tidak dikenal klien mana pun. Dan pemeriksaannya yang memerlukan basis data, "pelanggannya ada" dan "layanannya ditawarkan", berjalan di *controller* hanya setelah `@Valid` lolos, sehingga permintaan dengan kedua jenis kesalahan menerima galatnya dalam dua putaran.

12.7.4 Penyelesaian di Spring

Kode 12.9 membuang metode `@AssertTrue`; *record* itu hanya menyimpan aturan yang menyangkut satu isian.

Kode 12.9 `CreateOrderRequest.java` (Spring): hanya aturan satu isian.

```
package com.profitina.laundry.web;

import jakarta.validation.constraints.DecimalMax;
import jakarta.validation.constraints.DecimalMin;
import jakarta.validation.constraints.Digits;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Size;
import java.math.BigDecimal;

/** The pickup-needs-an-address rule now lives in OrderController. */
public record CreateOrderRequest(
        @NotNull Long customerId,
        @NotBlank String serviceId,
        @NotNull @DecimalMin("0.1") @DecimalMax("100")
        @Digits(integer = 3, fraction = 2) BigDecimal quantity,
        boolean pickup,
        @Size(max = 200) String deliveryAddress,
        @Size(max = 200) String note) {
}
```

Metode `store` pada *controller*, bagian pertama **Kode 12.10**, tidak lagi memasang `@Valid` pada parameternya. Metode ini meminta `Validator` Jakarta, yang disuntikkan melalui konstruktor, untuk memvalidasi *record*, mengubah jalur properti setiap pelanggaran menjadi kunci snake_case, lalu menambahkan pemeriksaannya sendiri ke peta yang sama: pelanggan, layanan, dan alamat pengantaran. Hanya bila petanya masih kosong ia membuat pesanan; bila tidak, ia melempar satu `ApiValidationException` yang memuat semua galat.

Kode 12.10 `OrderController.java` (Spring): semua galat sekaligus, alur kerja sebagai 422, pesanan tertutup sebagai 409, dan pembayaran.

```java
package com.profitina.laundry.web;

import com.profitina.laundry.domain.Customer;
import com.profitina.laundry.domain.LaundryRules;
import com.profitina.laundry.domain.Order;
import com.profitina.laundry.domain.OrderStatus;
import com.profitina.laundry.domain.PaymentMethod;
import com.profitina.laundry.domain.Service;
import com.profitina.laundry.repository.CustomerRepository;
import com.profitina.laundry.repository.OrderRepository;
import com.profitina.laundry.repository.PaymentRepository;
import com.profitina.laundry.repository.ServiceRepository;
import jakarta.validation.ConstraintViolation;
import jakarta.validation.Valid;
import jakarta.validation.Validator;
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.http.HttpStatus;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PatchMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.server.ResponseStatusException;

@RestController
@RequestMapping("/api/orders")
public class OrderController {

    private final OrderRepository orders;
    private final CustomerRepository customers;
    private final ServiceRepository services;
    private final PaymentRepository payments;
    private final Validator validator;
```

Kode 12.10 (Lanjutan) *OrderController.java (Spring): semua galat sekaligus, alur kerja sebagai 422, pesanan tertutup sebagai 409, dan pembayaran.*

```
    public OrderController(OrderRepository orders,
                           CustomerRepository customers,
                           ServiceRepository services,
                           PaymentRepository payments, Validator validator) {
        this.orders = orders;
        this.customers = customers;
        this.services = services;
        this.payments = payments;
        this.validator = validator;
    }

    @GetMapping
    @Transactional(readOnly = true)
    public PageView<OrderView> index(
            @RequestParam(required = false) String status,
            @RequestParam(defaultValue = "1") int page) {
        PageRequest pageable = PageRequest.of(Math.max(page, 1) - 1, 20);
        Page<Order> result = status == null
                ? orders.findAllByOrderByReceivedAtDescIdDesc(pageable)
                : orders.findByStatusOrderByReceivedAtDescIdDesc(
                        parseStatus(status), pageable);
        return PageView.of(result, OrderView::of);
    }

    @GetMapping("/{id}")
    @Transactional(readOnly = true)
    public DataView<OrderView> show(@PathVariable Long id) {
        return new DataView<>(OrderView.withPayments(find(id)));
    }

    /** Every broken rule is reported at once, as Laravel does. */
    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    @Transactional
    public DataView<OrderView> store(@RequestBody CreateOrderRequest in) {
        Map<String, List<String>> errors = new LinkedHashMap<>();
        for (ConstraintViolation<CreateOrderRequest> v
                : validator.validate(in)) {
            add(errors, ApiExceptionHandler.snake(
                    v.getPropertyPath().toString()), v.getMessage());
        }
        Customer customer = in.customerId() == null ? null
                : customers.findById(in.customerId()).orElse(null);
        if (in.customerId() != null && customer == null) {
            add(errors, "customer_id", "The selected customer is invalid.");
        }
```

Kode 12.10 (Lanjutan) *OrderController.java (Spring): semua galat sekaligus, alur kerja sebagai 422, pesanan tertutup sebagai 409, dan pembayaran.*

```
        Service service = in.serviceId() == null ? null
                : services.findById(in.serviceId())
                        .filter(Service::isActive).orElse(null);
        if (in.serviceId() != null && service == null) {
            add(errors, "service_id", "The selected service is invalid.");
        }
        if (in.pickup() && (in.deliveryAddress() == null
                || in.deliveryAddress().isBlank())) {
            add(errors, "delivery_address",
                    "Where should the laundry be delivered?");
        }
        if (!errors.isEmpty()) {
            throw new ApiValidationException(errors);
        }
        Order order = new Order(customer, service, in.quantity(),
                in.pickup(), in.deliveryAddress(), in.note(), now());
        return new DataView<>(OrderView.of(orders.save(order)));
    }

    @PatchMapping("/{id}/status")
    @Transactional
    public DataView<OrderView> updateStatus(@PathVariable Long id,
            @Valid @RequestBody StatusRequest in) {
        Order order = find(id);
        ensureOpen(order);
        OrderStatus wanted = parseStatus(in.status());
        OrderStatus next;
        try {
            next = LaundryRules.advance(order.getStatus(), order.balance());
        } catch (IllegalStateException e) {
            throw new ApiValidationException("status", e.getMessage());
        }
        if (wanted != next) {
            throw new ApiValidationException("status", "An order that is "
                    + order.getStatus().value() + " can only become "
                    + next.value() + ".");
        }
        order.advance(now());
        return new DataView<>(OrderView.of(order));
    }
```

Kode 12.10 (Lanjutan) *OrderController.java (Spring): semua galat sekaligus, alur kerja sebagai 422, pesanan tertutup sebagai 409, dan pembayaran.*

```
    @PostMapping("/{id}/payments")
    @ResponseStatus(HttpStatus.CREATED)
    @Transactional
    public DataView<OrderView> pay(@PathVariable Long id,
            @Valid @RequestBody CreatePaymentRequest in) {
        Order order = find(id);
        ensureOpen(order);
        PaymentMethod method;
        try {
            method = PaymentMethod.fromValue(in.method());
        } catch (IllegalArgumentException e) {
            throw new ApiValidationException("method",
                    "The selected method is invalid.");
        }
        if (in.amount() > order.balance()) {
            throw new ApiValidationException("amount", "Only "
                    + LaundryRules.rupiah(order.balance())
                    + " is still owed.");
        }
        payments.save(order.addPayment(in.amount(), method, now()));
        return new DataView<>(OrderView.withPayments(order));
    }

    private Order find(Long id) {
        return orders.findWithDetailsById(id)
                .orElseThrow(() -> new ResponseStatusException(
                        HttpStatus.NOT_FOUND, "No order " + id + "."));
    }

    /** The job of Laravel's EnsureOrderIsOpen middleware. */
    private static void ensureOpen(Order order) {
        if (order.getStatus() == OrderStatus.DEPARTED) {
            throw new ResponseStatusException(HttpStatus.CONFLICT, "Order #"
                    + order.getId() + " has left the shop and is closed.");
        }
    }

    private static OrderStatus parseStatus(String value) {
        try {
            return OrderStatus.fromValue(value);
        } catch (IllegalArgumentException e) {
            throw new ApiValidationException("status",
                    "The selected status is invalid.");
        }
    }
```

Kode 12.10 (Lanjutan) *OrderController.java (Spring): semua galat sekaligus, alur kerja sebagai 422, pesanan tertutup sebagai 409, dan pembayaran.*

```
    private static void add(Map<String, List<String>> errors, String field,
                            String message) {
        errors.computeIfAbsent(field, k -> new ArrayList<>()).add(message);
    }

    private static LocalDateTime now() {
        return LocalDateTime.now().truncatedTo(ChronoUnit.SECONDS);
    }
}
```

Pendekatan yang sama adalah cara kerja *form request* Laravel di dalamnya: aturan lebih dahulu, lalu *hook* `after`, lalu satu daftar galat. Menuliskannya dengan tangan di Spring memerlukan sekitar lima belas baris, dan pengetahuannya tetap berada di satu metode.

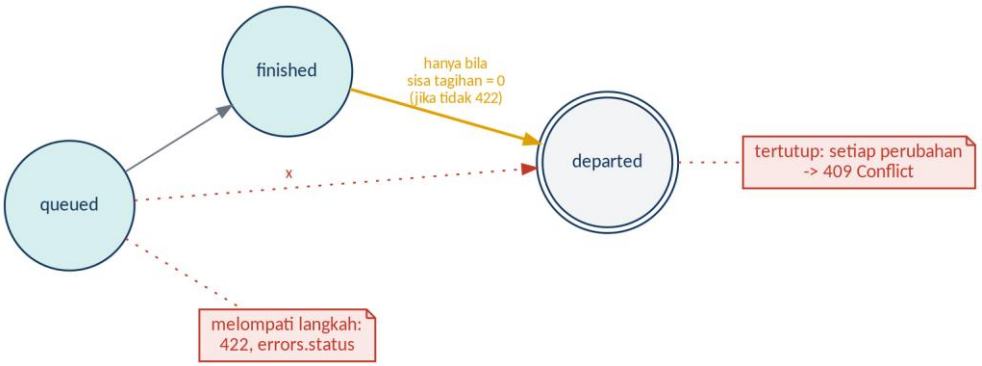
12.7.5 Penyelesaian keliru yang menggoda

Mempertahankan `@Valid` dan mengganti nama metode `@AssertTrue` menjadi `isDeliveryAddress()` agar propertinya menjadi `deliveryAddress` tampak cerdik. Cara itu bertabrakan dengan komponen `deliveryAddress` yang sesungguhnya, membingungkan siapa pun yang membaca *record*-nya, dan tetap menyisakan pemeriksaan basis data di putaran kedua. Aturan tentang dua isian dan aturan yang memerlukan basis data sama-sama tempatnya di kode yang dapat melihat semuanya, dan melapor ke daftar yang sama.

12.8 Masalah 12.5: Memajukan Pesanan, dan Pesanan yang Tertutup

12.8.1 Masalahnya

`PATCH /api/orders/{id}/status` dengan `{"status": "..."}` memajukan pesanan satu langkah menurut alur kerja di **Gambar 12.2**. Langkah yang tidak diizinkan alur kerja, yaitu langkah yang dilompati atau kepergian dengan tagihan tersisa, dijawab dengan 422 dan `errors.status`. Setiap perubahan pada pesanan yang sudah pergi, baik perubahan status maupun pembayaran, dijawab dengan 409 dan sebuah `message`.



Gambar 12.2 Alur kerja pesanan sebagaimana dinyatakan kontrak: satu langkah setiap kali, pergi hanya bila sisa tagihan 0, dan pesanan yang sudah pergi tertutup bagi setiap perubahan.

12.8.2 Memahaminya dengan tangan

Sebuah pesanan baru seharga Rp 34.000 berstatus *queued*. `departed` melompati `finished`: 422. `finished`: 200, dan `finished_at` mendapat waktu saat itu. `departed` dengan tagihan Rp 34.000: 422, dengan pesan alur kerja "Collect Rp 34.000 before handing over the laundry." Setelah pembayaran Rp 34.000, `departed`: 200. Lalu pembayaran Rp 1.000: 409, karena pesanannya sudah ditutup.

12.8.3 Membandingkan keduanya

API Laravel sudah menaati hal ini: `UpdateOrderStatusRequest` mengubah penolakan alur kerja menjadi galat 422 pada `status`, dan *middleware* `order.open` Bab 9 menjawab 409 untuk pesanan yang sudah pergi. API Spring Bab 11 menjawab setiap langkah yang ditolak dengan 409, termasuk langkah yang dilompati dan kepergian yang belum lunas.

Kedua kode status itu berarti hal yang berbeda, dan kontrak harus memisahkannya. 422 berkata "permintaan Anda dipahami tetapi melanggar aturan tentang isinya": klien dapat memperbaiki permintaannya, misalnya dengan menagih pembayaran lebih dahulu. 409 berkata "sumber daya ini berada dalam keadaan yang bertentangan dengan permintaan semacam itu": tidak ada yang

dapat dikirim klien untuk mengubah pesanan yang sudah pergi [1]. Aplikasi ponsel lalu dapat menampilkan galat 422 di samping tombolnya, dan 409 sebagai pemberitahuan bahwa pesanan sudah ditutup dan layar perlu dimuat ulang.

12.8.4 Penyelesaian di Spring

Pada **Kode 12.10**, `updateStatus` lebih dahulu memanggil `ensureOpen`, pasangan Spring bagi *middleware* Bab 9, yang melempar `ResponseStatusException` dengan 409 untuk pesanan yang sudah pergi, dengan pesan yang sama dengan pesan Laravel. Kemudian ia meminta status berikutnya kepada `LaundryRules.advance`, mengubah penolakan aturan menjadi `ApiValidationException` pada `status`, lalu membandingkan status berikutnya dengan status yang diminta, dan sekali lagi menjawab 422 bila keduanya berbeda. Baru sesudah itu ia memanggil `order.advance`. Laravel tidak memerlukan perubahan.

12.8.5 Penyelesaian keliru yang menggoda

Membiarkan klien hanya mengirim "maju", tanpa menyebutkan statusnya, seperti tombol di meja kasir web, menghindari perbandingan sama sekali. Namun cara itu juga membuat dua sopir yang menekan "sudah diambil" pada saat bersamaan memajukan pesanan *finished* menjadi *departed* lalu menggagalkan permintaan kedua dengan 409 yang membingungkan, atau lebih buruk lagi, dengan "maju" pada formulir web, memajukan pesanan *queued* dua langkah. Menyebutkan status yang diinginkan membuat permintaan aman diulang: permintaan identik yang kedua ditolak, bukan melakukan hal yang baru.

12.9 Masalah 12.6: Pembayaran

12.9.1 Masalahnya

`POST /api/orders/{id}/payments` dengan `{"amount": ..., "method": "..."}` mencatat pembayaran sedikitnya Rp 1.000 secara tunai, QRIS, atau transfer, tidak pernah melebihi sisa tagihan di Persamaan (12.1), dan menjawab 201 dengan seluruh pesanan, termasuk daftar `payments`-nya.

12.9.2 Memahaminya dengan tangan

Pesanan dari Masalah 12.5 masih bertagihan Rp 34.000. Pembayaran Rp 40.000 ditolak dengan 422 dan `errors.amount` ("Only Rp 34.000 is still owed."). Pembayaran Rp 34.000 dengan QRIS dicatat, dan jawabannya menunjukkan `"balance": 0` serta daftar `payments` yang unsur pertamanya memiliki `"method": "qris"` dan sebuah `id`.

12.9.3 Membandingkan keduanya

Laravel sudah memiliki *endpoint* ini sejak Bab 10, dengan `StorePaymentRequest` Bab 9. Spring tidak memilikinya. Keputusannya sederhana, yaitu perilaku Laravel; pekerjaannya ada di Spring.

12.9.4 Penyelesaian di Spring

Kode 12.11 adalah *record* permintaannya. Jumlahnya bertipe `Long`, bukan `long`, agar jumlah yang tidak dikirim bernilai `null` dan gagal di `@NotNull`, bukan diam-diam menjadi 0.

Kode 12.11 `CreatePaymentRequest.java` (Spring): jumlah dan metode pembayaran.

```
package com.profitina.laundry.web;

import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;

public record CreatePaymentRequest(
        @NotNull @Min(1000) Long amount,
        @NotBlank String method) {
}
```

Metode `pay` pada **Kode 12.10** memeriksa, berturut-turut: pesanannya masih terbuka (409), metodenya salah satu dari tiga kata (422 pada `method`), dan jumlahnya tidak melebihi sisa tagihan (422 pada `amount`). Kemudian ia memanggil `order.addPayment`, metode baru pada entitas `Order` yang membuat `Payment`, menambahkannya ke daftar pesanan dan mengembalikannya, lalu langsung menyimpan pembayaran itu dengan `payments.save(...)`, melalui `PaymentRepository` yang kini juga diterima *controller*. Karena id pembayaran dibangkitkan basis data, penyimpanan itu langsung mengirim perintah *insert*, sehingga pembayaran baru sudah memiliki `id` ketika `OrderView.withPayments` menulis jawabannya; kemudian, ketika transaksi di-*commit*, *cascade* Bab 11 mendapati pembayaran itu sudah tersimpan dan tidak melakukan apa-apa lagi. **Kode 12.12** adalah pembayaran di dalam jawaban.

Kode 12.12 `PaymentView.java` (Spring): satu pembayaran, dengan waktunya dalam format kontrak.

```
package com.profitina.laundry.web;

import com.fasterxml.jackson.annotation.JsonFormat;
import com.profitina.laundry.domain.Payment;
import java.time.LocalDateTime;

public record PaymentView(
        Long id,
        long amount,
        String method,
        @JsonFormat(pattern = OrderView.TIME) LocalDateTime paidAt) {

    public static PaymentView of(Payment p) {
        return new PaymentView(p.getId(), p.getAmount(),
                p.getMethod().value(), p.getPaidAt());
    }
}
```

12.9.5 Penyelesaian keliru yang menggoda

Menghilangkan `payments.save` dan mengandalkan *cascade* di akhir metode berhasil untuk basis datanya: ketika transaksi di-*commit*, pembayaran di dalam daftar pesanan tersimpan. Namun jawabannya ditulis *sebelum commit*, ketika pembayaran baru masih memiliki `id` bernilai `null`, sehingga klien menerima `"id": null`. Ini kekeliruan kecil yang langsung tertangkap uji dan tidak pernah terlihat oleh orang yang mengklik aplikasi, dan justru karena itulah uji di Subbab 12.11 memeriksa jawabannya, bukan hanya kode statusnya.

12.10 Masalah 12.7: Laporan Harian

12.10.1 Masalahnya

`GET /api/reports/daily?date=YYYY-MM-DD` menjawab dengan angka-angka hari itu: `orders_received`, jumlah pesanan yang diterima hari itu; `billed`, jumlah total pesanan-pesanan itu; `collected`, jumlah pembayaran yang dilakukan hari itu, dari hari mana pun pesanannya berasal; dan `by_status`, jumlah pesanan hari itu di setiap status, dengan setiap status hadir, bahkan bila nilainya 0. Tanggal yang tidak ada atau ditulis keliru adalah 422 pada `date`. Hitungannya harus memenuhi Persamaan (12.2).

$$\sum_{s \in S} n_s = N \tag{12.2}$$

Di sini S adalah himpunan status {queued, finished, departed}, n_s adalah jumlah pesanan hari itu yang kini berstatus s, dan N adalah `orders_received`. Persamaan itu menyatakan bahwa setiap pesanan yang diterima hari itu terhitung tepat di satu status.

12.10.2 Memahaminya dengan tangan

Misalkan pada 25 September datang dua pesanan, satu seharga Rp 34.000 yang kini sudah *departed* dan satu seharga Rp 36.000 yang masih *queued*, dan pembayaran hari itu hanyalah Rp 34.000 untuk pesanan pertama dan Rp 20.000 untuk pesanan yang diterima sehari sebelumnya. Laporannya adalah `orders_received` 2, `billed` 70000, `collected` 54000, dan `by_status` `{"queued": 1, "finished": 0, "departed": 1}`; $1 + 0 + 1 = 2$, seperti yang dituntut Persamaan (12.2). Perhatikan bahwa `collected` bukan bagian dari `billed`: uang yang diterima hari ini dapat saja milik pesanan kemarin.

12.10.3 Membandingkan keduanya

Kedua API belum memiliki *endpoint* ini. Bab 8 membangun laporan web berisi pembayaran per hari dan per metode di Laravel; kuerinya menjadi titik berangkat. “Satu hari” adalah selang setengah terbuka dari tengah malam sampai tengah malam berikutnya, $[d, d + 1)$, sehingga pesanan yang diterima tepat pukul 00.00 hanya termasuk satu hari.

12.10.4 Penyelesaian di kedua kerangka kerja

Kode 12.13 adalah *controller* Laravel.

Kode 12.13 `Api/ReportController.php` (Laravel): hitungan per status, total dan pembayaran hari itu, dengan setiap status hadir.

```php
<?php

namespace App\Http\Controllers\Api;

use App\Enums\OrderStatus;
use App\Http\Controllers\Controller;
use App\Models\Order;
use App\Models\Payment;
use Illuminate\Http\Request;
use Illuminate\Support\Carbon;
use Illuminate\Support\Facades\DB;

class ReportController extends Controller
{
    // GET /api/reports/daily?date=2026-09-25
    public function daily(Request $request)
    {
        $request->validate(['date' => ['required', 'date_format:Y-m-d']]);

        $from = Carbon::createFromFormat('Y-m-d', $request->query('date'))
            ->startOfDay();
        $to = $from->copy()->addDay();              // [from, to): one day

        $orders = Order::query()
            ->where('received_at', '>=', $from)
            ->where('received_at', '<', $to);

        $counts = (clone $orders)->toBase()
            ->select('status', DB::raw('count(*) as n'))
            ->groupBy('status')
            ->pluck('n', 'status');

        $byStatus = [];
        foreach (OrderStatus::cases() as $status) {   // every status, even 0
            $byStatus[$status->value] = (int) ($counts[$status->value] ?? 0);
        }
```

Kode 12.13 (Lanjutan) *Api/ReportController.php (Laravel): hitungan per status, total dan pembayaran hari itu, dengan setiap status hadir.*

```php
        return response()->json(['data' => [
            'date' => $from->format('Y-m-d'),
            'orders_received' => array_sum($byStatus),
            'billed' => (int) (clone $orders)->sum('total'),
            'collected' => (int) Payment::query()
                ->where('paid_at', '>=', $from)
                ->where('paid_at', '<', $to)
                ->sum('amount'),
            'by_status' => $byStatus,
        ]]);
    }
}
```

Aturan validasi `date_format:Y-m-d` menolak apa pun selain tanggal yang ditulis sesuai kontrak. `$orders` adalah *query builder* untuk pesanan hari itu, dan `clone` membuat salinannya untuk setiap pemakaian, karena sebuah *builder* berubah oleh setiap `where` yang ditambahkan kepadanya. Hitungan per status dikerjakan basis data dengan `GROUP BY`; `toBase()` meminta baris biasa alih-alih model `Order`, sehingga *cast* *enum* tidak mengganggu `pluck('n', 'status')`, yang mengembalikan hitungan dengan kunci kata status. Perulangan atas `OrderStatus::cases()` menambahkan status yang tidak memiliki pesanan, dengan nilai 0, sehingga klien tidak pernah harus menebak apakah kunci yang hilang berarti nol.

Versi Spring memerlukan dua metode *repository* baru. **Kode 12.14** menambahkannya ke `OrderRepository` sebagai kueri JPQL dengan `@Query`. `countByStatusBetween` mengembalikan daftar `StatusCount`, antarmuka *projection* kecil yang *getter*-nya diisi Spring Data dari alias `status` dan `n` di kueri [3]. `billedBetween` menjumlahkan total; `coalesce(..., 0)` mengubah `null` yang dikembalikan `SUM` bila tidak ada baris menjadi 0.

Kode 12.14 `OrderRepository.java` (Spring): kueri-kueri Bab 11 ditambah dua kueri yang diperlukan laporan.

```
package com.profitina.laundry.repository;

import com.profitina.laundry.domain.Order;
import com.profitina.laundry.domain.OrderStatus;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.EntityGraph;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

public interface OrderRepository extends JpaRepository<Order, Long> {

    @EntityGraph(attributePaths = {"customer", "service"})
    Page<Order> findAllByOrderByReceivedAtDescIdDesc(Pageable pageable);

    @EntityGraph(attributePaths = {"customer", "service"})
    Page<Order> findByStatusOrderByReceivedAtDescIdDesc(OrderStatus status,
                                                        Pageable pageable);

    @EntityGraph(attributePaths = {"customer", "service", "payments"})
    Optional<Order> findWithDetailsById(Long id);

    /** One row per status: the shape the daily report needs. */
    interface StatusCount {
        OrderStatus getStatus();

        long getN();
    }

    @Query("""
            select o.status as status, count(o) as n
            from Order o
            where o.receivedAt >= :from and o.receivedAt < :to
            group by o.status""")
    List<StatusCount> countByStatusBetween(@Param("from") LocalDateTime from,
                                           @Param("to") LocalDateTime to);
```

Kode 12.14 (Lanjutan) *OrderRepository.java (Spring): kueri-kueri Bab 11 ditambah dua kueri yang diperlukan laporan.*

```
    @Query("""
            select coalesce(sum(o.total), 0)
            from Order o
            where o.receivedAt >= :from and o.receivedAt < :to""")
    long billedBetween(@Param("from") LocalDateTime from,
                       @Param("to") LocalDateTime to);
}
```

`PaymentRepository` mendapat `collectedBetween`, penjumlahan sejenis atas `paidAt`. **Kode 12.15** adalah *controller*-nya, dan *record* `DailyReport` di folder bab ini memuat kelima angka itu, yang ditulis strategi snake_case sebagai `orders_received` dan `by_status`.

Kode 12.15 `ReportController.java` (Spring): laporan yang sama, dengan `LinkedHashMap` yang mendaftar setiap status secara berurutan.

```
package com.profitina.laundry.web;

import com.profitina.laundry.domain.OrderStatus;
import com.profitina.laundry.repository.OrderRepository;
import com.profitina.laundry.repository.PaymentRepository;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.format.DateTimeParseException;
import java.util.LinkedHashMap;
import java.util.Map;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ReportController {

    private final OrderRepository orders;
    private final PaymentRepository payments;

    public ReportController(OrderRepository orders,
                            PaymentRepository payments) {
        this.orders = orders;
        this.payments = payments;
    }

    /** GET /api/reports/daily?date=2026-09-25 */
    @GetMapping("/api/reports/daily")
    @Transactional(readOnly = true)
    public DataView<DailyReport> daily(
            @RequestParam(required = false) String date) {
        LocalDate day = parseDate(date);
        LocalDateTime from = day.atStartOfDay();
        LocalDateTime to = from.plusDays(1);   // [from, to): one whole day
```

Kode 12.15 (Lanjutan) *ReportController.java (Spring): laporan yang sama, dengan LinkedHashMap yang mendaftar setiap status secara berurutan.*

```
        Map<String, Long> byStatus = new LinkedHashMap<>();
        for (OrderStatus s : OrderStatus.values()) {
            byStatus.put(s.value(), 0L);       // every status, even if 0
        }
        for (OrderRepository.StatusCount c
                : orders.countByStatusBetween(from, to)) {
            byStatus.put(c.getStatus().value(), c.getN());
        }
        long received = byStatus.values().stream()
                .mapToLong(Long::longValue).sum();

        return new DataView<>(new DailyReport(day.toString(), received,
                orders.billedBetween(from, to),
                payments.collectedBetween(from, to), byStatus));
    }

    private static LocalDate parseDate(String date) {
        if (date == null || date.isBlank()) {
            throw new ApiValidationException("date",
                    "The date field is required.");
        }
        try {
            return LocalDate.parse(date);
        } catch (DateTimeParseException e) {
            throw new ApiValidationException("date",
                    "The date field must match the format Y-m-d.");
        }
    }
}
```

12.10.5 Penyelesaian keliru yang menggoda

Memilih hari dengan `WHERE date(received_at) = ?` enak dibaca dan benar pada tabel yang kecil. Cara itu juga lambat pada tabel yang besar, karena menerapkan fungsi pada kolom mencegah basis data memakai indeks pada `received_at`, dan pada sebagian basis data `date()` menafsirkan waktu yang tersimpan menurut zona server, bukan zona toko. Bentuk selang `received_at >= :from AND received_at < :to` membandingkan kolomnya secara langsung, dapat memakai indeks, dan menyatakan dengan tepat saat-saat mana yang termasuk hari itu. (Server rujukan memakai perbandingan teks sepuluh karakter pertama, yang benar untuk format penyimpanannya dan cukup untuk ukurannya, sekaligus contoh jalan pintas yang sebaiknya tetap tinggal di mainan.)

12.11 Memeriksa Kontrak

12.11.1 Pemeriksanya

Kode 12.16 adalah kontrak dalam bentuk program. Program ini hanya memerlukan PHP: ia mengirim setiap permintaan dengan fungsi *stream* HTTP milik PHP sendiri, membaca kode status dari *header* respons pertama, menguraikan JSON-nya, lalu mencetak PASS atau FAIL untuk masing-masing dari enam belas pemeriksaan.

Kode 12.16 `contract/check_api.php`: enam belas pemeriksaan kontrak, terhadap server mana pun.

```php
<?php

// The Profitina Laundry API contract as a program. Plain PHP, no packages:
//   php check_api.php http://127.0.0.1:8000 [token]
// Point it at the Laravel app, the Spring Boot app or any other server.

$base = rtrim($argv[1] ?? 'http://127.0.0.1:8000', '/');
$token = $argv[2] ?? null;
$passed = $failed = 0;

// Sends one request; returns [status code, decoded JSON body].
function call(string $method, string $path, ?array $body = null): array
{
    global $base, $token;
    $headers = ['Accept: application/json',
                'Content-Type: application/json'];
    if ($token !== null) {
        $headers[] = "Authorization: Bearer $token";
    }
    $context = stream_context_create(['http' => [
        'method' => $method,
        'header' => implode("\r\n", $headers),
        'content' => $body === null ? '' : json_encode($body),
        'ignore_errors' => true,          // give us the 4xx bodies too
    ]]);
    $raw = @file_get_contents($base . $path, false, $context);
    if ($raw === false) {
        fwrite(STDERR, "Cannot reach $base$path\n");
        exit(2);
    }
    preg_match('/^HTTP\/\S+ (\d{3})/', $http_response_header[0], $m);
    return [(int) $m[1], json_decode($raw, true)];
}

function check(string $what, bool $ok): void
{
    global $passed, $failed;
    $ok ? $passed++ : $failed++;
    printf("%s  %s\n", $ok ? 'PASS' : 'FAIL', $what);
}

$time = '/^\d{4}-\d\d-\d\dT\d\d:\d\d:\d\d$/';
```

Kode 12.16 (Lanjutan) *contract/check_api.php: enam belas pemeriksaan kontrak, terhadap server mana pun.*

```
[$s, $j] = call('GET', '/api/services');
$first = $j['data'][0] ?? [];
check('GET /api/services -> 200, S0001 first, 8000 per kg',
    $s === 200 && ($first['id'] ?? '') === 'S0001'
    && ($first['unit'] ?? '') === 'kg' && ($first['price'] ?? 0) === 8000);

$phone = '0812' . random_int(10000000, 99999999);   // new on every run
[$s, $j] = call('POST', '/api/customers', ['name' => 'Sri Rahayu',
    'phone' => substr($phone, 0, 4) . '-' . substr($phone, 4)]);
$customer = $j['data'] ?? [];
check('POST /api/customers -> 201, phone kept as digits only',
    $s === 201 && ($customer['phone'] ?? '') === $phone);
[$s, $j] = call('POST', '/api/customers', ['name' => 'Sri Rahayu',
    'phone' => $phone]);
check('the same phone again -> 422, errors.phone',
    $s === 422 && isset($j['errors']['phone']));

[$s, $j] = call('POST', '/api/orders', [
    'customer_id' => $customer['id'] ?? 0, 'outlet_id' => 'O01',
    'service_id' => 'S0001', 'quantity' => 2.5, 'pickup' => true,
    'delivery_address' => 'Jl. Kenanga 5, Surabaya']);
$o = $j['data'] ?? [];
check('POST /api/orders -> 201, total 34000 (3 kg + pickup), queued',
    $s === 201 && ($o['total'] ?? 0) === 34000
    && ($o['balance'] ?? 0) === 34000 && ($o['status'] ?? '') === 'queued');
check('times are YYYY-MM-DDTHH:MM:SS, finished_at is null',
    preg_match($time, $o['received_at'] ?? '') === 1
    && array_key_exists('finished_at', $o) && $o['finished_at'] === null);
$url = '/api/orders/' . ($o['id'] ?? 0);

[$s, $j] = call('POST', '/api/orders', ['quantity' => 0, 'pickup' => true]);
$fields = array_keys($j['errors'] ?? []);
check('a bad order -> 422, all of its errors at once',
    $s === 422 && is_string($j['message'] ?? null)
    && ! array_diff(['customer_id', 'service_id', 'quantity',
        'delivery_address'], $fields));
```

Kode 12.16 (Lanjutan) *contract/check_api.php: enam belas pemeriksaan kontrak, terhadap server mana pun.*

```
$move = fn (string $to) => call('PATCH', "$url/status", ['status' => $to]);
[$s, $j] = $move('departed');
check('queued -> departed skips a step: 422, errors.status',
    $s === 422 && isset($j['errors']['status']));
[$s, $j] = $move('finished');
check('queued -> finished: 200, finished_at set',
    $s === 200 && preg_match($time, $j['data']['finished_at'] ?? '') === 1);
[$s, $j] = $move('departed');
check('finished -> departed before payment: 422, errors.status',
    $s === 422 && isset($j['errors']['status']));

[$s, $j] = call('POST', "$url/payments", ['amount' => 40000,
    'method' => 'cash']);
check('paying more than the balance -> 422, errors.amount',
    $s === 422 && isset($j['errors']['amount']));
[$s, $j] = call('POST', "$url/payments", ['amount' => 34000,
    'method' => 'qris']);
check('paying the balance -> 201, balance 0, payments[0].method qris',
    $s === 201 && ($j['data']['balance'] ?? -1) === 0
    && ($j['data']['payments'][0]['method'] ?? '') === 'qris');

[$s, $j] = $move('departed');
check('finished -> departed after payment: 200',
    $s === 200 && ($j['data']['status'] ?? '') === 'departed');
[$s, $j] = call('POST', "$url/payments", ['amount' => 1000,
    'method' => 'cash']);
check('a departed order is closed: 409 with a message',
    $s === 409 && is_string($j['message'] ?? null));

[$s, $j] = call('GET', '/api/orders?status=departed');
check('GET /api/orders?status=departed -> 200, data[] and meta',
    $s === 200 && array_is_list($j['data'] ?? null)
    && ($j['meta']['total'] ?? 0) >= 1
    && ($j['meta']['current_page'] ?? 0) === 1);
[$s, $j] = call('GET', '/api/orders/999999');
check('GET /api/orders/999999 -> 404 with a message',
    $s === 404 && is_string($j['message'] ?? null));

[$s, $j] = call('GET', '/api/reports/daily?date=' . date('Y-m-d'));
$r = $j['data'] ?? [];
check('GET /api/reports/daily -> 200, the counts add up',
    $s === 200 && ($r['orders_received'] ?? 0) >= 1
    && ($r['collected'] ?? 0) >= 34000
    && array_sum($r['by_status'] ?? []) === $r['orders_received']);
```

Kode 12.16 (Lanjutan) *contract/check_api.php: enam belas pemeriksaan kontrak, terhadap server mana pun.*

```
printf("\n%d passed, %d failed\n", $passed, $failed);
exit($failed === 0 ? 0 : 1);
```

Pemeriksa ini mengikuti kehidupan satu pesanan, seperti yang akan dilakukan aplikasi seorang sopir: ia membaca daftar harga, mendaftarkan pelanggan baru dengan nomor telepon yang dipilih acak, agar dapat dijalankan berulang-ulang terhadap basis data yang sama, mencoba telepon yang sama untuk kedua kalinya, membuat pesanan, melanggar setiap aturan pesanan baru sekaligus, membawa pesanan itu melewati status-statusnya dengan lompatan yang ditolak dan kepergian belum lunas yang ditolak, membayar terlalu banyak lalu membayar jumlah yang tepat, menutup pesanan, mencoba membayar pesanan yang sudah tertutup, mendaftar pesanan yang sudah pergi, meminta pesanan yang tidak ada, dan akhirnya membaca laporan hari ini serta memeriksa

Persamaan (12.2). `'ignore_errors' => true` itulah yang memungkinkannya membaca badan jawaban 4xx; tanpanya PHP akan menganggapnya kegagalan dan tidak mengembalikan apa pun.

12.11.2 Satu jalan yang sungguhan

Dijalankan di satu terminal dengan `php -S 127.0.0.1:8000 reference/server.php` dan diperiksa dari terminal lain dengan `php check_api.php http://127.0.0.1:8000`, server rujukan memberikan keluaran berikut, yang merupakan keluaran sesungguhnya dari jalan yang dilakukan di tempat buku ini ditulis, pada basis data kosong dan sekali lagi pada basis data yang sama:

```
PASS  GET /api/services -> 200, S0001 first, 8000 per kg
PASS  POST /api/customers -> 201, phone kept as digits only
PASS  the same phone again -> 422, errors.phone
PASS  POST /api/orders -> 201, total 34000 (3 kg + pickup), queued
PASS  times are YYYY-MM-DDTHH:MM:SS, finished_at is null
PASS  a bad order -> 422, all of its errors at once
PASS  queued -> departed skips a step: 422, errors.status
PASS  queued -> finished: 200, finished_at set
PASS  finished -> departed before payment: 422, errors.status
PASS  paying more than the balance -> 422, errors.amount
PASS  paying the balance -> 201, balance 0, payments[0].method qris
PASS  finished -> departed after payment: 200
PASS  a departed order is closed: 409 with a message
PASS  GET /api/orders?status=departed -> 200, data[] and meta
PASS  GET /api/orders/999999 -> 404 with a message
PASS  GET /api/reports/daily -> 200, the counts add up

16 passed, 0 failed
```

Server rujukan bukan aplikasi ketiga untuk toko. Ia adalah kontrak yang dibuat dapat dijalankan dengan cara yang paling sederhana, dan nilainya adalah membuktikan pemeriksanya: ketika pemeriksa melaporkan kegagalan terhadap Laravel atau Spring, kesalahannya ada di aplikasi itu, bukan di pemeriksa. `README.md` bab ini memperlihatkan cara mengarahkan pemeriksa ke kedua aplikasi sungguhan; API Laravel memerlukan token Bab 10 sebagai argumen kedua, sedangkan API Spring belum memiliki proses masuk sampai Bab 14.

Sebelum perubahan bab ini, pemeriksa akan menghentikan aplikasi Spring Bab 11 di pemeriksaan pertamanya, karena `data` tidak ada pada daftar harga, dan aplikasi Laravel Bab 10 di pemeriksaan pertamanya juga, karena tidak ada *endpoint* daftar harga; lebih ke bawah, format waktu, bentuk galat, dan *endpoint* pembayaran yang belum ada akan gagal berturut-turut. Begitulah pemeriksaan kontrak dipakai dalam praktik: jalankan, perbaiki kegagalan pertama, jalankan lagi.

12.11.3 Uji di dalam setiap kerangka kerja

Pemeriksa itu menguji dari luar. Setiap kerangka kerja juga mendapat ujinya sendiri, yang berjalan tanpa server dan tanpa jaringan, dengan gaya Bab 10. **Kode 12.17** menambahkan hal-hal yang belum dicakup uji Bab 10: daftar harga, format waktu, dan laporan. `travelTo` menetapkan jam pada saat yang diketahui, sehingga `received_at` yang diharapkan dapat ditulis dengan tepat [4].

Kode 12.17 `tests/Feature/ContractTest.php` (Laravel): hal-hal baru dalam kontrak, dengan jam yang ditetapkan.

```php
<?php

namespace Tests\Feature;

use App\Models\Customer;
use App\Models\User;
use Database\Seeders\PriceListSeeder;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Laravel\Sanctum\Sanctum;
use Tests\TestCase;

// The points of the contract that Chapter 10's API did not yet keep.
class ContractTest extends TestCase
{
    use RefreshDatabase;

    protected function setUp(): void
    {
        parent::setUp();
        $this->seed(PriceListSeeder::class);
        Sanctum::actingAs(User::factory()->create());
    }

    public function test_the_price_list_is_an_endpoint(): void
    {
        $this->getJson('/api/services')->assertOk()
            ->assertJsonPath('data.0.id', 'S0001')
            ->assertJsonPath('data.0.unit', 'kg')
            ->assertJsonPath('data.0.price', 8000);
    }

    public function test_times_use_the_contract_format(): void
    {
        $this->travelTo('2026-09-25 08:15:00');

        $this->postJson('/api/orders', [
            'customer_id' => Customer::factory()->create()->id,
            'outlet_id' => 'O01', 'service_id' => 'S0001', 'quantity' => 4.5,
        ])->assertCreated()
            ->assertJsonPath('data.received_at', '2026-09-25T08:15:00')
            ->assertJsonPath('data.finished_at', null);
    }
```

Kode 12.17 (Lanjutan) *tests/Feature/ContractTest.php (Laravel): hal-hal baru dalam kontrak, dengan jam yang ditetapkan.*

```php
    public function test_the_daily_report(): void
    {
        $this->getJson('/api/reports/daily')
            ->assertStatus(422)->assertJsonValidationErrors('date');

        $this->getJson('/api/reports/daily?date=2026-09-25')
            ->assertOk()
            ->assertJsonPath('data.orders_received', 0)
            ->assertJsonPath('data.by_status.queued', 0);
    }
}
```

Kode 12.18 adalah uji pertama aplikasi Spring yang menjalankan seluruh aplikasi. `@SpringBootTest` menjalankannya dengan konfigurasi sungguhan dan basis data di memorinya, yang diisi `DemoData`; `@AutoConfigureMockMvc` menyediakan `MockMvc`, yang mengirim permintaan ke *controller* tanpa jaringan, seperti `postJson` di Laravel [5]. Di Spring Boot 4 anotasi ini berasal dari *starter* `spring-boot-starter-webmvc-test`, yang ditambahkan `pom.xml` bab ini [6].

Kode 12.18 `OrderApiTest.java` (Spring): aturan harga, semua galat sekaligus, seluruh kehidupan sebuah pesanan, dan laporan, melalui MockMvc.

```
package com.profitina.laundry.web;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

import com.jayway.jsonpath.JsonPath;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.webmvc.test.autoconfigure.AutoConfigureMockMvc;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;

/** The whole application, started once, driven through HTTP calls. */
@SpringBootTest
@AutoConfigureMockMvc
class OrderApiTest {

    @Autowired
    private MockMvc mvc;

    private String send(String method, String url, String json, int expect)
            throws Exception {
        var request = switch (method) {
            case "POST" -> post(url);
            case "PATCH" -> patch(url);
            default -> get(url);
        };
        if (json != null) {
            request.contentType(MediaType.APPLICATION_JSON).content(json);
        }
        return mvc.perform(request).andExpect(status().is(expect))
                .andReturn().getResponse().getContentAsString();
    }

    @Test
    void anOrderIsPricedByTheRule() throws Exception {
        mvc.perform(post("/api/orders")
                        .contentType(MediaType.APPLICATION_JSON)
                        .content("""
                            {"customer_id": 1, "service_id": "S0001",
                             "quantity": 2.5, "pickup": true,
                             "delivery_address": "Jl. Kenanga 5"}"""))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.data.total").value(34000))
                .andExpect(jsonPath("$.data.status").value("queued"));
```

Kode 12.18 (Lanjutan) *OrderApiTest.java (Spring): aturan harga, semua galat sekaligus, seluruh kehidupan sebuah pesanan, dan laporan, melalui MockMvc.*

```
    }

    @Test
    void badInputGets422WithEveryError() throws Exception {
        mvc.perform(post("/api/orders")
                        .contentType(MediaType.APPLICATION_JSON)
                        .content("{\"quantity\": 0, \"pickup\": true}"))
                .andExpect(status().is(422))
                .andExpect(jsonPath("$.errors.customer_id").exists())
                .andExpect(jsonPath("$.errors.service_id").exists())
                .andExpect(jsonPath("$.errors.quantity").exists())
                .andExpect(jsonPath("$.errors.delivery_address").exists());
    }

    @Test
    void theWholeLifeOfAnOrder() throws Exception {
        String body = send("POST", "/api/orders", """
                {"customer_id": 2, "service_id": "S0001", "quantity": 2.5,
                 "pickup": true, "delivery_address": "Jl. Kenanga 5"}""",
                201);
        String url = "/api/orders/" + JsonPath.read(body, "$.data.id");
        String to = "{\"status\": \"%s\"}";

        send("PATCH", url + "/status", to.formatted("departed"), 422);
        send("PATCH", url + "/status", to.formatted("finished"), 200);
        send("PATCH", url + "/status", to.formatted("departed"), 422);
        send("POST", url + "/payments",
                "{\"amount\": 40000, \"method\": \"cash\"}", 422);
        send("POST", url + "/payments",
                "{\"amount\": 34000, \"method\": \"qris\"}", 201);
        send("PATCH", url + "/status", to.formatted("departed"), 200);
        send("POST", url + "/payments",
                "{\"amount\": 1000, \"method\": \"cash\"}", 409);
    }

    @Test
    void theDailyReportNeedsADate() throws Exception {
        send("GET", "/api/reports/daily", null, 422);
        mvc.perform(get("/api/reports/daily?date=2026-09-25"))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.data.by_status.queued").exists());
    }
}
```

`jsonPath("$.data.total").value(34000)` memeriksa satu nilai dalam jawaban dengan ekspresi JSONPath; `JsonPath.read` mengambil id pesanan baru dari badan jawaban agar permintaan-permintaan berikutnya dapat memakainya. Metode pembantu `send` membuat kehidupan sebuah pesanan cukup ditulis satu baris per langkah, dengan status yang diharapkan di ujungnya, sehingga uji itu terbaca seperti contoh yang dikerjakan dengan tangan di Masalah 12.5.

12.12 Memakainya Sungguhan

- **Tulislah kontrak sebelum klien kedua.** Satu klien dapat hidup dengan apa pun yang dikirim server. Pada hari muncul klien kedua atau server kedua, tuliskan aturannya, seperti yang dilakukan **Tabel 12.1** dan **Tabel 12.2**, dan jadikan aturan itu dapat dijalankan.
- **Terbitkan dalam format baku.** OpenAPI Specification menggambarkan URL, isian, tipe, dan kode status dalam berkas yang dapat dibaca perangkat lunak, diubah menjadi dokumentasi, dan

dipakai untuk membangkitkan kode klien [7]. Tabel-tabel di Subbab 12.3 adalah ringkasan yang mudah dibaca dari apa yang akan dinyatakan berkas semacam itu.

- **Tambahkan, jangan pernah mengubah.** Isian baru, *endpoint* baru, atau parameter opsional baru tidak merusak klien yang mengabaikan apa yang tidak dikenalnya. Mengganti nama atau membuang isian akan merusaknya; lakukanlah di versi API yang baru, seperti `/api/v2`, sementara versi lama tetap berjalan sampai setiap klien pindah.
- **Simpanlah satu daftar penolakan.** Kode status yang dipilih kasus demi kasus akan saling menyimpang. Tabel singkat tentang arti 404, 409, dan 422 dalam API ini, yang diterapkan di mana-mana, lebih berharga daripada pilihan cerdik mana pun.
- **Periksalah setiap server dengan cara yang sama.** Pemeriksa seperti `check_api.php` berjalan dalam hitungan detik sebelum setiap rilis, di laptop maupun di server *build*, dan memeriksa apa yang dilihat klien, yang tidak diperiksa uji milik kerangka kerja itu sendiri.

12.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Membiarkan kerangka kerja memilih kontraknya.** Format tanggal bawaan, badan galat bawaan, dan amplop bawaan berbeda-beda; pilihlah masing-masing dengan sengaja dan tuliskan.
- **Membandingkan waktu sebagai teks lintas zona.** `2026-09-25T01:15:00Z` dan `2026-09-25T08:15:00` adalah saat yang sama dan *string* yang berbeda. Tetapkan satu format dan satu zona di kontrak.
- **Mencampuradukkan 400, 409, dan 422.** 400 untuk badan yang tidak dapat dibaca, 422 untuk isi yang melanggar aturan, 409 untuk sumber daya yang keadaannya melarang permintaan itu.
- **Kunci galat dengan nama milik server.** `serviceId` pada galat Java, atau `service` padahal permintaannya berbunyi `service_id`, tidak dapat dicocokkan klien dengan isiannya. Pakailah nama isian dari permintaan.
- **Galat dalam dua putaran.** Klien yang memperbaiki tiga galat yang diperlihatkan lalu menerima galat keempat merasa dicurangi. Kumpulkan setiap galat sebelum menjawab.
- **Kunci yang hilang untuk nilai nol.** Laporan yang meninggalkan status tanpa pesanan memaksa setiap klien menebak; kirimkan setiap kunci.
- **Hanya memeriksa kode status dalam uji.** Jawaban dengan status yang benar tetapi `"id": null` tetap merusak klien. Periksalah isian-isian yang dipakai klien.

12.14 Ringkasan Bab

- Kontrak menetapkan *endpoint* (**Tabel 12.1**), amplop `data`, nama snake_case, kata status berhuruf kecil, rupiah utuh, satu format tanggal-waktu dalam waktu toko, dan penolakan di **Tabel 12.2**.
- Masalah 12.1 menambahkan daftar harga ke Laravel dan amplop `data` ke Spring (`DataView`).
- Masalah 12.2 menetapkan setiap waktu menjadi `YYYY-MM-DDTHH:MM:SS` dalam waktu toko: format eksplisit di *resource* Laravel dan `@JsonFormat` di *record* Spring.
- Masalah 12.3 memberi Spring bentuk galat 422 milik Laravel melalui *exception handler* yang mengubah nama isian menjadi snake_case.
- Masalah 12.4 membuat Spring melaporkan setiap galat pesanan baru sekaligus dengan memvalidasi secara manual dan menambahkan pemeriksaan basis data ke peta yang sama.
- Masalah 12.5 memisahkan 422 untuk langkah yang ditolak dari 409 untuk pesanan yang tertutup, mengikuti arti kedua kode HTTP itu.

- Masalah 12.6 menambahkan pembayaran ke Spring, dengan menyimpan setiap pembayaran seketika sehingga jawaban membawa id-nya; sisa tagihan mengikuti Persamaan (12.1).
- Masalah 12.7 menambahkan laporan harian ke keduanya, atas hari setengah terbuka $[d, d+1)$, dengan setiap status hadir sehingga Persamaan (12.2) terpenuhi.
- `check_api.php` memeriksa server mana pun terhadap kontrak; jalannya terhadap server rujukan lolos keenam belas pemeriksaan, dan setiap kerangka kerja memiliki ujinya sendiri untuk hal-hal yang sama.

12.15 Soal Latihan

1. Tambahkan ke kontrak penyaring opsional `GET /api/orders?customer_id=5`. Tulislah kalimat kontrak untuknya, dan perubahan pada metode `index` di setiap kerangka kerja.
2. Tanpa menjalankan apa pun, telusurilah `check_api.php` terhadap aplikasi Spring Bab 11 (sebelum perubahan bab ini). Pemeriksaan mana yang gagal pertama kali, dan mengapa? Sebutkan tiga pemeriksaan berikutnya yang juga akan gagal, beserta masalah bab ini yang memperbaiki masing-masing.
3. Laravel menerima `"quantity": "2,5"` karena `StoreOrderRequest` mengubah koma menjadi titik; Jackson di Spring menolak *string* itu. Haruskah kontrak mengizinkan jumlah yang ditulis sebagai teks dengan koma? Berikan argumen untuk satu jawaban, dan sebutkan apa yang harus diubah setiap kerangka kerja untuk mengikutinya.
4. Tulislah dua pemeriksaan tambahan untuk `check_api.php`: bahwa `GET /api/customers?phone=` dengan telepon pelanggan baru menemukan tepat pelanggan itu, dan bahwa `GET /api/orders/{id}` untuk pesanan baru membawa pembayarannya. *Endpoint* mana yang perlu diubah di kerangka kerja mana agar pemeriksaan pertama lolos?
5. Kantor akuntan juga menginginkan laporan berupa angka per metode pembayaran. Rancanglah isian tambahan pada jawaban sehingga klien lama tetap berjalan, dan tulislah kueri Laravel yang mengisinya.
6. Toko kedua di Bali, satu jam lebih awal dari Surabaya, akan memakai API yang sama. Apa yang terjadi pada aturan waktu dalam kontrak, dan apa yang akan Anda ubah: formatnya, zonanya, atau keduanya?

12.16 Penyelesaian Soal Latihan

1. Kontrak: "`GET /api/orders` menerima `customer_id` yang opsional; bila diberikan, hanya pesanan pelanggan itu yang didaftar; id yang bukan angka adalah 422 pada `customer_id`." Laravel: tambahkan `'customer_id' => ['nullable', 'integer']` ke aturan di `index`, dan `->when($request->integer('customer_id'), fn ($q, $id) => $q->where('customer_id', $id))` ke kuerinya. Spring: tambahkan `@RequestParam(required = false) Long customerId` ke `index`, dan metode *repository* `findByCustomerIdOrderByReceivedAtDescIdDesc(Long customerId, Pageable pageable)` dengan *entity graph* yang sama, dipilih bila `customerId` tidak `null`; Spring menjawab nilai yang bukan angka dengan galat konversi 400, sehingga untuk menaati kontrak, terimalah parameternya sebagai `String` lalu lemparkan `ApiValidationException("customer_id", ...)` bila tidak dapat diurai.

2. Pemeriksaan pertama gagal: `GET /api/services` menjawab dengan daftar polos, sehingga `$j['data'][0]` tidak ada dan pemeriksaan "S0001 first" melaporkan FAIL (Masalah 12.1). Kemudian: "a bad order → 422" gagal, karena Spring menjawab 400 dengan *problem details* tanpa `errors` (Masalah 12.3 dan 12.4); "times are YYYY-MM-DDTHH:MM:SS" dapat gagal, karena Bab 11

tidak memiliki format tetap (Masalah 12.2); "queued → departed skips a step: 422" gagal dengan 409 (Masalah 12.5); setiap pemeriksaan pembayaran gagal dengan 404 atau 405, karena tidak ada *endpoint* pembayaran (Masalah 12.6); dan pemeriksaan laporan gagal dengan 404 (Masalah 12.7). Perhatikan bahwa pemeriksa tidak berhenti pada kegagalan pertama; ia menjalankan setiap pemeriksaan, dan sebagian pemeriksaan yang bergantung pada pemeriksaan sebelumnya, seperti pemeriksaan pesanan setelah pembuatan pelanggan gagal, ikut gagal sebagai akibatnya.

3. Kontrak sebaiknya menyatakan bahwa jumlah adalah angka JSON, dan tidak lain. Koma adalah urusan cara manusia menulis angka di formulir, dan formulir adalah milik klien: aplikasi ponsel mengubah apa yang diketik sopir menjadi angka sebelum mengirimnya, persis seperti yang dilakukan `prepareForValidation` pada formulir web Laravel untuk peramban. Menerima teks akan memaksa setiap server mengurai *string* yang bergantung pada kebiasaan setempat. Spring sudah mengikuti aturan ini. Laravel boleh mempertahankan kelonggarannya, karena tidak terlihat oleh klien yang mengirim angka, tetapi kontrak tidak menjanjikannya; API Laravel yang lebih ketat akan menambahkan aturan `'quantity' => [... 'numeric' ...]` hanya untuk permintaan JSON, dan itu tidak sepadan dengan kerumitannya.

4. Pemeriksaan pertama: `[$s, $j] = call('GET', '/api/customers?phone=' . $phone); check('GET /api/customers?phone= finds the new customer', $s === 200 && count($j['data'] ?? []) === 1 && ($j['data'][0]['id'] ?? 0) === ($customer['id'] ?? -1));`. Kedua: `[$s, $j] = call('GET', $url); check('GET /api/orders/{id} carries its payments', $s === 200 && count($j['data']['payments'] ?? []) === 1);`, ditempatkan setelah pembayaran. `index` pelanggan di Laravel menjawab dengan koleksi berhalaman yang `data`-nya berupa daftar, sehingga lolos. `CustomerController` Spring Bab 12 sudah membungkus daftarnya dalam `DataView`, sehingga juga lolos; server rujukan belum memiliki daftar pelanggan dan memerlukan cabang `GET /api/customers`.

5. Tambahkan isian baru, `collected_by_method`, sebuah objek dengan kunci `cash`, `qris`, dan `transfer`, masing-masing hadir bahkan bila nilainya 0; klien lama mengabaikannya, dan `collected` tetap seperti semula, sama dengan jumlah ketiganya. Laravel: `$byMethod = Payment::query()->where('paid_at', '>=', $from)->where('paid_at', '<', $to)->toBase()->select('method', DB::raw('sum(amount) as s'))->groupBy('method')->pluck('s', 'method');` diikuti perulangan atas `PaymentMethod::cases()` yang menulis `(int) ($byMethod[$m->value] ?? 0)` untuk setiap metode, persis seperti yang dilakukan laporan untuk status.

6. Dengan zona kedua, "waktu lokal toko" menjadi taksa: pukul 08.15 di Bali adalah pukul 07.15 di Surabaya. Pertahankan formatnya dan ubahlah aturan zonanya: kirimkan setiap waktu dengan *offset*-nya, `2026-09-25T08:15:00+08:00`, sehingga waktu setiap toko tetap mudah dibaca dan setiap saat tidak taksa, atau kirimkan UTC dengan `Z` dan biarkan klien menampilkan waktu setempat. Laporan harian lalu juga memerlukan zona: "hari" toko Bali berjalan dari tengah malamnya sendiri, sehingga laporan menerima gerai (atau zonanya) sebagai parameter. Karena mengubah format isian yang sudah ada merusak klien lama, perkenalkan bentuk yang baru di `/api/v2`, atau tambahkan isian baru seperti `received_at_utc`, sementara isian lama tetap bermakna seperti semula bagi toko-toko di Surabaya.

Daftar Pustaka

[1] R. Fielding, M. Nottingham, and J. Reschke, Eds., "HTTP Semantics," RFC 9110, IETF, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9110

[2] Spring, “Exceptions,” *Spring Framework Reference Documentation: Web on Servlet Stack*. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-controller/ann-exceptionhandler.html

[3] Spring, “Projections,” *Spring Data JPA Reference Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.spring.io/spring-data/jpa/reference/repositories/projections.html

[4] Laravel, “Mocking: Interacting With Time,” *Laravel 13.x Documentation*. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/mocking#interacting-with-time

[5] Spring, “MockMvc,” *Spring Framework Reference Documentation: Testing*. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.spring.io/spring-framework/reference/testing/mockmvc.html

[6] Spring, “Modularizing Spring Boot,” spring.io blog, Oct. 28, 2025. Accessed: Sep. 25, 2026. [Online]. Available: https://spring.io/blog/2025/10/28/modularizing-spring-boot/

[7] OpenAPI Initiative, “OpenAPI Specification v3.2.0,” Sep. 2025. Accessed: Sep. 25, 2026. [Online]. Available: https://spec.openapis.org/oas/v3.2.0.html

Bab 13

Spring Boot II: Arsitektur Berlapis, Validasi, dan CRUD Lengkap dengan Thymeleaf

Bab 11 dan 12 memberi Profitina Laundry sebuah API Spring Boot yang berbicara dengan JSON yang sama seperti API Laravel: daftar harga, pelanggan, pesanan yang bergerak dari queued ke finished ke departed, pembayaran, dan laporan harian. API itu berjalan, tetapi punya dua kelemahan yang dimiliki setiap API pertama. Seluruh pekerjaan berada di controller, yang membaca basis data, menerapkan aturan toko, dan memutuskan jawaban HTTP di metode yang sama; dan tidak ada apa pun untuk orang-orang di konter, yang memerlukan halaman, bukan JSON. Bab ini memperbaiki keduanya. Di akhir bab Anda akan mampu membagi aplikasi Spring Boot menjadi lapisan controller, service, dan repository serta menyebutkan kode mana yang tinggal di mana; memindahkan aturan usaha dan transaksi dari controller ke service yang dapat dipanggil baik oleh API maupun halaman web; memvalidasi masukan formulir dengan Jakarta Bean Validation dan menampilkan setiap pesan di samping field yang tepat; membangun halaman *create, read, update,* dan *delete* yang lengkap dengan Thymeleaf; menangani "tidak ditemukan" di satu tempat; menyimpan data di SQLite di antara dua kali nyala; serta menguji lapisan service tanpa menyalakan Spring.

13.1 Titik Berangkat Kita

Di akhir Bab 12 proyek Spring Boot memiliki empat entitas (`Customer`, `Service`, `Order`, `Payment`), aturan toko dalam kelas Java biasa `LaundryRules`, repository Spring Data, dan lima controller REST. Kontrak Bab 12 memeriksanya, field demi field, terhadap API Laravel: `POST /api/orders` menjawab 201 dengan `{"data": {...}}`, pesanan yang salah menjawab 422 dengan semua galatnya sekaligus, pesanan yang sudah departed ditutup dengan 409, dan seterusnya.

Lihat lagi `OrderController.store` dari Bab 12, yang menerima pesanan baru. Dalam satu metode ia memvalidasi permintaan, mencari pelanggan dan layanan di repository-nya, memeriksa bahwa antar-jemput punya alamat pengantaran, mengumpulkan galat, membangun pesanan, menyimpannya, dan membungkus hasilnya dalam JSON. `updateStatus` membaca pesanan, memeriksa bahwa pesanan masih terbuka, menanyakan langkah berikutnya kepada `LaundryRules`, membandingkannya dengan status yang diminta klien, dan mencatat waktunya. Setiap metode membawa `@Transactional`. Bab 11 sudah mengingatkan bahwa cara ini dapat diterima untuk API pertama yang kecil dan menjanjikan bahwa Bab 13 akan memindahkan pekerjaan itu ke *lapisan service*. Janji itu kini jatuh tempo, karena Bu Wati menginginkan sesuatu yang tidak dapat diberikan oleh API saja.

Outlet Wiyung, yang dibuka Januari 2026, masih menerima pesanan di atas kertas. Konternya punya komputer kecil dengan peramban, dan Bu Wati ingin layar konter yang sama seperti di outlet lain: daftar pelanggan yang dapat dicari, formulir untuk menambah pelanggan atau membetulkan nomor telepon, formulir untuk menerima sekantong cucian, antrean pesanan, dan halaman pesanan yang menampilkan total, pembayaran, dan sisa tagihan, dengan tombol yang

memajukan pesanan dan menolak menyerahkan cucian yang belum dibayar. Kantor akuntan yang tersambung ke API di Bab 11 harus tetap bekerja persis seperti sebelumnya.

Kalau halaman ditulis dengan cara Bab 12 menulis API, setiap aturan akan ada dua kali: sekali di controller JSON dan sekali di controller halaman. Saat pertama kali seseorang mengubah aturan antar-jemput di satu tempat dan lupa di tempat lain, konter Wiyung dan aplikasi ponsel akan berbeda pendapat tentang harga kantong cucian yang sama. Karena itu bab ini mengerjakan semuanya dalam urutan yang benar: lebih dulu lapisannya (Subbab 13.2 sampai 13.5), lalu halaman di atasnya (Subbab 13.6 sampai 13.8), lalu SQLite, menjalankan, dan menguji (Subbab 13.9 dan 13.10). Kodenya ada di `Chapter13/spring`; ia bermula sebagai salinan proyek Bab 12.

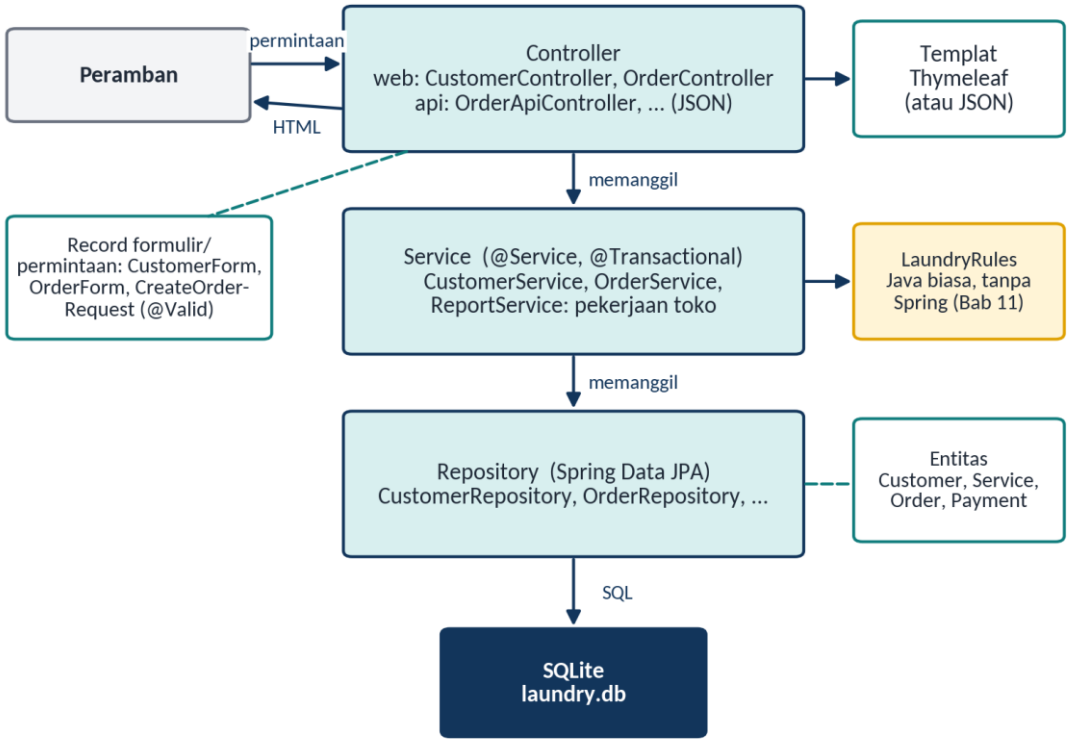
13.2 Gagasannya dalam Bahasa Sehari-hari

13.2.1 Satu kelas yang mengerjakan segalanya

Metode controller yang membaca basis data, menerapkan aturan, memulai transaksi, dan memilih status HTTP cepat ditulis dan mudah diikuti selama hanya ada satu. Masalahnya dimulai dengan pemanggil kedua. Halaman penerima pesanan memerlukan setiap pemeriksaan yang dilakukan `store`, tetapi harus menjawab dengan formulir dan pesan merah, bukan JSON dan status 422. Tugas malam yang menutup pesanan yang terlupa memerlukan aturan alur kerja, tetapi sama sekali tidak punya permintaan HTTP. Setiap pemanggil baru menyalin aturan, dan salinan-salinan itu lama-lama saling menjauh. Pengujian juga sulit: untuk memeriksa “cucian hanya keluar bila sudah dibayar”, sebuah tes harus membangun permintaan HTTP, menyalakan lapisan web dan basis data, lalu membaca jawaban dari JSON atau HTML. Tes yang lambat adalah tes yang lama-lama tidak dijalankan orang.

13.2.2 Tiga lapisan, satu arah

Jawaban bakunya, yang dipakai ketiga kerangka kerja di buku ini, adalah membagi aplikasi menjadi *lapisan*, masing-masing dengan satu jenis tanggung jawab, dan membiarkan setiap lapisan hanya memanggil lapisan di bawahnya. **Gambar 13.1** menunjukkan lapisan-lapisan aplikasi bab ini.



Gambar 13.1 Lapisan-lapisan aplikasi laundry Spring Boot: controller halaman dan API berbicara HTTP, service mengerjakan pekerjaan toko di dalam transaksi, repository berbicara SQL; aturannya sendiri adalah Java biasa dari Bab 11.

> **DEFINISI.** Dalam *arsitektur berlapis*, lapisan **controller** (presentasi) mengubah permintaan HTTP menjadi pemanggilan metode dan hasilnya menjadi HTML atau JSON; lapisan **service** menjalankan satu operasi usaha per metode, menerapkan aturan usaha, dan menentukan di mana transaksi basis data dimulai dan berakhir; lapisan **repository** (persistensi) memuat dan menyimpan entitas. Pemanggilan hanya mengarah ke bawah: controller memanggil service, service memanggil repository, dan tidak pernah sebaliknya.

Gagasan ini lebih tua daripada ketiga kerangka kerja itu. Fowler menyebutnya pola *Service Layer*: lapisan service yang menetapkan batas aplikasi, menawarkan operasi-operasi yang didukungnya, dan mengoordinasikan tanggapan aplikasi dalam setiap operasi [1]. Kosakata Spring mengikutinya: kelas beranotasi `@Controller` atau `@RestController` ditemukan oleh lapisan web, kelas beranotasi `@Service` adalah *bean* biasa yang boleh dipanggil *bean* lain, dan antarmuka yang memperluas `JpaRepository` diimplementasikan untuk Anda oleh Spring Data [2], [3].

Akibat yang menarik ada di bagian atas **Gambar 13.1**. Begitu pekerjaan tinggal di service, dua jenis controller dapat berdampingan: controller JSON dari Bab 12, yang kini tipis, dan controller halaman baru yang merender templat Thymeleaf. Keduanya memanggil `OrderService.create` yang sama, sehingga keduanya menerapkan aturan yang sama dengan sendirinya. Di sebelah kanan ada `LaundryRules`, tidak berubah sejak Bab 11: Java biasa yang dipanggil service, dan dapat diperiksa hanya dengan JDK.

13.2.3 Apa yang boleh diketahui setiap lapisan

Cara yang baik untuk memutuskan di mana sebaris kode harus tinggal adalah bertanya apa yang perlu *diketahuinya*.

Tabel 13.1 Apa yang boleh diketahui setiap lapisan, dan apa yang tidak boleh.

Lapisan	Tahu tentang	Tidak boleh tahu tentang	Contoh di bab ini
Controller	URL, parameter permintaan, JSON dan formulir, *redirect*, templat mana yang ditampilkan	SQL, transaksi, harga	`OrderController.advance`
Service	operasi dan aturan toko, transaksi	HTML, URL, kode status HTTP	`OrderService.advance`
Repository	tabel, kueri	aturan, HTTP	`OrderRepository.billedBetween`
Aturan	angka, status	semua yang lain	`LaundryRules.advance`

Ingat **Tabel 13.1** sepanjang sisa bab: ikuti satu klik pada "Hand over to customer" dari controller halaman ke service ke repository, dan pada setiap langkah kodenya hanya mengerjakan apa yang diizinkan barisnya.

13.2.4 Transaksi: semua atau tidak sama sekali

Menerima pembayaran melibatkan dua langkah: mencari tahu berapa yang masih terutang, lalu menyimpan pembayarannya. Bila dua kasir menerima pembayaran untuk pesanan yang sama pada saat yang sama, keduanya bisa membaca "terutang Rp 34.000" dan keduanya menyimpan Rp 34.000, sehingga pelanggan membayar dua kali. *Transaksi* mengelompokkan operasi basis data agar berhasil atau gagal bersama-sama, dan basis data mengisolasinya dari transaksi lain yang berjalan bersamaan. Di Spring Anda menandai sebuah metode, atau seluruh kelas, dengan `@Transactional`; Spring memulai transaksi sebelum metode berjalan, melakukan *commit* ketika metode selesai dengan normal, dan melakukan *rollback* ketika metode melempar eksepsi *unchecked* [2]. Satu transaksi sebaiknya mencakup satu operasi usaha, dan operasi usaha persis adalah sebuah metode service. Itulah sebabnya bab ini mencabut `@Transactional` dari setiap controller dan memasangnya di service.

> **WAWASAN KUNCI:** Lapisan service adalah tempat tiga hal bertemu: satu operasi usaha, satu kumpulan aturan, dan satu transaksi. Bila Anda ragu di mana sebuah kode harus tinggal, tanyakan "apakah ini bagian dari apa yang *dikerjakan* toko?" Bila ya, kode itu masuk ke service.

13.3 Di Mana Kode Ditempatkan

Proyek Bab 12 menaruh controller JSON-nya di paket bernama `web`. Dengan datangnya halaman, `web` menjadi rumah controller halaman, dan controller JSON pindah ke paket baru `api`. **Tabel 13.2** adalah peta lengkapnya.

Tabel 13.2 Paket-paket proyek Bab 13.

Paket	Isi	Baru di bab ini?
`domain`	`Customer`, `Service`, `Order`, `Payment`, status, `LaundryRules`	hanya `Customer.change`
`repository`	antarmuka Spring Data	beberapa metode kueri
`service`	`CustomerService`, `OrderService`, `ReportService`, tiga eksepsi	semuanya baru
`api`	`OrderApiController` dan controller JSON lain, *record* permintaan dan tampilan, `ApiExceptionHandler`	pindah dari `web`, kini tipis
`web`	controller halaman, *record* formulir, `Forms`, `WebErrorAdvice`	semuanya baru

Controller JSON mendapat akhiran `Api`: `OrderApiController`, `CustomerApiController`, `ServiceApiController`, `ReportApiController`. Ini bukan sekadar kerapian. Spring menamai *bean* menurut kelasnya, `orderController` untuk `OrderController`, dan dua kelas dengan nama sederhana yang sama di paket berbeda akan sama-sama mengklaim nama itu, sehingga aplikasi menolak menyala. URL, JSON, dan kontrak Bab 12 sama sekali tidak berubah.

13.4 Lapisan Service

13.4.1 Tiga eksepsi

Service harus dapat berkata "tidak" tanpa tahu apakah pemanggilnya akan mengubah jawaban itu menjadi JSON atau pesan merah di halaman. Service melakukannya dengan tiga eksepsi (**Kode 13.1**, **Kode 13.2**, dan **Kode 13.3**).

Kode 13.1 `RuleException.java`: masukan yang melanggar aturan, dengan pesan per field.

```
package com.profitina.laundry.service;

import java.util.List;
import java.util.Map;

/**
 * Masukan yang melanggar aturan toko, per field. API mengubahnya
 * menjadi 422 {"errors": {...}}; halaman menampilkan pesan di field-nya.
 */
public class RuleException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    private final transient Map<String, List<String>> errors;

    public RuleException(Map<String, List<String>> errors) {
        super("The given data was invalid.");
        this.errors = errors;
    }

    public RuleException(String field, String message) {
        this(Map.of(field, List.of(message)));
    }

    public Map<String, List<String>> getErrors() {
        return errors;
    }
}
```

Kode 13.2 `NotFoundException.java`: record yang tidak ada.

```
package com.profitina.laundry.service;

/** Meminta record yang tidak ada: 404 di API dan di halaman. */
public class NotFoundException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public NotFoundException(String what, Object id) {
        super("No " + what + " " + id + ".");
    }
}
```

Kode 13.3 `ConflictException.java`: permintaan yang tidak diizinkan oleh keadaan record saat ini.

```
package com.profitina.laundry.service;

/** Bentuknya benar, tetapi tidak mungkin pada keadaan record kini: 409. */
public class ConflictException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public ConflictException(String message) {
        super(message);
    }
}
```

`RuleException` adalah `ApiValidationException` Bab 12 dengan nama yang tidak lagi menyebut API: ia membawa *map* dari nama field ke pesan-pesan, persis objek `errors` dalam kontrak. Nama field-nya adalah nama snake_case milik API (`customer_id`), karena aturannya ditulis untuk API lebih dulu; Subbab 13.6 memperlihatkan bagaimana halaman menerjemahkannya. `NotFoundException` membangun kalimat yang sama dengan Bab 12 (“No order 7.”), dan `ConflictException` menangani satu-satunya 409 dalam kontrak, pesanan yang sudah ditutup, ditambah satu yang baru untuk pelanggan di Subbab 13.4.2. Ketiganya memperluas `RuntimeException`. Ini penting dua kali: eksepsi *unchecked* tidak memerlukan klausa `throws` di sepanjang jalan ke atas, dan Spring secara bawaan hanya melakukan *rollback* untuk eksepsi *unchecked* [2], sehingga aturan yang gagal di tengah metode service membatalkan apa pun yang sudah ditulis metode itu.

13.4.2 Pelanggan

Halaman harus dapat mengubah dan menghapus pelanggan, hal yang tidak pernah diperlukan API. Entitasnya mendapat satu metode, `change` (**Kode 13.4**); waktu pembuatan pelanggan tetap tidak punya *setter* sama sekali, karena tidak pernah berubah.

Kode 13.4 `Customer.java`: pelanggan dari Bab 11 dengan satu metode baru, `change`.

```
package com.profitina.laundry.domain;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import java.time.LocalDateTime;

@Entity
@Table(name = "customers")
public class Customer {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, length = 80)
    private String name;

    @Column(nullable = false, length = 20, unique = true)
    private String phone;

    @Column(length = 200)
    private String address;

    @Column(nullable = false)
    private LocalDateTime createdAt;

    protected Customer() {                // hanya untuk JPA
    }

    public Customer(String name, String phone, String address,
                    LocalDateTime createdAt) {
        this.name = name;
        this.phone = phone;
        this.address = address;
        this.createdAt = createdAt;
    }
```

Kode 13.4 (Lanjutan) *Customer.java: pelanggan dari Bab 11 dengan satu metode baru, change.*

```
    /** Satu-satunya field yang boleh diubah formulir; created_at tetap. */
    public void change(String name, String phone, String address) {
        this.name = name;
        this.phone = phone;
        this.address = address;
    }

    public Long getId() { return id; }
    public String getName() { return name; }
    public String getPhone() { return phone; }
    public String getAddress() { return address; }
    public LocalDateTime getCreatedAt() { return createdAt; }
}
```

Repository-nya mendapat tiga kueri turunan (**Kode 13.5**).

Kode 13.5 `CustomerRepository.java`: kueri yang diturunkan dari nama metode.

```
package com.profitina.laundry.repository;

import com.profitina.laundry.domain.Customer;
import java.util.List;
import java.util.Optional;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CustomerRepository extends JpaRepository<Customer, Long> {

    Optional<Customer> findByPhone(String phone);

    boolean existsByPhone(String phone);

    boolean existsByPhoneAndIdNot(String phone, Long id);

    List<Customer> findByPhoneContainingOrderByName(String digits);

    List<Customer> findByNameContainingIgnoreCaseOrderByName(String name);
}
```

`existsByPhoneAndIdNot` menjawab pertanyaan keunikan saat mengubah data: ketika pelanggan 5 disimpan dengan sebuah nomor telepon, tidak boleh ada pelanggan *lain* yang memilikinya. `findByNameContainingIgnoreCaseOrderByName` menjadi `where lower(name) like lower(?)` dengan argumen yang diapit tanda `%`, diikat sebagai parameter, sehingga pelanggan bernama `' or 1=1 --` hanyalah nama yang aneh [3]. **Kode 13.6** adalah service-nya.

Kode 13.6 `CustomerService.java`: mencari, membuat, mengubah, dan menghapus pelanggan, di dalam transaksi.

```
package com.profitina.laundry.service;

import com.profitina.laundry.domain.Customer;
import com.profitina.laundry.repository.CustomerRepository;
import com.profitina.laundry.repository.OrderRepository;
import java.time.Clock;
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;
import java.util.List;
import java.util.Optional;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

/** Semua yang dilakukan toko terhadap pelanggan, di satu tempat. */
@Service
@Transactional
public class CustomerService {

    private final CustomerRepository customers;
    private final OrderRepository orders;
    private final Clock clock;

    public CustomerService(CustomerRepository customers,
                           OrderRepository orders, Clock clock) {
        this.customers = customers;
        this.orders = orders;
        this.clock = clock;
    }

    /** Menurut nama, atau menurut telepon bila teksnya angka. */
    @Transactional(readOnly = true)
    public List<Customer> search(String text) {
        String q = text == null ? "" : text.trim();
        if (q.isEmpty()) {
            return customers.findAll(Sort.by("name"));
        }
        String digits = q.replaceAll("[^0-9]", "");
        return q.matches("[0-9 +-]+")
                ? customers.findByPhoneContainingOrderByName(digits)
                : customers.findByNameContainingIgnoreCaseOrderByName(q);
    }
```

Kode 13.6 (Lanjutan) *CustomerService.java: mencari, membuat, mengubah, dan menghapus pelanggan, di dalam transaksi.*

```
    @Transactional(readOnly = true)
    public Optional<Customer> findByPhone(String phone) {
        return customers.findByPhone(phone.replaceAll("[^0-9]", ""));
    }

    @Transactional(readOnly = true)
    public Customer get(Long id) {
        return customers.findById(id)
                .orElseThrow(() -> new NotFoundException("customer", id));
    }

    public Customer create(String name, String phone, String address) {
        if (customers.existsByPhone(phone)) {     // Laravel: aturan unique
            throw new RuleException("phone",
                    "The phone has already been taken.");
        }
        return customers.save(new Customer(name.trim(), phone,
                blankToNull(address), now()));
    }

    public Customer update(Long id, String name, String phone,
                           String address) {
        Customer customer = get(id);
        if (customers.existsByPhoneAndIdNot(phone, id)) {
            throw new RuleException("phone",
                    "The phone has already been taken.");
        }
        customer.change(name.trim(), phone, blankToNull(address));
        return customer;            // disimpan saat commit (dirty checking)
    }

    public void delete(Long id) {
        Customer customer = get(id);
        if (orders.existsByCustomerId(id)) {
            throw new ConflictException(customer.getName()
                    + " has orders and cannot be deleted.");
        }
        customers.delete(customer);
    }

    private LocalDateTime now() {
        return LocalDateTime.now(clock).truncatedTo(ChronoUnit.SECONDS);
    }
```

Kode 13.6 (Lanjutan) *CustomerService.java: mencari, membuat, mengubah, dan menghapus pelanggan, di dalam transaksi.*

```
    private static String blankToNull(String s) {
        return s == null || s.isBlank() ? null : s.trim();
    }
}
```

Konstruktornya menerima dua repository dan sebuah `Clock`. Ia tidak membuatnya sendiri: Spring melihat bahwa `CustomerService` adalah `@Service` dengan satu konstruktor, mencari *bean* dari ketiga tipe itu, dan memberikannya. Inilah *constructor injection*, bentuk injeksi dependensi yang dianjurkan Bab 2 dan yang membuat kelas mudah diuji dengan pengganti. `Clock` adalah *bean* baru (**Kode 13.7**); service menanyakan waktu kepadanya alih-alih memanggil `LocalDateTime.now()`, sehingga sebuah tes dapat menetapkan waktu menjadi "25 September 2026, 09.00 di Surabaya".

Kode 13.7 `LaundryApplication.java`: titik masuk, dan jam toko dalam Waktu Indonesia Barat.

```
package com.profitina.laundry;

import java.time.Clock;
import java.time.ZoneId;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class LaundryApplication {

    public static void main(String[] args) {
        SpringApplication.run(LaundryApplication.class, args);
    }

    /** Jam toko, waktu Surabaya; tes dapat memberi jam yang tetap. */
    @Bean
    Clock clock() {
        return Clock.system(ZoneId.of("Asia/Jakarta"));
    }
}
```

`@Transactional` pada tingkat kelas menjadikan setiap metode publik satu transaksi; `search`, `findByPhone`, dan `get` menyempitkannya menjadi `readOnly = true`, petunjuk yang membuat Hibernate tidak perlu mencari perubahan. Metode-metodenya menerima nilai biasa, bukan objek formulir: service tidak boleh tahu apakah ia dipanggil dari halaman atau dari JSON, dan itu menjaga dependensi tetap mengarah ke bawah, seperti dalam **Gambar 13.1**.

`update` tampak seolah lupa menyimpan. Ia memuat pelanggan, memanggil `change`, lalu kembali. Inilah *dirty checking* JPA: entitas yang dimuat di dalam transaksi bersifat *managed*, dan ketika transaksi di-*commit*, Hibernate membandingkannya dengan keadaan saat dimuat dan menulis `UPDATE` untuk apa pun yang berubah. `delete` melindungi riwayat toko: pelanggan yang punya pesanan tidak dapat dihapus, karena pesanan-pesanannya akan kehilangan pemilik. Basis data juga akan menolak, melalui *foreign key*, tetapi dengan pesan yang tidak dapat dibaca kasir mana pun; service menolak lebih dulu, dengan sebuah kalimat, sebagai `ConflictException`.

13.4.3 Pesanan

Kode 13.8 menambahkan `existsByCustomerId` ke repository pesanan Bab 12; sisanya tidak berubah, termasuk *entity graph* yang memuat pelanggan dan layanan sebuah pesanan dalam kueri yang sama.

Kode 13.8 `OrderRepository.java`: halaman-halaman pesanan, satu pesanan beserta rinciannya, dan kueri laporan.

```
package com.profitina.laundry.repository;

import com.profitina.laundry.domain.Order;
import com.profitina.laundry.domain.OrderStatus;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.EntityGraph;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

public interface OrderRepository extends JpaRepository<Order, Long> {

    @EntityGraph(attributePaths = {"customer", "service"})
    Page<Order> findAllByOrderByReceivedAtDescIdDesc(Pageable pageable);

    @EntityGraph(attributePaths = {"customer", "service"})
    Page<Order> findByStatusOrderByReceivedAtDescIdDesc(OrderStatus status,
                                                        Pageable pageable);

    @EntityGraph(attributePaths = {"customer", "service", "payments"})
    Optional<Order> findWithDetailsById(Long id);

    boolean existsByCustomerId(Long customerId);

    /** Satu baris per status: bentuk yang diperlukan laporan harian. */
    interface StatusCount {
        OrderStatus getStatus();

        long getN();
    }

    @Query("""
            select o.status as status, count(o) as n
            from Order o
            where o.receivedAt >= :from and o.receivedAt < :to
            group by o.status""")
    List<StatusCount> countByStatusBetween(@Param("from") LocalDateTime from,
                                           @Param("to") LocalDateTime to);
```

Kode 13.8 (Lanjutan) *OrderRepository.java: halaman-halaman pesanan, satu pesanan beserta rinciannya, dan kueri laporan.*

```
    @Query("""
            select coalesce(sum(o.total), 0)
            from Order o
            where o.receivedAt >= :from and o.receivedAt < :to""")
    long billedBetween(@Param("from") LocalDateTime from,
                       @Param("to") LocalDateTime to);
}
```

Kode 13.9 adalah tempat pekerjaan toko kini berlangsung.

Kode 13.9 `OrderService.java`: menerima pesanan, memajukannya, dan menerima uang, dengan setiap aturan di satu tempat.

```java
package com.profitina.laundry.service;

import com.profitina.laundry.domain.Customer;
import com.profitina.laundry.domain.LaundryRules;
import com.profitina.laundry.domain.Order;
import com.profitina.laundry.domain.OrderStatus;
import com.profitina.laundry.domain.PaymentMethod;
import com.profitina.laundry.domain.Service;
import com.profitina.laundry.repository.CustomerRepository;
import com.profitina.laundry.repository.OrderRepository;
import com.profitina.laundry.repository.PaymentRepository;
import com.profitina.laundry.repository.ServiceRepository;
import java.math.BigDecimal;
import java.time.Clock;
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.transaction.annotation.Transactional;

/** Menerima pesanan, memajukannya, menerima uang: pekerjaan toko. */
// Ditulis lengkap: entitas "Service" sudah memakai nama pendeknya.
@org.springframework.stereotype.Service
@Transactional
public class OrderService {

    /** Yang diperlukan pesanan baru, dari formulir maupun JSON. */
    public record NewOrder(Long customerId, String serviceId,
                           BigDecimal quantity, boolean pickup,
                           String deliveryAddress, String note) {
    }

    private final OrderRepository orders;
    private final CustomerRepository customers;
    private final ServiceRepository services;
    private final PaymentRepository payments;
    private final Clock clock;
```

Kode 13.9 (Lanjutan) *OrderService.java: menerima pesanan, memajukannya, dan menerima uang, dengan setiap aturan di satu tempat.*

```
    public OrderService(OrderRepository orders, CustomerRepository customers,
                        ServiceRepository services,
                        PaymentRepository payments, Clock clock) {
        this.orders = orders;
        this.customers = customers;
        this.services = services;
        this.payments = payments;
        this.clock = clock;
    }

    /** Nomor halaman bagi manusia mulai dari 1; Spring Data dari 0. */
    @Transactional(readOnly = true)
    public Page<Order> page(OrderStatus status, int page, int size) {
        PageRequest request = PageRequest.of(Math.max(page, 1) - 1, size);
        return status == null
                ? orders.findAllByOrderByReceivedAtDescIdDesc(request)
                : orders.findByStatusOrderByReceivedAtDescIdDesc(status,
                        request);
    }

    @Transactional(readOnly = true)
    public Order get(Long id) {
        return orders.findWithDetailsById(id)
                .orElseThrow(() -> new NotFoundException("order", id));
    }

    @Transactional(readOnly = true)
    public List<Service> activeServices() {
        return services.findByActiveTrueOrderByIdAsc();
    }

    @Transactional(readOnly = true)
    public Service service(String id) {
        return services.findById(id)
                .orElseThrow(() -> new NotFoundException("service", id));
    }
```

Kode 13.9 (Lanjutan) *OrderService.java: menerima pesanan, memajukannya, dan menerima uang, dengan setiap aturan di satu tempat.*

```
    /** Aturan yang memerlukan basis data; map kosong berarti semua beres. */
    @Transactional(readOnly = true)
    public Map<String, List<String>> check(NewOrder in) {
        Map<String, List<String>> errors = new LinkedHashMap<>();
        if (in.customerId() != null
                && !customers.existsById(in.customerId())) {
            add(errors, "customer_id", "The selected customer is invalid.");
        }
        if (in.serviceId() != null && !in.serviceId().isBlank()
                && services.findById(in.serviceId())
                        .filter(Service::isActive).isEmpty()) {
            add(errors, "service_id", "The selected service is invalid.");
        }
        if (in.pickup() && (in.deliveryAddress() == null
                || in.deliveryAddress().isBlank())) {
            add(errors, "delivery_address",
                    "Where should the laundry be delivered?");
        }
        return errors;
    }

    public Order create(NewOrder in) {
        Map<String, List<String>> errors = check(in);
        if (!errors.isEmpty()) {
            throw new RuleException(errors);
        }
        Customer customer = customers.findById(in.customerId()).orElseThrow();
        Service service = services.findById(in.serviceId()).orElseThrow();
        return orders.save(new Order(customer, service, in.quantity(),
                in.pickup(), in.pickup() ? in.deliveryAddress() : null,
                in.note(), now()));
    }

    /**
     * Satu langkah queued -> finished -> departed. Bila status yang diminta
     * disebut (API selalu menyebutnya), harus tepat langkah berikutnya.
     */
    public Order advance(Long id, OrderStatus wanted) {
        Order order = get(id);
        ensureOpen(order);
        OrderStatus next;
        try {
            next = LaundryRules.advance(order.getStatus(), order.balance());
        } catch (IllegalStateException e) {
            throw new RuleException("status", e.getMessage());
        }
```

Kode 13.9 (Lanjutan) *OrderService.java: menerima pesanan, memajukannya, dan menerima uang, dengan setiap aturan di satu tempat.*

```
        if (wanted != null && wanted != next) {
            throw new RuleException("status", "An order that is "
                    + order.getStatus().value() + " can only become "
                    + next.value() + ".");
        }
        order.advance(now());
        return order;
    }

    public Order pay(Long id, long amount, PaymentMethod method) {
        Order order = get(id);
        ensureOpen(order);
        if (amount > order.balance()) {
            throw new RuleException("amount", "Only "
                    + LaundryRules.rupiah(order.balance())
                    + " is still owed.");
        }
        payments.save(order.addPayment(amount, method, now()));
        return order;
    }

    /** Cucian yang sudah keluar toko tidak menerima langkah atau uang lagi. */
    private static void ensureOpen(Order order) {
        if (order.getStatus() == OrderStatus.DEPARTED) {
            throw new ConflictException("Order #" + order.getId()
                    + " has left the shop and is closed.");
        }
    }

    private static void add(Map<String, List<String>> errors, String field,
                            String message) {
        errors.computeIfAbsent(field, k -> new ArrayList<>()).add(message);
    }

    private LocalDateTime now() {
        return LocalDateTime.now(clock).truncatedTo(ChronoUnit.SECONDS);
    }
}
```

Hal pertama yang patut diperhatikan adalah anotasi yang ditulis lengkap, `@org.springframework.stereotype.Service`. Bab 11 sudah menjumpai bentrokan yang sama: entitas `Service` memakai nama pendek itu di berkas ini, dan Java tidak dapat mengimpor dua kelas berbeda bernama `Service` ke dalam satu berkas. Komentar di listing menjelaskan alasannya, supaya pembaca berikutnya tidak "memperbaikinya".

`NewOrder` adalah *record* kecil yang menggambarkan pesanan baru tanpa bergantung pada asalnya; controller JSON membangunnya dari `CreateOrderRequest`, controller halaman dari sebuah formulir. Aturan-aturan yang memerlukan basis data tinggal di `check`, yang mengembalikan *map* galat alih-alih melempar eksepsi. Desain ini menepati janji kontrak: pesanan yang salah dilaporkan dengan *semua* galatnya sekaligus, yang ditemukan Bean Validation (pelanggan kosong, jumlah 0) bersama-sama dengan yang hanya dapat ditemukan basis data (pelanggan yang tidak ada, layanan yang sudah ditarik) dan aturan antar-jemput. Controller menggabungkan kedua daftar dan menjawab sekali. `create` memanggil `check` lagi dan melempar eksepsi bila ada yang salah, sehingga pemanggil yang lupa memeriksa tidak dapat menyelipkan pesanan yang salah melewati service.

`advance` adalah `updateStatus` lama tanpa HTTP. Ia memuat pesanan, menolak pesanan yang sudah ditutup dengan `ConflictException`, menanyakan langkah berikutnya kepada

`LaundryRules.advance` (yang juga menolak mengeluarkan cucian yang belum dibayar), dan, bila pemanggil menyebut status yang diinginkan, bersikeras bahwa status itu tepat langkah berikutnya. API selalu menyebutnya (`{"status": "finished"}`); tombol di halaman tidak, dan mengirim `null`, artinya "apa pun langkah berikutnya". `pay` menolak jumlah di atas sisa tagihan dan mencatat pembayarannya. Karena setiap metode berjalan dalam satu transaksi, membaca sisa tagihan dan menyisipkan pembayaran terjadi bersama-sama.

Uang masih ditangani oleh aturan Bab 11, dan aturan itu layak dinyatakan ulang, karena setiap halaman di bab ini menampilkan hasilnya. Bila q adalah jumlah dan $h = 100q$ jumlah dalam perseratus (4,5 kg adalah 450), perseratus yang ditagih b adalah $\max(h, 300)$ untuk layanan per kilogram dan h yang dibulatkan ke atas menjadi satuan utuh untuk yang lain. Dengan harga p per satuan, subtotal dan totalnya adalah

$$S = \left\lfloor \frac{p \cdot b + 50}{100} \right\rfloor, \qquad T = S + \begin{cases} 10000 & \text{pickup and } S < 100000 \\ 0 & \text{otherwise} \end{cases} \tag{13.1}$$

sehingga 2,5 kg cuci-lipat seharga Rp 8.000 dengan antar-jemput berharga $8000 \times 300/100 = 24000$ ditambah biaya Rp 10.000, yaitu Rp 34.000. Bila pembayaran-pembayarannya berjumlah P, sisa tagihan yang ditampilkan halaman pesanan adalah

$$B = T - P \tag{13.2}$$

dan service tidak pernah membiarkan P melampaui T, sehingga B tidak pernah negatif.

13.4.4 Laporan harian

Kode 13.10 memindahkan aritmetika laporan Bab 12 ke sebuah service.

Kode 13.10 `ReportService.java`: angka-angka satu hari.

```
package com.profitina.laundry.service;

import com.profitina.laundry.domain.OrderStatus;
import com.profitina.laundry.repository.OrderRepository;
import com.profitina.laundry.repository.PaymentRepository;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.EnumMap;
import java.util.Map;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

/** Angka satu hari, untuk laporan pemilik. */
@Service
@Transactional(readOnly = true)
public class ReportService {

    /** Pesanan per status, uang ditagih dan uang diterima pada satu hari. */
    public record DayFigures(LocalDate day, Map<OrderStatus, Long> byStatus,
                             long billed, long collected) {

        public long received() {
            return byStatus.values().stream().mapToLong(Long::longValue).sum();
        }
    }

    private final OrderRepository orders;
    private final PaymentRepository payments;

    public ReportService(OrderRepository orders, PaymentRepository payments) {
        this.orders = orders;
        this.payments = payments;
    }

    public DayFigures daily(LocalDate day) {
        LocalDateTime from = day.atStartOfDay();
        LocalDateTime to = from.plusDays(1);   // [from, to): satu hari penuh
        Map<OrderStatus, Long> byStatus = new EnumMap<>(OrderStatus.class);
        for (OrderStatus s : OrderStatus.values()) {
            byStatus.put(s, 0L);               // setiap status, walau 0
        }
        for (OrderRepository.StatusCount c
                : orders.countByStatusBetween(from, to)) {
            byStatus.put(c.getStatus(), c.getN());
        }
        return new DayFigures(day, byStatus, orders.billedBetween(from, to),
```

Kode 13.10 (Lanjutan) *ReportService.java: angka-angka satu hari.*

```
                payments.collectedBetween(from, to));
    }
}
```

Service ini mengembalikan `DayFigures`, *record* yang memakai enum `OrderStatus` sebagai kunci *map*; mengubah enum menjadi kata `queued`, `finished`, dan `departed` dalam JSON adalah tugas controller. `EnumMap` menjaga status dalam urutan alaminya, sehingga laporan selalu mencantumkannya dengan cara yang sama.

13.5 Controller API yang Tipis

Dengan pekerjaan berada di service, controller JSON menjadi pendek. **Kode 13.11** adalah controller pesanan; bandingkan dengan milik Bab 12.

Kode 13.11 `OrderApiController.java`: baca JSON, panggil `OrderService`, tulis JSON.

```
package com.profitina.laundry.api;

import com.profitina.laundry.domain.OrderStatus;
import com.profitina.laundry.domain.PaymentMethod;
import com.profitina.laundry.service.OrderService;
import com.profitina.laundry.service.OrderService.NewOrder;
import com.profitina.laundry.service.RuleException;
import jakarta.validation.ConstraintViolation;
import jakarta.validation.Valid;
import jakarta.validation.Validator;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PatchMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

/** Tipis: baca JSON, panggil OrderService, tulis JSON. Tanpa aturan/SQL. */
@RestController
@RequestMapping("/api/orders")
public class OrderApiController {

    private final OrderService orders;
    private final Validator validator;

    public OrderApiController(OrderService orders, Validator validator) {
        this.orders = orders;
        this.validator = validator;
    }

    @GetMapping
    public PageView<OrderView> index(
            @RequestParam(required = false) String status,
            @RequestParam(defaultValue = "1") int page) {
        OrderStatus wanted = status == null ? null : parseStatus(status);
        return PageView.of(orders.page(wanted, page, 20), OrderView::of);
    }
```

Kode 13.11 (Lanjutan) *OrderApiController.java: baca JSON, panggil OrderService, tulis JSON.*

```
    @GetMapping("/{id}")
    public DataView<OrderView> show(@PathVariable Long id) {
        return new DataView<>(OrderView.withPayments(orders.get(id)));
    }

    /** Setiap aturan yang dilanggar dilaporkan sekaligus, seperti Laravel. */
    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public DataView<OrderView> store(@RequestBody CreateOrderRequest in) {
        NewOrder order = new NewOrder(in.customerId(), in.serviceId(),
                in.quantity(), in.pickup(), in.deliveryAddress(), in.note());
        Map<String, List<String>> errors = new LinkedHashMap<>();
        for (ConstraintViolation<CreateOrderRequest> v
                : validator.validate(in)) {
            errors.computeIfAbsent(ApiExceptionHandler.snake(
                    v.getPropertyPath().toString()), k -> new ArrayList<>())
                    .add(v.getMessage());
        }
        orders.check(order).forEach((field, messages) -> errors
                .computeIfAbsent(field, k -> new ArrayList<>())
                .addAll(messages));
        if (!errors.isEmpty()) {
            throw new RuleException(errors);
        }
        return new DataView<>(OrderView.of(orders.create(order)));
    }

    @PatchMapping("/{id}/status")
    public DataView<OrderView> updateStatus(@PathVariable Long id,
            @Valid @RequestBody StatusRequest in) {
        return new DataView<>(OrderView.of(
                orders.advance(id, parseStatus(in.status()))));
    }

    @PostMapping("/{id}/payments")
    @ResponseStatus(HttpStatus.CREATED)
    public DataView<OrderView> pay(@PathVariable Long id,
            @Valid @RequestBody CreatePaymentRequest in) {
        PaymentMethod method;
        try {
            method = PaymentMethod.fromValue(in.method());
        } catch (IllegalArgumentException e) {
            throw new RuleException("method",
                    "The selected method is invalid.");
        }
        return new DataView<>(OrderView.withPayments(
```

Kode 13.11 (Lanjutan) *OrderApiController.java: baca JSON, panggil OrderService, tulis JSON.*

```
                orders.pay(id, in.amount(), method)));
    }

    private static OrderStatus parseStatus(String value) {
        try {
            return OrderStatus.fromValue(value);
        } catch (IllegalArgumentException e) {
            throw new RuleException("status",
                    "The selected status is invalid.");
        }
    }
}
```

`store` memperlihatkan validasi yang digabungkan. `Validator` Spring memeriksa batasan Bean Validation pada `CreateOrderRequest` (*record* itu tidak berubah dari Bab 12), `orders.check`

menambahkan aturan basis data, dan hanya bila keduanya kosong `create` dijalankan. `ApiExceptionHandler.snake` mengubah `customerId`, nama Java sebuah komponen *record*, menjadi `customer_id`, nama yang dikirim klien. Tidak ada apa pun di kelas ini yang memulai transaksi, menyentuh repository, atau mengetahui biaya antar-jemput. Tiga controller JSON lainnya sama tipisnya (**Kode 13.12**, **Kode 13.13**, **Kode 13.14**).

Kode 13.12 `CustomerApiController.java`: pelanggan dalam JSON.

```
package com.profitina.laundry.api;

import com.profitina.laundry.domain.Customer;
import com.profitina.laundry.service.CustomerService;
import jakarta.validation.Valid;
import java.util.List;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/customers")
public class CustomerApiController {

    private final CustomerService customers;

    public CustomerApiController(CustomerService customers) {
        this.customers = customers;
    }

    @GetMapping
    public DataView<List<CustomerView>> index(
            @RequestParam(required = false) String phone) {
        List<Customer> found = phone == null ? customers.search("")
                : customers.findByPhone(phone).stream().toList();
        return new DataView<>(found.stream().map(CustomerView::of).toList());
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public DataView<CustomerView> store(
            @Valid @RequestBody CreateCustomerRequest in) {
        return new DataView<>(CustomerView.of(
                customers.create(in.name(), in.phone(), in.address())));
    }
}
```

Kode 13.13 `ServiceApiController.java`: daftar harga dalam JSON.

```java
package com.profitina.laundry.api;

import com.profitina.laundry.service.OrderService;
import java.util.List;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/services")
public class ServiceApiController {

    private final OrderService orders;

    public ServiceApiController(OrderService orders) {
        this.orders = orders;
    }

    @GetMapping
    public DataView<List<ServiceView>> index() {
        return new DataView<>(orders.activeServices().stream()
                .map(ServiceView::of).toList());
    }

    @GetMapping("/{id}")
    public DataView<ServiceView> show(@PathVariable String id) {
        return new DataView<>(ServiceView.of(orders.service(id)));
    }
}
```

Kode 13.14 `ReportApiController.java`: laporan harian dalam JSON.

```
package com.profitina.laundry.api;

import com.profitina.laundry.service.ReportService;
import com.profitina.laundry.service.ReportService.DayFigures;
import com.profitina.laundry.service.RuleException;
import java.time.LocalDate;
import java.time.format.DateTimeParseException;
import java.util.LinkedHashMap;
import java.util.Map;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ReportApiController {

    private final ReportService reports;

    public ReportApiController(ReportService reports) {
        this.reports = reports;
    }

    /** GET /api/reports/daily?date=2026-09-25 */
    @GetMapping("/api/reports/daily")
    public DataView<DailyReport> daily(
            @RequestParam(required = false) String date) {
        DayFigures f = reports.daily(parseDate(date));
        Map<String, Long> byStatus = new LinkedHashMap<>();
        f.byStatus().forEach((status, n) -> byStatus.put(status.value(), n));
        return new DataView<>(new DailyReport(f.day().toString(),
                f.received(), f.billed(), f.collected(), byStatus));
    }

    private static LocalDate parseDate(String date) {
        if (date == null || date.isBlank()) {
            throw new RuleException("date", "The date field is required.");
        }
        try {
            return LocalDate.parse(date);
        } catch (DateTimeParseException e) {
            throw new RuleException("date",
                    "The date field must match the format Y-m-d.");
        }
    }
}
```

Mengurai tanggal tetap berada di controller laporan, karena “teks di *query string* bukan tanggal” adalah pertanyaan tentang masukan HTTP, bukan tentang toko. Terakhir, **Kode 13.15** adalah penangan galat Bab 12, yang kini mendengarkan eksepsi dari service.

Kode 13.15 `ApiExceptionHandler.java`: setiap galat API dalam bentuk kontrak.

```java
package com.profitina.laundry.api;

import com.profitina.laundry.service.ConflictException;
import com.profitina.laundry.service.NotFoundException;
import com.profitina.laundry.service.RuleException;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.http.converter.HttpMessageNotReadableException;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import org.springframework.web.server.ResponseStatusException;

/** Setiap galat dalam bentuk kontrak: {"message": ..., "errors": ...}. */
@RestControllerAdvice(basePackages = "com.profitina.laundry.api")
public class ApiExceptionHandler {

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Map<String, Object>> invalid(
            MethodArgumentNotValidException e) {
        Map<String, List<String>> errors = new LinkedHashMap<>();
        for (FieldError f : e.getBindingResult().getFieldErrors()) {
            errors.computeIfAbsent(snake(f.getField()),
                    k -> new ArrayList<>()).add(f.getDefaultMessage());
        }
        return unprocessable(errors);
    }

    @ExceptionHandler(RuleException.class)
    public ResponseEntity<Map<String, Object>> rule(
            RuleException e) {
        return unprocessable(e.getErrors());
    }
```

Kode 13.15 (Lanjutan) *ApiExceptionHandler.java: setiap galat API dalam bentuk kontrak.*

```
    @ExceptionHandler(ResponseStatusException.class)
    public ResponseEntity<Map<String, Object>> status(
            ResponseStatusException e) {
        String reason = e.getReason() == null ? "" : e.getReason();
        return ResponseEntity.status(e.getStatusCode())
                .body(Map.of("message", reason));
    }

    @ExceptionHandler(NotFoundException.class)
    public ResponseEntity<Map<String, Object>> notFound(NotFoundException e) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND)
                .body(Map.of("message", e.getMessage()));
    }

    @ExceptionHandler(ConflictException.class)
    public ResponseEntity<Map<String, Object>> conflict(ConflictException e) {
        return ResponseEntity.status(HttpStatus.CONFLICT)
                .body(Map.of("message", e.getMessage()));
    }

    @ExceptionHandler(HttpMessageNotReadableException.class)
    public ResponseEntity<Map<String, Object>> unreadable(
            HttpMessageNotReadableException e) {
        return ResponseEntity.badRequest().body(Map.of("message",
                "The request body is not valid JSON for this endpoint."));
    }

    private static ResponseEntity<Map<String, Object>> unprocessable(
            Map<String, List<String>> errors) {
        Map<String, Object> body = new LinkedHashMap<>();
        body.put("message", "The given data was invalid.");
        body.put("errors", errors);
        return ResponseEntity.status(HttpStatus.UNPROCESSABLE_CONTENT)
                .body(body);
    }

    /** customerId -> customer_id: nama yang dikirim klien. */
    static String snake(String name) {
        return name.replaceAll("([a-z0-9])([A-Z])", "$1_$2").toLowerCase();
    }
}
```

`basePackages = "com.profitina.laundry.api"` membatasi penangan ini pada controller JSON. Tanpanya, `RuleException` yang dilempar saat sebuah *halaman* dilayani juga akan dijawab dengan JSON dan status 422, persis hal yang tidak boleh diterima peramban. Pemetaannya mengikuti kontrak: `RuleException` menjadi 422 dengan `errors`, `NotFoundException` 404, dan `ConflictException` 409, masing-masing dengan `message`.

Karena URL, *record* permintaan, *record* tampilan, dan bentuk galatnya adalah milik Bab 12, program kontrak Bab 12 berlaku tanpa perubahan; menjalankannya terhadap aplikasi bab ini adalah cara tercepat membuktikan bahwa *refactoring* ini tidak mengubah apa pun yang dapat dilihat klien.

13.6 Memvalidasi Masukan Formulir

13.6.1 Jakarta Bean Validation

Spring Boot memvalidasi dengan *Jakarta Bean Validation* 3.1, diimplementasikan oleh Hibernate Validator dan dibawa oleh `spring-boot-starter-validation` [4]. Anda memasang anotasi

batasan pada field; *validator* memeriksa objek terhadap batasan itu dan mengembalikan daftar pelanggaran, masing-masing dengan nama field dan pesan. **Tabel 13.3** mencantumkan batasan yang dipakai formulir-formulir bab ini.

Tabel 13.3 Batasan Bean Validation yang dipakai formulir.

Anotasi	Memeriksa bahwa nilainya…	Dipakai pada
`@NotBlank`	tidak null dan memuat karakter selain spasi	nama, telepon, layanan, metode pembayaran
`@NotNull`	ada	pelanggan, jumlah, nominal
`@Size(min, max)`	panjangnya antara `min` dan `max` karakter	nama, alamat, catatan
`@Pattern(regexp = r)`	cocok dengan ekspresi reguler *r*	telepon
`@DecimalMin`, `@DecimalMax`	berada di antara dua batas desimal	jumlah
`@Digits(integer, fraction)`	punya paling banyak sekian angka sebelum dan sesudah koma	jumlah
`@Min`	paling sedikit sebuah bilangan bulat	nominal pembayaran

Dua fakta menghemat banyak kebingungan. Sebagian besar batasan menganggap `null` *sah*, sehingga alamat opsional dengan `@Size(max = 200)` boleh dikosongkan; field wajib menyatakannya dengan `@NotNull` atau `@NotBlank`. Dan batasan memeriksa *bentuk*, bukan *makna*: apakah nomor telepon sudah dipakai, atau apakah pesanan masih punya utang, hanya dapat dijawab dengan basis data, dan itulah sebabnya pemeriksaan semacam itu tetap berada di service.

13.6.2 *Record* formulir

Kode 13.16 adalah apa yang dikirim formulir pelanggan.

Kode 13.16 `CustomerForm.java`: formulir pelanggan, dengan batas yang sama seperti permintaan API.

```
package com.profitina.laundry.web;

import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Pattern;
import jakarta.validation.constraints.Size;

/** Isi kiriman formulir pelanggan; bukan entitasnya. */
public record CustomerForm(
        @NotBlank(message = "Please enter the customer's name.")
        @Size(min = 2, max = 80) String name,
        @NotBlank(message = "Please enter a phone number.")
        @Pattern(regexp = "^08[0-9]{8,11}$",
                 message = "Write the phone as 08..., 10 to 13 digits.")
        String phone,
        @Size(max = 200) String address) {

    /** Sisakan angka saja, seperti API: "0812-3456-7890" -> angka. */
    public CustomerForm {
        phone = phone == null ? null : phone.replaceAll("[^0-9]", "");
    }

    public static CustomerForm empty() {
        return new CustomerForm("", "", "");
    }
}
```

Record Java adalah kelas ringkas yang komponennya ditetapkan saat konstruksi; Spring MVC membuatnya dari field yang dikirim dengan memanggil konstruktornya, dan Thymeleaf membaca *accessor*-nya untuk mengisi kembali input. *Compact constructor* berjalan sebelum komponen disimpan dan hanya menyisakan angka pada telepon, persis seperti `CreateCustomerRequest` di Bab 11, sehingga "0812-3456-7890" diperiksa dan disimpan sebagai `081234567890`.

Mengapa formulir tidak diikat langsung ke entitas `Customer`? Karena entitas punya field yang tidak boleh diisi formulir mana pun, seperti `id` dan `createdAt`. Bila entitas menjadi objek formulir, siapa pun dapat menambahkan `createdAt=2001-01-01` ke permintaan dan Spring akan mengikatnya. Serangan ini disebut *mass assignment*, dan *record* terpisah menutupnya dengan sendirinya: hanya komponennya yang dapat diikat. **Kode 13.17** dan **Kode 13.18** menggambarkan dua formulir lainnya.

Kode 13.17 `OrderForm.java`: formulir pesanan baru, yang tahu cara menjadi `NewOrder`.

```
package com.profitina.laundry.web;

import com.profitina.laundry.service.OrderService.NewOrder;
import jakarta.validation.constraints.DecimalMax;
import jakarta.validation.constraints.DecimalMin;
import jakarta.validation.constraints.Digits;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Size;
import java.math.BigDecimal;

/** Formulir pesanan baru: batas yang sama dengan CreateOrderRequest. */
public record OrderForm(
        @NotNull(message = "Choose a customer.") Long customerId,
        @NotBlank(message = "Choose a service.") String serviceId,
        @NotNull(message = "Enter the weight or the number of items.")
        @DecimalMin("0.1") @DecimalMax("100")
        @Digits(integer = 3, fraction = 2) BigDecimal quantity,
        boolean pickup,
        @Size(max = 200) String deliveryAddress,
        @Size(max = 200) String note) {

    public NewOrder toNewOrder() {
        return new NewOrder(customerId, serviceId, quantity, pickup,
                deliveryAddress, note);
    }
}
```

Kode 13.18 `PaymentForm.java`: formulir pembayaran.

```
package com.profitina.laundry.web;

import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;

public record PaymentForm(
        @NotNull(message = "Enter the amount.")
        @Min(value = 1000, message = "At least Rp 1.000.") Long amount,
        @NotBlank String method) {
}
```

Field-nya bertipe pembungkus (`Long`, `BigDecimal`), bukan primitif, agar field kosong tiba sebagai `null` dan `@NotNull` melaporkannya dengan kata-kata biasa; `long` primitif akan mengubah field kosong menjadi galat konversi dengan pesan teknis.

13.6.3 Pesan aturan di field yang tepat

Service memakai nama field API, `customer_id` dan `delivery_address`; formulir punya `customerId` dan `deliveryAddress`. **Kode 13.19** menerjemahkan yang satu ke yang lain dan menempelkan setiap pesan pada field formulir pemiliknya.

Kode 13.19 `Forms.java`: pesan aturan dari service menjadi galat pada field formulir.

```
package com.profitina.laundry.web;

import java.util.List;
import java.util.Map;
import org.springframework.validation.BindingResult;

/** Menaruh pesan aturan dari service di samping field formulir yang tepat. */
final class Forms {

    private Forms() {
    }

    /** {"customer_id": [...]} -> galat pada field formulir customerId. */
    static void reject(BindingResult result, Map<String, List<String>> errors) {
        errors.forEach((field, messages) -> messages.forEach(
                message -> result.rejectValue(camel(field), "rule", message)));
    }

    /** customer_id -> customerId: aturan memakai nama-nama API. */
    static String camel(String snake) {
        StringBuilder out = new StringBuilder();
        boolean up = false;
        for (char c : snake.toCharArray()) {
            if (c == '_') {
                up = true;
            } else {
                out.append(up ? Character.toUpperCase(c) : c);
                up = false;
            }
        }
        return out.toString();
    }
}
```

`BindingResult` adalah wadah Spring untuk segala yang salah saat mengikat dan memvalidasi satu objek formulir; `rejectValue` menambahkan galat pada sebuah field, dan templat menampilkannya di samping field itu persis seperti pesan Bean Validation.

13.7 Halaman dengan Thymeleaf

13.7.1 Halaman pelanggan

Kode 13.20 adalah controller pelanggan yang lengkap: daftar dan pencarian, baru, buat, ubah, simpan perubahan, dan hapus.

Kode 13.20 `CustomerController.java`: CRUD lengkap untuk pelanggan sebagai halaman.

```java
package com.profitina.laundry.web;

import com.profitina.laundry.domain.Customer;
import com.profitina.laundry.service.ConflictException;
import com.profitina.laundry.service.CustomerService;
import com.profitina.laundry.service.RuleException;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

/** Halaman pelanggan: daftar dan cari, tambah, ubah, hapus. */
@Controller
@RequestMapping("/customers")
public class CustomerController {

    private final CustomerService customers;

    public CustomerController(CustomerService customers) {
        this.customers = customers;
    }

    @GetMapping
    public String list(@RequestParam(required = false) String q, Model model) {
        model.addAttribute("customers", customers.search(q));
        model.addAttribute("q", q);
        return "customers/list";
    }

    @GetMapping("/new")
    public String newForm(Model model) {
        model.addAttribute("form", CustomerForm.empty());
        return "customers/form";
    }
```

Kode 13.20 (Lanjutan) *CustomerController.java: CRUD lengkap untuk pelanggan sebagai halaman.*

```
    @PostMapping
    public String create(@Validated @ModelAttribute("form") CustomerForm form,
                         BindingResult errors, RedirectAttributes flash) {
        if (errors.hasErrors()) {
            return "customers/form";
        }
        try {
            Customer c = customers.create(form.name(), form.phone(),
                    form.address());
            flash.addFlashAttribute("message", c.getName() + " was added.");
            return "redirect:/customers";
        } catch (RuleException e) {
            Forms.reject(errors, e.getErrors());
            return "customers/form";
        }
    }

    @GetMapping("/{id}/edit")
    public String editForm(@PathVariable Long id, Model model) {
        Customer c = customers.get(id);
        model.addAttribute("form",
                new CustomerForm(c.getName(), c.getPhone(), c.getAddress()));
        model.addAttribute("customerId", id);
        return "customers/form";
    }

    @PostMapping("/{id}")
    public String update(@PathVariable Long id,
                         @Validated @ModelAttribute("form") CustomerForm form,
                         BindingResult errors, Model model,
                         RedirectAttributes flash) {
        model.addAttribute("customerId", id);
        if (errors.hasErrors()) {
            return "customers/form";
        }
        try {
            customers.update(id, form.name(), form.phone(), form.address());
            flash.addFlashAttribute("message", "Changes saved.");
            return "redirect:/customers";
        } catch (RuleException e) {
            Forms.reject(errors, e.getErrors());
            return "customers/form";
        }
    }
```

Kode 13.20 (Lanjutan) *CustomerController.java: CRUD lengkap untuk pelanggan sebagai halaman.*

```
    @PostMapping("/{id}/delete")
    public String delete(@PathVariable Long id, RedirectAttributes flash) {
        try {
            customers.delete(id);
            flash.addFlashAttribute("message", "Customer deleted.");
        } catch (ConflictException e) {
            flash.addFlashAttribute("error", e.getMessage());
        }
        return "redirect:/customers";
    }
}
```

Pemetaannya adalah pemetaan lazim untuk halaman yang dirender di server (**Tabel 13.4**).

Tabel 13.4 URL halaman pelanggan.

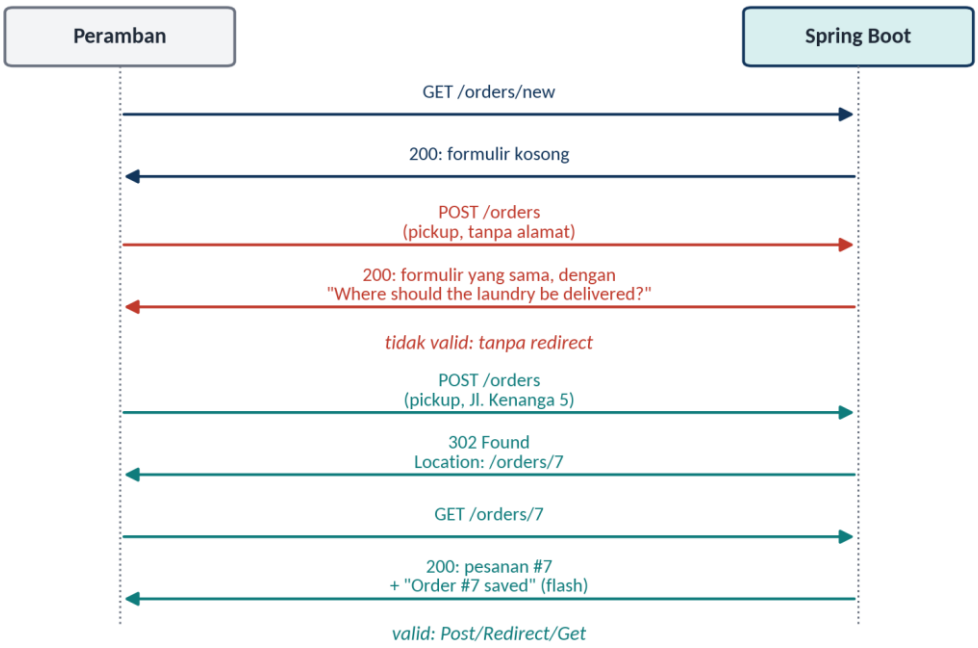
Metode dan URL	Aksi	Jawaban
`GET /customers?q=...`	daftar dan cari	halaman daftar
`GET /customers/new`	tampilkan formulir kosong	halaman formulir
`POST /customers`	buat	*redirect* ke daftar, atau formulir dengan galat
`GET /customers/{id}/edit`	tampilkan formulir terisi	halaman formulir
`POST /customers/{id}`	simpan perubahan	*redirect* ke daftar, atau formulir dengan galat
`POST /customers/{id}/delete`	hapus	*redirect* ke daftar dengan pesan

Formulir HTML hanya dapat mengirim `GET` dan `POST`, sehingga mengubah dan menghapus memakai `POST`. Menghapus tidak pernah memakai `GET`: mesin pencari dan peramban yang memuat lebih dulu mengikuti tautan, dan tautan yang menghapus akan benar-benar menghapus.

Perhatikan `create`. Parameter `@Validated @ModelAttribute("form") CustomerForm form` meminta Spring MVC membangun *record* dari permintaan, memvalidasinya, dan menaruhnya di model dengan nama `form`. Parameter tepat sesudahnya, `BindingResult errors`, menerima galat validasi. Parameter ini *harus* datang tepat setelah parameter yang divalidasi; bila tidak ada, Spring melempar eksepsi alih-alih membiarkan metode menangani galatnya. Ketika ada galat, metode mengembalikan templat formulir lagi, dengan ketikan pengguna dan pesan-pesannya masih di model. Ketika service menolak (“The phone has already been taken.”), `Forms.reject` menambahkan penolakan itu ke `BindingResult` yang sama, dan halaman menampilkannya di bawah kotak telepon.

13.7.2 Post/Redirect/Get

Ketika pelanggan berhasil dibuat, metode tidak merender daftar. Ia mengembalikan `"redirect:/customers"`, yang membuat peramban memuat daftar dengan `GET` baru. **Gambar 13.2** memperlihatkan alasannya, dengan formulir pesanan sebagai contoh.



Gambar 13.2 Post/Redirect/Get: formulir yang tidak sah dijawab dengan halaman yang sama beserta pesannya; yang sah dijawab dengan *redirect*, sehingga menyegarkan halaman tidak pernah menerima pesanan yang sama dua kali.

Bila halaman dirender sebagai jawaban atas `POST`, menekan F5 (atau mundur lalu maju) membuat peramban mengirim `POST` itu lagi, dan toko punya dua pesanan kembar. Setelah *redirect*, halaman di layar adalah jawaban atas `GET` yang tidak berbahaya dan boleh diulang berapa kali pun. Pola ini disebut *Post/Redirect/Get* (PRG) dan bersandar pada kode status *redirect* HTTP [5]. Pesan seperti "Order #7 saved: Rp 34.000." tidak dapat melintasi *redirect* di dalam model, karena model hanya hidup selama satu permintaan; `RedirectAttributes.addFlashAttribute` menyimpannya di sesi untuk tepat satu permintaan berikutnya, itulah sebabnya disebut atribut *flash*. Sebaliknya, formulir yang *tidak sah* langsung dijawab dengan formulir: tidak ada yang disimpan, jadi tidak ada yang dapat terulang, dan ketikan kasir tidak hilang.

13.7.3 Templat

Templat Thymeleaf adalah berkas HTML dengan atribut tambahan dalam *namespace* `th:`, disimpan di `src/main/resources/templates` [6]. Templat terbuka di peramban sebagai HTML biasa dengan teks contohnya, sehingga perancang dapat mengerjakannya tanpa menjalankan aplikasi. **Kode 13.21** memuat bagian yang dipakai bersama semua halaman.

Kode 13.21 `fragments.html`: kepala halaman, navigasi, dan pesan *flash*.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head th:fragment="head(title)">
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title th:text="${title} + ' | Profitina Laundry'">Profitina Laundry</title>
  <link rel="stylesheet" th:href="@{/css/app.css}">
</head>
<body>
<header th:fragment="nav">
  <strong>Profitina Laundry</strong>
  <nav>
    <a th:href="@{/orders(status='queued')}">Queue</a>
    <a th:href="@{/orders}">All orders</a>
    <a th:href="@{/orders/new}">New order</a>
    <a th:href="@{/customers}">Customers</a>
  </nav>
</header>
<div th:fragment="flash">
  <p class="ok" th:if="${message}" th:text="${message}">Saved.</p>
  <p class="bad" th:if="${error}" th:text="${error}">Not allowed.</p>
</div>
</body>
</html>
```

`th:fragment="head(title)"` mendeklarasikan potongan yang dapat dipakai ulang dengan sebuah parameter; halaman menyertakannya dengan `th:replace="~{fragments :: head('Customers')}"`. `@{/orders}` membangun URL relatif terhadap *context path* aplikasi, sehingga halaman tetap berjalan bila aplikasi dipasang di bawah `/laundry/`. `${message}` membaca atribut *flash*, dan `th:if` membuang elemennya bila atribut itu tidak ada. **Kode 13.22** adalah daftar pelanggan.

Kode 13.22 `customers/list.html`: daftar pelanggan yang dapat dicari.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head th:replace="~{fragments :: head('Customers')}"></head>
<body>
<header th:replace="~{fragments :: nav}"></header>
<main>
  <h1>Customers</h1>
  <div th:replace="~{fragments :: flash}"></div>
  <form method="get" th:action="@{/customers}" class="inline">
    <input name="q" th:value="${q}" placeholder="Name or phone">
    <button>Search</button>
    <a class="button" th:href="@{/customers/new}">New customer</a>
  </form>
  <table>
    <thead><tr><th>Name</th><th>Phone</th><th>Address</th><th></th></tr></thead>
    <tbody>
    <tr th:each="c : ${customers}">
      <td th:text="${c.name}">Sari Wulandari</td>
      <td th:text="${c.phone}">081200000101</td>
      <td th:text="${c.address} ?: '-'">Jl. Kertajaya 12</td>
      <td class="actions">
        <a th:href="@{/orders/new(customerId=${c.id})}">New order</a>
        <a th:href="@{/customers/{id}/edit(id=${c.id})}">Edit</a>
        <form method="post" th:action="@{/customers/{id}/delete(id=${c.id})}"
              onsubmit="return confirm('Delete this customer?')">
          <button class="link">Delete</button>
        </form>
      </td>
    </tr>
    <tr th:if="${customers.isEmpty()}"><td colspan="4">No customers found.</td></tr>
    </tbody>
  </table>
</main>
</body>
</html>
```

`th:each` mengulang baris sekali per pelanggan. `th:text` menaruh teks ke dalam elemen dan meng-*escape*-nya: pelanggan bernama `<script>alert(1)</script>` ditampilkan sebagai karakter-karakter itu, tidak pernah dijalankan. Perlindungan terhadap *cross-site scripting* (XSS) ini adalah bawaan Thymeleaf; `th:utext`, yang tidak meng-*escape*, hanya untuk HTML yang Anda tulis sendiri. Operator *elvis* `${c.address} ?: '-'` mencetak tanda hubung untuk pelanggan tanpa alamat. **Kode 13.23** melayani “baru” maupun “ubah”.

Kode 13.23 `customers/form.html`: satu formulir untuk menambah dan mengubah, dengan pesan di samping setiap field.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head th:replace="~{fragments :: head('Customer')}"></head>
<body>
<header th:replace="~{fragments :: nav}"></header>
<main>
  <h1 th:text="${customerId} ? 'Edit customer' : 'New customer'">Customer</h1>
  <form method="post" th:object="${form}"
        th:action="${customerId} ? @{/customers/{id}(id=${customerId})}
                                 : @{/customers}">
    <p class="bad" th:each="err : ${#fields.globalErrors()}" th:text="${err}"></p>
    <label>Name
      <input th:field="*{name}" th:errorclass="invalid" autofocus>
    </label>
    <p class="bad" th:errors="*{name}"></p>
    <label>Phone
      <input th:field="*{phone}" th:errorclass="invalid" placeholder="08...">
    </label>
    <p class="bad" th:errors="*{phone}"></p>
    <label>Address (for pickup and delivery)
      <input th:field="*{address}" th:errorclass="invalid">
    </label>
    <p class="bad" th:errors="*{address}"></p>
    <button>Save</button>
    <a th:href="@{/customers}">Cancel</a>
  </form>
</main>
</body>
</html>
```

`th:object="${form}"` menjadikan *record* itu objek formulir, dan `th:field="*{name}"` mengerjakan tiga hal sekaligus: menetapkan `name` dan `id` input menjadi `name`, mengisi `value`-nya dari *record*, dan, bersama `th:errorclass="invalid"`, menambahkan kelas CSS `invalid` bila field itu bergalat. `th:errors="*{phone}"` mencetak pesan untuk field telepon, entah berasal dari `@Pattern` atau dari service. Tujuan formulir bergantung pada ada tidaknya `customerId` di model: perubahan dikirim ke `/customers/5`, pelanggan baru ke `/customers`.

13.7.4 Pesanan dan pembayaran

Kode 13.24 menjalankan konter.

Kode 13.24 `OrderController.java`: antrean, pesanan baru, satu pesanan, langkah status, dan pembayaran.

```
package com.profitina.laundry.web;

import com.profitina.laundry.domain.LaundryRules;
import com.profitina.laundry.domain.Order;
import com.profitina.laundry.domain.OrderStatus;
import com.profitina.laundry.domain.PaymentMethod;
import com.profitina.laundry.service.ConflictException;
import com.profitina.laundry.service.CustomerService;
import com.profitina.laundry.service.OrderService;
import com.profitina.laundry.service.RuleException;
import java.util.List;
import java.util.Map;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

/** Halaman konter: antrean, pesanan baru, satu pesanan dan uangnya. */
@Controller
@RequestMapping("/orders")
public class OrderController {

    private final OrderService orders;
    private final CustomerService customers;

    public OrderController(OrderService orders, CustomerService customers) {
        this.orders = orders;
        this.customers = customers;
    }
```

Kode 13.24 (Lanjutan) *OrderController.java: antrean, pesanan baru, satu pesanan, langkah status, dan pembayaran.*

```
    @GetMapping
    public String list(@RequestParam(required = false) String status,
                       @RequestParam(defaultValue = "1") int page,
                       Model model) {
        OrderStatus wanted = parse(status);
        model.addAttribute("orders", orders.page(wanted, page, 30));
        model.addAttribute("status", wanted);
        model.addAttribute("statuses", OrderStatus.values());
        return "orders/list";
    }

    @GetMapping("/new")
    public String newForm(@RequestParam(required = false) Long customerId,
                          Model model) {
        model.addAttribute("form", new OrderForm(customerId, "", null,
                false, "", ""));
        return fillChoices(model);
    }

    @PostMapping
    public String create(@Validated @ModelAttribute("form") OrderForm form,
                         BindingResult errors, Model model,
                         RedirectAttributes flash) {
        Map<String, List<String>> rules = orders.check(form.toNewOrder());
        Forms.reject(errors, rules);            // semua masalah sekaligus
        if (errors.hasErrors()) {
            return fillChoices(model);
        }
        Order order = orders.create(form.toNewOrder());
        flash.addFlashAttribute("message", "Order #" + order.getId()
                + " saved: " + LaundryRules.rupiah(order.getTotal()) + ".");
        return "redirect:/orders/" + order.getId();
    }

    @GetMapping("/{id}")
    public String detail(@PathVariable Long id, Model model) {
        if (!model.containsAttribute("payment")) {
            model.addAttribute("payment", new PaymentForm(null, "cash"));
        }
        model.addAttribute("order", orders.get(id));
        model.addAttribute("methods", PaymentMethod.values());
        return "orders/detail";
    }
```

Kode 13.24 (Lanjutan) *OrderController.java: antrean, pesanan baru, satu pesanan, langkah status, dan pembayaran.*

```
    /** Satu langkah alur kerja; service yang menentukan langkahnya. */
    @PostMapping("/{id}/advance")
    public String advance(@PathVariable Long id, RedirectAttributes flash) {
        try {
            Order order = orders.advance(id, null);
            flash.addFlashAttribute("message", "Order #" + id + " is now "
                    + order.getStatus().value() + ".");
        } catch (RuleException e) {
            flash.addFlashAttribute("error",
                    e.getErrors().values().iterator().next().get(0));
        } catch (ConflictException e) {
            flash.addFlashAttribute("error", e.getMessage());
        }
        return "redirect:/orders/" + id;
    }

    @PostMapping("/{id}/payments")
    public String pay(@PathVariable Long id,
                      @Validated @ModelAttribute("payment") PaymentForm payment,
                      BindingResult errors, Model model,
                      RedirectAttributes flash) {
        if (!errors.hasErrors()) {
            try {
                orders.pay(id, payment.amount(),
                        PaymentMethod.fromValue(payment.method()));
                flash.addFlashAttribute("message", "Payment recorded.");
                return "redirect:/orders/" + id;
            } catch (RuleException e) {
                Forms.reject(errors, e.getErrors());
            } catch (ConflictException e) {
                flash.addFlashAttribute("error", e.getMessage());
                return "redirect:/orders/" + id;
            }
        }
        return detail(id, model);
    }

    /** "queued" -> QUEUED; teks kosong/tak dikenal berarti "semua pesanan". */
    private static OrderStatus parse(String status) {
        try {
            return status == null ? null : OrderStatus.fromValue(status);
        } catch (IllegalArgumentException e) {
            return null;
        }
    }
```

Kode 13.24 (Lanjutan) *OrderController.java: antrean, pesanan baru, satu pesanan, langkah status, dan pembayaran.*

```
    private String fillChoices(Model model) {
        model.addAttribute("customers", customers.search(""));
        model.addAttribute("services", orders.activeServices());
        return "orders/form";
    }
}
```

Perhatikan betapa sedikit yang dikerjakan setiap metode. `create` meminta service memeriksa (`check`) pesanan baru dan menambahkan setiap masalah ke formulir, sehingga kasir melihat semua pesan sekaligus, seperti klien API; hanya formulir yang bersih yang dibuat. `advance` memanggil service dengan `null` (“langkah berikutnya”), menaruh hasilnya dalam pesan *flash*, lalu melakukan *redirect*; ia tidak tahu bahwa pembayaran ada, service yang tahu. Ketika service

berkata tidak (“Collect Rp 34.000 before handing over the laundry.”), penolakan itu menjadi baris merah di halaman pesanan. `list` membaca filter status dari *query string* dan diam-diam menampilkan semua pesanan untuk kata yang tidak dikenal, yang lebih ramah untuk halaman daripada galat. `HomeController` (**Kode 13.25**) membuka toko pada antrean.

Kode 13.25 `HomeController.java`: halaman depan adalah antrean.

```
package com.profitina.laundry.web;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HomeController {

    /** Toko dibuka pada halaman antrean. */
    @GetMapping("/")
    public String home() {
        return "redirect:/orders?status=queued";
    }
}
```

Ketiga templat pesanan mengikuti pola templat pelanggan: **Kode 13.26** adalah antrean, **Kode 13.27** formulir pesanan baru, **Kode 13.28** satu pesanan dengan pembayarannya.

Kode 13.26 `orders/list.html`: satu halaman pesanan, disaring menurut status.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head th:replace="~{fragments :: head('Orders')}"></head>
<body>
<header th:replace="~{fragments :: nav}"></header>
<main th:with="R=${T(com.profitina.laundry.domain.LaundryRules)}">
  <h1 th:text="${status} ? 'Orders: ' + ${status.value()} : 'All orders'">
    Orders</h1>
  <div th:replace="~{fragments :: flash}"></div>
  <p>
    <a th:each="s : ${statuses}" th:href="@{/orders(status=${s.value()})}"
       th:text="${s.value()}" th:classappend="${s == status} ? 'current'">
      queued</a>
    <a th:href="@{/orders}">all</a>
  </p>
  <table>
    <thead><tr><th>#</th><th>Received</th><th>Customer</th><th>Service</th>
      <th class="num">Qty</th><th class="num">Total</th><th>Status</th></tr>
    </thead>
    <tbody>
    <tr th:each="o : ${orders.content}">
      <td><a th:href="@{/orders/{id}(id=${o.id})}" th:text="${o.id}">1</a></td>
      <td th:text="${#temporals.format(o.receivedAt, 'dd MMM HH:mm')}">01 Sep</td>
      <td th:text="${o.customer.name}">Sari</td>
      <td th:text="${o.service.name()}">Wash, dry &amp;amp; fold</td>
      <td class="num"
          th:text="${o.quantity} + ' ' + ${o.service.unit()}">4.50 kg</td>
      <td class="num" th:text="${R.rupiah(o.total)}">Rp 36.000</td>
      <td th:text="${o.status.value()}">queued</td>
    </tr>
    <tr th:if="${orders.isEmpty()}"><td colspan="7">No orders here.</td></tr>
    </tbody>
  </table>
  <p th:if="${orders.totalPages > 1}"
     th:text="'Page ' + ${orders.number + 1} + ' of ' + ${orders.totalPages}">
    Page 1 of 2</p>
</main>
</body>
</html>
```

Kode 13.27 `orders/form.html`: menerima sekantong cucian.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head th:replace="~{fragments :: head('New order')}"></head>
<body>
<header th:replace="~{fragments :: nav}"></header>
<main th:with="R=${T(com.profitina.laundry.domain.LaundryRules)}">
  <h1>New order</h1>
  <form method="post" th:action="@{/orders}" th:object="${form}">
    <label>Customer
      <select th:field="*{customerId}" th:errorclass="invalid">
        <option value="">-- choose --</option>
        <option th:each="c : ${customers}" th:value="${c.id}"
                th:text="${c.name} + ' (' + ${c.phone} + ')'">Sari</option>
      </select>
    </label>
    <p class="bad" th:errors="*{customerId}"></p>
    <label>Service
      <select th:field="*{serviceId}" th:errorclass="invalid">
        <option value="">-- choose --</option>
        <option th:each="s : ${services}" th:value="${s.id}"
                th:text="${s.description} + ', ' + ${R.rupiah(s.price)}">
          Wash, dry &amp;amp; fold (per kg), Rp 8.000</option>
      </select>
    </label>
    <p class="bad" th:errors="*{serviceId}"></p>
    <label>Quantity (kg, items or pairs)
      <input th:field="*{quantity}" th:errorclass="invalid" inputmode="decimal">
    </label>
    <p class="bad" th:errors="*{quantity}"></p>
    <label class="check"><input type="checkbox" th:field="*{pickup}">
      Pickup and delivery (Rp 10.000 below Rp 100.000)</label>
    <label>Delivery address
      <input th:field="*{deliveryAddress}" th:errorclass="invalid">
    </label>
    <p class="bad" th:errors="*{deliveryAddress}"></p>
    <label>Note
      <input th:field="*{note}" placeholder="e.g. separate the white shirts">
    </label>
    <p class="bad" th:errors="*{note}"></p>
    <button>Save order</button>
  </form>
</main>
</body>
</html>
```

Kode 13.28 `orders/detail.html`: pesanan dengan total, pembayaran, dan sisa tagihannya, serta tombol untuk langkah berikutnya.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head th:replace="~{fragments :: head('Order')}"></head>
<body>
<header th:replace="~{fragments :: nav}"></header>
<main th:with="R=${T(com.profitina.laundry.domain.LaundryRules)}">
  <h1 th:text="'Order #' + ${order.id}">Order #1</h1>
  <div th:replace="~{fragments :: flash}"></div>
  <dl>
    <dt>Customer</dt><dd th:text="${order.customer.name}">Sari</dd>
    <dt>Service</dt>
    <dd th:text="${order.service.name()} + ', ' + ${order.quantity} + ' '
                + ${order.service.unit()}">Wash, dry &amp;amp; fold, 4.50 kg</dd>
    <dt>Pickup</dt>
    <dd th:text="${order.pickup} ? ${order.deliveryAddress} : 'no'">no</dd>
    <dt>Status</dt><dd th:text="${order.status.value()}">queued</dd>
    <dt>Total</dt><dd th:text="${R.rupiah(order.total)}">Rp 36.000</dd>
    <dt>Paid</dt><dd th:text="${R.rupiah(order.paid())}">Rp 0</dd>
    <dt>Balance</dt><dd th:text="${R.rupiah(order.balance())}">Rp 36.000</dd>
  </dl>
  <form method="post" th:action="@{/orders/{id}/advance(id=${order.id})}"
        th:unless="${order.status.value() == 'departed'}">
    <button th:text="${order.status.value() == 'queued'}
                     ? 'Mark as finished' : 'Hand over to customer'">Next</button>
  </form>
  <h2>Payments</h2>
  <table>
    <tr th:each="p : ${order.payments}">
      <td th:text="${#temporals.format(p.paidAt, 'dd MMM HH:mm')}">01 Sep</td>
      <td th:text="${p.method.value()}">cash</td>
      <td class="num" th:text="${R.rupiah(p.amount)}">Rp 10.000</td>
    </tr>
    <tr th:if="${order.payments.isEmpty()}"><td>No payments yet.</td></tr>
  </table>
  <form method="post" th:if="${order.balance() > 0}" th:object="${payment}"
        th:action="@{/orders/{id}/payments(id=${order.id})}" class="inline">
    <input th:field="*{amount}" th:errorclass="invalid" placeholder="Amount">
    <select th:field="*{method}">
      <option th:each="m : ${methods}" th:value="${m.value()}"
              th:text="${m.value()}">cash</option>
    </select>
    <button>Record payment</button>
    <p class="bad" th:errors="*{amount}"></p>
  </form>
</main>
</body>
```

Kode 13.28 (Lanjutan) *orders/detail.html: pesanan dengan total, pembayaran, dan sisa tagihannya, serta tombol untuk langkah berikutnya.*

```
</html>
```

Beberapa fitur Thymeleaf muncul di sini untuk pertama kali. `orders` di daftar adalah `Page` Spring Data, sehingga templat membaca `orders.content` untuk baris-barisnya dan `orders.totalPages` untuk baris penomoran halaman. `#temporals.format(o.receivedAt, 'dd MMM HH:mm')` memformat `LocalDateTime`. `${o.service.name()}` memanggil sebuah metode: `Service` punya `name()` dan `unit()` alih-alih *getter*, karena Bab 11 menurunkannya dari deskripsi. `T(com.profitina.laundry.domain.LaundryRules)` merujuk kelas Java agar templat dapat memanggil metode statis `rupiah`; halaman menyimpannya sekali dalam variabel lokal `R` dengan `th:with`. Memanggil fungsi pemformat kecil yang murni dari templat tidak masalah; memanggil service atau repository dari templat tidak boleh, karena itu memindahkan pekerjaan keluar dari lapisan tempatnya.

Formulir pembayaran hanya muncul selama sisa tagihan positif, dan tombol langkah berikutnya menghilang begitu pesanan sudah departed. Menyembunyikan tombol adalah kesopanan, bukan perlindungan: permintaan tetap dapat dikirim dengan tangan, itulah sebabnya service memeriksa syarat yang sama sekali lagi.

13.8 Satu Tempat untuk Galat

Apa yang terjadi ketika seseorang membuka `/orders/999` dan tidak ada pesanan 999? `OrderService.get` melempar `NotFoundException`. Tanpa petunjuk lebih lanjut Spring akan menjawab 500, "Internal Server Error", yang keliru (tidak ada yang rusak; permintaan meminta sesuatu yang tidak ada) dan tidak ramah. **Kode 13.29** membereskannya untuk semua halaman sekaligus.

Kode 13.29 `WebErrorAdvice.java`: setiap `NotFoundException` di halaman menjadi halaman 404.

```
package com.profitina.laundry.web;

import com.profitina.laundry.service.NotFoundException;
import org.springframework.http.HttpStatus;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;

/** Satu tempat yang mengubah "not found" menjadi halaman 404 ramah. */
@ControllerAdvice(basePackages = "com.profitina.laundry.web")
public class WebErrorAdvice {

    @ExceptionHandler(NotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public String notFound(NotFoundException e, Model model) {
        model.addAttribute("message", e.getMessage());
        return "error/404";
    }
}
```

Kelas beranotasi `@ControllerAdvice` menyumbang ke banyak controller sekaligus, di sini controller paket `web`; metode `@ExceptionHandler`-nya berjalan ketika sebuah metode controller melempar eksepsi yang disebut [7]. `@ResponseStatus(HttpStatus.NOT_FOUND)` menetapkan statusnya, dan metodenya menyebut templat **Kode 13.30**.

Kode 13.30 `error/404.html`: halaman untuk record yang tidak ada.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head th:replace="~{fragments :: head('Not found')}"></head>
<body>
<header th:replace="~{fragments :: nav}"></header>
<main>
  <h1>Not found</h1>
  <p th:text="${message}">That record does not exist.</p>
  <p><a th:href="@{/orders}">Back to the orders</a></p>
</main>
</body>
</html>
```

Halaman dan API kini masing-masing punya satu *advice* galat, terbatas pada paketnya sendiri: `NotFoundException` yang sama menjadi JSON berstatus 404 bagi aplikasi ponsel dan halaman ramah berstatus 404 bagi kasir.

13.9 SQLite, Konfigurasi, dan Menjalankannya

Bab 11 menyimpan data di H2, basis data di memori yang melupakan segalanya ketika aplikasi berhenti, dan menjanjikan SQLite untuk bab ini. **Kode 13.31** adalah berkas *build*-nya.

Kode 13.31 `pom.xml`: *build* Bab 12 ditambah Thymeleaf dan SQLite; H2 tetap untuk tes.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>4.1.1</version>
    <relativePath/>
  </parent>
  <groupId>com.profitina</groupId>
  <artifactId>laundry</artifactId>
  <version>13.0.0</version>
  <properties>
    <java.version>25</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-webmvc</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <!-- SQLite untuk toko, seperti dijanjikan Bab 11 -->
    <dependency>
      <groupId>org.xerial</groupId>
      <artifactId>sqlite-jdbc</artifactId>
      <version>3.53.2.0</version>
    </dependency>
    <dependency>
      <groupId>org.hibernate.orm</groupId>
      <artifactId>hibernate-community-dialects</artifactId>
      <version>${hibernate.version}</version>
    </dependency>
```

Kode 13.31 (Lanjutan) *pom.xml: build Bab 12 ditambah Thymeleaf dan SQLite; H2 tetap untuk tes.*

```
    <!-- H2 di memori, hanya untuk tes -->
    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-webmvc-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

Induk `spring-boot-starter-parent` 4.1.1 menetapkan versi pustaka Spring serta Hibernate, Jackson, dan Thymeleaf, sehingga sebagian besar dependensi tidak memerlukan versi. Spring Boot 4.1 dirilis pada 30 Juni 2026, dan dukungan *open source*-nya berlaku sampai 31 Juli 2027, menurut kalender yang dihimpun endoflife.date [8]; Subbab 15.7 membandingkan kalender itu dengan milik .NET dan Laravel. *Driver* SQLite 3.53.2.0 berasal dari proyek xerial [9], dan `SQLiteDialect` dari *community dialects* Hibernate mengajari Hibernate ragam SQL milik SQLite [10]; keduanya tidak ada dalam daftar yang dikelola Spring Boot, sehingga keduanya menyebut versi. H2 pindah ke *scope* `test`. **Kode 13.32** adalah konfigurasi aplikasi, dan **Kode 13.33** konfigurasi yang dipakai tes sebagai gantinya.

Kode 13.32 `application.properties`: SQLite dalam berkas, skema dijaga mutakhir, JSON snake_case.

```
spring.application.name=profitina-laundry
server.port=8080

# SQLite: sebuah file di samping proyek, tetap ada antar-start. DemoData
# mengisinya pada start pertama saja.
spring.datasource.url=jdbc:sqlite:laundry.db?foreign_keys=true
spring.datasource.driver-class-name=org.sqlite.JDBC
spring.datasource.hikari.maximum-pool-size=1
spring.jpa.database-platform=org.hibernate.community.dialect.SQLiteDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.open-in-view=false

# Nama JSON dalam snake_case (customer_id), sama dengan API Laravel.
spring.jackson.property-naming-strategy=SNAKE_CASE
```

Kode 13.33 `src/test/resources/application.properties`: basis data H2 baru untuk setiap kali tes dijalankan.

```
# Tes memakai basis data H2 baru di memori, seperti di Bab 12.
spring.datasource.url=jdbc:h2:mem:laundry-test
spring.datasource.driver-class-name=org.h2.Driver
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.open-in-view=false
spring.jackson.property-naming-strategy=SNAKE_CASE
```

Setiap pengaturan layak mendapat satu baris penjelasan. `foreign_keys=true` menyalakan pemeriksaan *foreign key* SQLite, yang secara bawaan mati. `maximum-pool-size=1` memberi aplikasi satu koneksi saja: SQLite hanya mengizinkan satu proses menulis pada satu waktu, dan satu koneksi menghindarkan galat "database is locked" di toko kecil. `ddl-auto=update` membiarkan Hibernate membuat tabel pada nyala pertama dan menambahkan kolom yang kurang kemudian; ia tidak pernah menghapus atau mengganti nama apa pun, sehingga proyek nyata beralih ke migrasi (Flyway atau Liquibase di Spring, migrasi EF Core di Bab 5, migrasi Laravel di Bab 9). `open-in-view=false`, dipertahankan dari Bab 11, menutup sesi basis data ketika pemanggilan service selesai, sehingga data setiap halaman harus dimuat di dalam service, persis tempat *entity graph* repository memuatnya. `DemoData` dari Bab 11 masih mengisi daftar harga dan tiga pelanggan, tetapi hanya ketika tabelnya kosong, sehingga menyalakan ulang aplikasi mempertahankan data toko.

Untuk menjalankan aplikasi di Windows, macOS, atau Linux, pasang JDK 25 dan Maven seperti dijelaskan Bab 1, buka `Chapter13/spring` di terminal (atau di VS Code dengan paket ekstensi Java dan Spring Boot), lalu ketik

```
mvn spring-boot:run
```

Setelah Maven mengunduh pustaka-pustakanya (hanya pertama kali) dan mengompilasi proyek, Spring Boot menyalakan Tomcat di port 8080. Buka `http://localhost:8080`; Anda tiba di antrean, yang sudah memuat pesanan demo dari Bab 11. Jalan-jalan pertama yang baik di toko:

1. Buka **Customers**, cari "sari", lalu tambahkan pelanggan baru dengan telepon "0813-1111-2222". Daftarnya menampilkan telepon itu sebagai `081311112222`.
2. Coba tambahkan pelanggan kedua dengan "0813 1111 2222". Formulir kembali dengan "The phone has already been taken." di bawah kotak telepon.
3. Buka **New order**, pilih pelanggan baru itu dan "Wash, dry & fold", ketik `2.5`, centang antar-jemput, biarkan alamat kosong, lalu simpan. Formulir kembali dengan "Where should the laundry be delivered?". Ketik "Jl. Kenanga 5" dan simpan: halaman pesanan menyebut Rp 34.000, minimum dan biaya antar-jemput yang bekerja (Persamaan (13.1)).
4. Tekan **Mark as finished**, lalu **Hand over to customer**. Halaman berkata "Collect Rp 34.000 before handing over the laundry."
5. Catat Rp 40.000 tunai: "Only Rp 34.000 is still owed." Catat Rp 20.000 tunai dan Rp 14.000 lewat QRIS, lalu serahkan pesanannya. Sisa tagihan Rp 0 (Persamaan (13.2)) dan statusnya `departed`.

Pemeriksaan aturan dari Bab 11 masih berjalan dengan JDK saja, dari folder `spring`:

```
javac -d out -sourcepath src/main/java check/RulesCheck.java
java -cp out RulesCheck
```

Program itu mencetak ini di mesin tempat buku ini ditulis:

```
Prices (Problem 8.1)
  Wash, dry & fold             4.50 kg            36000
  Wash, dry & fold             2.00 kg            24000
  Wash, dry & fold             4.02 kg            32160
  Ironing only                 3.35 kg            20100
  Curtain                      4.40 kg   pickup   76000
  Bed cover, any size          2.00 item pickup   80000
  Dry cleaning - suit 2 pieces 2.00 item pickup  130000
  Dry cleaning - shirt/blouse  5.00 item pickup  100000
  Shoes deep clean             1.50 pair          90000
Workflow (Problem 8.2)
  Queued   owing  25000 -> Finished
  Finished owing  25000 -> Collect Rp 25.000 before handing over the laundry.
  Finished owing      0 -> Departed
  Departed owing      0 -> This order has already left the shop.
```

Mesin itu tidak dapat mengunduh pustaka Spring, sehingga aplikasinya sendiri tidak dinyalakan di sana. Setiap berkas Java proyek ini dikompilasi terhadap pengganti-pengganti kecil untuk tipe Spring, Jakarta, dan JUnit yang dipakainya, tanpa galat, dan aturan di atas dikompilasi serta dijalankan sungguhan. README bab ini menyatakan persis hal itu dan meminta Anda menjalankan `mvn test` lebih dulu, sebagai pemeriksaan yang sesungguhnya.

13.10 Menguji Lapisan Service

Aturan sudah diuji sejak Bab 11 (`LaundryRulesTest`), dan API sejak Bab 12 (`OrderApiTest`, kini di paket `api` dan selebihnya tidak berubah). Lapisan service masih baru, dan ia dapat diuji tanpa Spring, tanpa server web, dan tanpa basis data. **Kode 13.34** melakukannya dengan *mock*: pengganti repository yang menjawab apa yang diperintahkan tes.

Kode 13.34 `OrderServiceTest.java`: lapisan service dengan repository tiruan dan jam yang tetap.

```
package com.profitina.laundry.service;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

import com.profitina.laundry.domain.Customer;
import com.profitina.laundry.domain.Order;
import com.profitina.laundry.domain.OrderStatus;
import com.profitina.laundry.domain.PaymentMethod;
import com.profitina.laundry.domain.Service;
import com.profitina.laundry.repository.CustomerRepository;
import com.profitina.laundry.repository.OrderRepository;
import com.profitina.laundry.repository.PaymentRepository;
import com.profitina.laundry.repository.ServiceRepository;
import java.math.BigDecimal;
import java.time.Clock;
import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.util.Optional;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

/** Lapisan service saja: tanpa Spring, tanpa basis data, jam yang tetap. */
class OrderServiceTest {

    private final OrderRepository orders = mock(OrderRepository.class);
    private final CustomerRepository customers =
            mock(CustomerRepository.class);
    private final ServiceRepository services = mock(ServiceRepository.class);
    private final Clock clock = Clock.fixed(
            Instant.parse("2026-09-25T02:00:00Z"), ZoneId.of("Asia/Jakarta"));
    private OrderService service;
    private Order order;
```

Kode 13.34 (Lanjutan) *OrderServiceTest.java: lapisan service dengan repository tiruan dan jam yang tetap.*

```
    @BeforeEach
    void setUp() {
        service = new OrderService(orders, customers, services,
                mock(PaymentRepository.class), clock);
        Customer sari = new Customer("Sari Wulandari", "081200000101", null,
                LocalDateTime.now(clock));
        Service wash = new Service("S0001", "Wash, dry & fold (per kg)",
                8000);
        order = new Order(sari, wash, new BigDecimal("4.5"), false, null,
                null, LocalDateTime.now(clock));
        when(orders.findWithDetailsById(7L)).thenReturn(Optional.of(order));
    }

    @Test
    void finishingStampsTheShopTime() {
        service.advance(7L, null);
        assertEquals(OrderStatus.FINISHED, order.getStatus());
        assertEquals(LocalDateTime.of(2026, 9, 25, 9, 0),
                order.getFinishedAt());
    }

    @Test
    void unpaidLaundryCannotLeave() {
        service.advance(7L, null);                      // queued -> finished
        RuleException e = assertThrows(RuleException.class,
                () -> service.advance(7L, OrderStatus.DEPARTED));
        assertEquals("Collect Rp 36.000 before handing over the laundry.",
                e.getErrors().get("status").get(0));
        assertEquals(OrderStatus.FINISHED, order.getStatus());
    }

    @Test
    void aStepCannotBeSkipped() {
        RuleException e = assertThrows(RuleException.class,
                () -> service.advance(7L, OrderStatus.DEPARTED));
        assertEquals("An order that is queued can only become finished.",
                e.getErrors().get("status").get(0));
    }
```

Kode 13.34 (Lanjutan) *OrderServiceTest.java: lapisan service dengan repository tiruan dan jam yang tetap.*

```
    @Test
    void overpaymentIsRefused() {
        RuleException e = assertThrows(RuleException.class,
                () -> service.pay(7L, 40_000, PaymentMethod.CASH));
        assertEquals("Only Rp 36.000 is still owed.",
                e.getErrors().get("amount").get(0));
    }

    @Test
    void paidLaundryLeavesAndIsThenClosed() {
        service.advance(7L, null);
        service.pay(7L, 36_000, PaymentMethod.QRIS);
        service.advance(7L, null);
        assertEquals(OrderStatus.DEPARTED, order.getStatus());
        assertThrows(ConflictException.class,
                () -> service.pay(7L, 1_000, PaymentMethod.CASH));
    }

    @Test
    void pickupNeedsAnAddress() {
        when(customers.existsById(1L)).thenReturn(true);
        var errors = service.check(new OrderService.NewOrder(1L, null,
                BigDecimal.ONE, true, " ", null));
        assertTrue(errors.containsKey("delivery_address"));
        assertEquals(1, errors.size());
    }
}
```

`mock(OrderRepository.class)` membuat objek yang mengimplementasikan antarmuka itu dan mengembalikan jawaban kosong; `when(orders.findWithDetailsById(7L)).thenReturn(Optional.of(order))` mengajarinya satu jawaban. Jamnya ditetapkan pada 02.00 UTC, yaitu 09.00 di Surabaya, sehingga `finishingStampsTheShopTime` dapat memeriksa waktu yang persis. `unpaidLaundryCannotLeave` adalah inti bab ini dalam bentuk tes: tanpa pembayaran pada pesanan Rp 36.000, langkah ke `departed` harus ditolak dengan pesan yang persis dan statusnya harus tetap `finished`. Karena service menerima dependensinya melalui konstruktor, tes cukup memanggil `new OrderService(...)`: tanpa Spring, tanpa anotasi, dan keenam tes berjalan dalam sepersekian detik.

Jalankan semua tes dengan

```
mvn test
```

Maven mencetak berapa tes yang berjalan, gagal, dan bergalat, dan untuk setiap kegagalan nama tesnya beserta nilai yang diharapkan dan yang sebenarnya. Bab 14 menambahkan tes untuk lapisan web tersendiri, untuk kueri tersendiri, dan untuk sekuriti.

13.11 Membuatnya Lebih Baik

Mengubah daftar harga dari halaman. Bab 10 memberi pemilik di Laravel halaman daftar harga yang dilindungi *gate*; halaman Spring masih membaca daftar yang dimuat `DemoData`. `PriceController` dengan *record* formulir adalah CRUD persis seperti pelanggan, dan Bab 14 menambahkan login yang membatasinya untuk pemilik.

Memberi halaman pada daftar pelanggan. Daftar pelanggan menampilkan semua orang. Dengan beberapa ratus pelanggan tetap, tidak masalah; dengan ribuan, masalah. Repository dapat mengembalikan `Page` seperti pada pesanan.

Mengunci pesanan saat membayar. Satu transaksi per pembayaran mencegah sebagian besar pembayaran ganda. Dua kasir yang menekan "Record payment" pada detik yang sama masih mungkin sama-sama membaca sisa tagihan lama sebelum salah satunya menulis. SQLite hanya mengizinkan satu proses menulis pada satu waktu dan *pool* kita punya satu koneksi, sehingga aplikasi ini tidak dapat terkena masalah itu; di MySQL atau PostgreSQL service akan mengunci baris pesanan (`@Lock(LockModeType.PESSIMISTIC_WRITE)` pada metode repository) atau memakai kolom versi untuk *optimistic locking*.

Satu tata letak templat. Halaman-halaman mengulang `<head>` dan navigasinya melalui *fragment*. Thymeleaf Layout Dialect, atau ekspresi *fragment* Thymeleaf sendiri dengan templat tata letak, akan memberi satu tata letak tempat setiap halaman mengisi satu slot, seperti `@extends` di Blade atau `_Layout.cshtml` di Razor.

13.12 Memakainya Sungguhan

Tidak ada apa pun di bab ini yang khas laundry kecuali keempat entitas dan `LaundryRules`. Untuk menyesuaikan aplikasi dengan usaha kecil lain, bekerjalah dari aturan ke luar. Penjahit punya pelanggan, daftar harga jasa (mengecilkan pinggang, memendekkan celana, menjahit kebaya), pesanan dengan tanggal janji, dan pembayaran, biasanya uang muka pada hari itu dan sisanya saat diambil: tambahkan tanggal jatuh tempo pada pesanan, buang aturan kilogram, dan tulis "uang muka paling sedikit 30% sebelum pekerjaan dimulai" sebagai aturan dengan tesnya sendiri. Bengkel sepeda motor punya bentuk yang sama dengan nomor pelat di samping pelanggan dan suku cadang sebagai jenis baris kedua. Buku janji temu sebuah klinik kecil juga merupakan antrean dengan status (dipesan, datang, diperiksa), tetapi di sana aturannya menyangkut kesehatan orang, dan aplikasi hanya boleh mencatat dan mengingatkan, tidak pernah memutuskan: apakah seseorang perlu menemui dokter hari ini adalah hak dokter atau bidan untuk mengatakannya, bukan program.

Urutan kerjanya selalu sama: aturan dalam satu kelas biasa, teruji; lalu entitas; lalu repository; lalu satu metode service untuk setiap hal yang dikerjakan usaha itu; lalu controller dan halaman, yang semestinya berakhir sebagai bagian kode yang paling tipis.

13.13 Profitina dalam Praktik

Profitina, platform visibilitas AI yang namanya dipinjam contoh berjalan buku ini, ditulis dengan Python dan FastAPI alih-alih Spring Boot, dan menjaga pemisahan yang sama. Penangan permintaan mengubah HTTP menjadi pemanggilan; lapisan service menjalankan operasi seperti memindai situs web, menghitung skor kesiapan, atau memperpanjang langganan; basis data hanya dicapai melalui satu lapisan. Aturan yang paling penting bagi penggunanya, seperti cara langganan diperpanjang dari saldo dan batas harian yang berlaku saat mengumpulkan koin PROFIT, tinggal dalam fungsi biasa dengan tesnya sendiri, dan seluruh rangkaian tes otomatis, jauh lebih dari seribu tes, harus lulus sebelum setiap rilis. Pelajarannya berlaku lintas bahasa: bila "cucian hanya keluar bila sudah dibayar" atau "langganan hanya diperpanjang bila saldonya cukup" tinggal di satu tempat dan diuji di sana, setiap halaman dan setiap API yang memakainya langsung benar.

13.14 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Aturan di controller.** Gejalanya: `if` yang sama muncul di controller halaman dan controller JSON. Pindahkan ke metode service dan panggil dari keduanya.
- **Transaksi pada controller.** `@Transactional` tempatnya pada metode service yang mengerjakan pekerjaan; controller yang memerlukan transaksi sedang mengerjakan tugas service.
- **Dua *bean* dengan nama yang sama.** `web.OrderController` dan `api.OrderController` sama-sama menjadi *bean* `orderController`, dan aplikasi tidak menyala. Beri nama kelas yang berbeda.
- **Satu *advice* galat untuk segalanya.** `@RestControllerAdvice` tanpa `basePackages` juga menangkap eksepsi dari halaman dan menjawabnya dengan JSON. Batasi setiap *advice* pada paketnya.
- **Mengikat formulir ke entitas.** Formulir lalu dapat mengisi field yang tidak semestinya, seperti `id` atau `createdAt`. Pakai *record* formulir yang hanya berisi field yang boleh diubah formulir.
- **`BindingResult` di tempat yang salah.** Ia harus tepat mengikuti parameter `@Validated`; bila tidak, pengguna melihat halaman galat alih-alih formulir dengan pesan.
- **Menampilkan halaman sebagai jawaban `POST` yang berhasil.** Menyegarkan halaman lalu mengirim ulang. Lakukan *redirect* setelah berhasil, dan pakai atribut *flash* untuk pesannya.
- **`LazyInitializationException` di templat.** Dengan `open-in-view=false`, asosiasi yang tidak dimuat di service tidak dapat dimuat di templat. Muat di repository dengan *entity graph*.
- **`@Transactional` pada metode privat, atau pada metode yang dipanggil dari dalam kelas yang sama.** Spring menerapkan transaksi melalui *proxy* di sekeliling *bean*, sehingga hanya metode publik yang dipanggil dari *bean* lain yang dibungkus.
- **Memercayai tombol yang disembunyikan.** Tombol yang disembunyikan templat tetap dapat "ditekan" dengan permintaan buatan tangan. Service harus memeriksa setiap aturan sendiri.

13.15 Ringkasan Bab

- Arsitektur berlapis memisahkan controller (HTTP), service (operasi usaha, aturan, dan transaksi), dan repository (persistensi), dengan pemanggilan hanya ke bawah (**Gambar 13.1**, **Tabel 13.1**).
- Controller JSON Bab 12 pindah ke paket `api` dan menjadi tipis; controller halaman baru tinggal di `web`; keduanya memanggil service yang sama (**Tabel 13.2**).
- Service melempar tiga eksepsi: `RuleException` (422 atau pesan pada field formulir), `NotFoundException` (404), dan `ConflictException` (409).
- `OrderService.check` mengembalikan aturan basis data sebagai *map*, agar API maupun halaman dapat melaporkan setiap masalah sekaligus; `create` memeriksa lagi.
- Aturan uang Bab 11 tidak berubah: total menurut Persamaan (13.1), sisa tagihan menurut Persamaan (13.2).
- *Record* formulir dengan batasan Jakarta Bean Validation melindungi dari masukan yang salah dan dari *mass assignment*; `BindingResult` membuat controller dapat menampilkan formulir lagi dengan pesan batasan dan pesan service (**Tabel 13.3**).
- Templat Thymeleaf memakai ulang *fragment*, meng-*escape* teks secara bawaan, dan mengikat formulir dengan `th:object` dan `th:field`; *post* yang berhasil diikuti *redirect* (Post/Redirect/Get, **Gambar 13.2**, **Tabel 13.4**).

- Setiap lapisan punya satu *advice* galat: JSON untuk `api`, halaman 404 untuk `web`.
- Data kini tinggal di SQLite; tes memakai basis data H2 yang baru; lapisan service diuji dengan Mockito dan jam yang tetap.

13.16 Soal Latihan

1. Dengan tangan, memakai Persamaan (13.1), hitung total dari: (a) 1,2 kg setrika saja seharga Rp 6.000 per kg, tanpa antar-jemput; (b) 7,5 kg cuci-lipat ekspres seharga Rp 12.000, dengan antar-jemput; (c) 2,5 pasang sepatu seharga Rp 45.000 per pasang; (d) 4,02 kg cuci-kering-lipat seharga Rp 8.000 dengan antar-jemput.
2. Untuk setiap potongan kode, sebutkan lapisan tempatnya dan alasannya: (a) "bila pelanggan punya alamat, sarankan sebagai alamat pengantaran"; (b) "tampilkan paling banyak 30 pesanan per halaman"; (c) "jumlah semua pembayaran QRIS hari ini"; (d) "ubah kata `finished` di URL menjadi `OrderStatus.FINISHED`".
3. Tambahkan aturan: layanan ekspres (deskripsinya diawali "Express") diterima paling banyak 10 kg per pesanan. Taruh pemeriksaannya di tempat yang semestinya dan tulis tesnya di `OrderServiceTest`.
4. Seorang kasir mencari "0812 3456" dan tidak menemukan siapa pun, padahal pelanggan `081234560001` ada. Jelaskan mengapa `CustomerService.search` sebenarnya menemukannya, dan apa yang terjadi dengan teks pencarian "08123 x".
5. Jelaskan apa yang akan salah bila `OrderService.pay` *tidak* transaksional dan dua kasir mencatat pembayaran untuk pesanan yang sama pada saat bersamaan, dan mengapa aplikasi ini tetap aman.
6. Tulis *record* formulir dan daftar berkas untuk halaman daftar harga (`/prices`) tempat pemilik mengubah harga sebuah layanan dan menonaktifkannya.
7. Tulis tes di `OrderServiceTest` yang memeriksa bahwa pesanan baru untuk layanan yang sudah ditarik ditolak dengan pesan "The selected service is invalid.".

13.17 Penyelesaian Soal Latihan

1. (a) $h = 120$, per kg, sehingga $b = \max(120,300) = 300$; $S = \lfloor(6000 \cdot 300 + 50)/100\rfloor = 18000$; tanpa antar-jemput, $T = 18000$. (b) $h = 750$, $b = 750$, $S = \lfloor(12000 \cdot 750 + 50)/100\rfloor = 90000$; dengan antar-jemput dan $S < 100000$, $T = 100000$. (c) Pasang adalah satuan utuh: $h = 250$ dibulatkan ke atas menjadi $b = 300$, $S = 45000 \cdot 3 = 135000$, $T = 135000$. (d) $h = 402$, $b = 402$, $S = \lfloor(8000 \cdot 402 + 50)/100\rfloor = \lfloor 3216050/100\rfloor = 32160$; dengan antar-jemput, $T = 42160$.

2. (a) Service (atau controller halaman, sebagai kemudahan bagi formulir): menyarankan alamat adalah urusan presentasi, tetapi bila menjadi aturan ("antar ke alamat pelanggan kecuali disebut alamat lain") tempatnya di `OrderService.create`. (b) Controller: ukuran halaman adalah pilihan layar; service menerima halaman dan ukurannya sebagai parameter. (c) Repository: kueri dengan `sum` dan `where` pada metode dan tanggal, dipanggil oleh metode service yang membangun laporan. (d) Controller: ini soal teks sebuah URL.

3. Aturan ini memerlukan deskripsi layanan, sehingga tempatnya di `OrderService.check`, setelah pencarian layanan:

```
        services.findById(in.serviceId() == null ? "" : in.serviceId())
            .filter(s -> s.getDescription().startsWith("Express"))
            .filter(s -> in.quantity() != null
                    && in.quantity().compareTo(new BigDecimal("10")) > 0)
            .ifPresent(s -> add(errors, "quantity",
                    "Express service takes at most 10 kg per order."));
```

Sebuah tes, dengan *mock* yang diajari mengembalikan layanan ekspres:

```
    @Test
    void expressIsLimitedToTenKilos() {
        when(customers.existsById(1L)).thenReturn(true);
        when(services.findById("S0002")).thenReturn(Optional.of(new Service(
                "S0002", "Express wash & fold, 1 day (per kg)", 12000)));
        var errors = service.check(new OrderService.NewOrder(1L, "S0002",
                new BigDecimal("12"), false, null, null));
        assertEquals("Express service takes at most 10 kg per order.",
                errors.get("quantity").get(0));
    }
```

4. "0812 3456" cocok dengan `[0-9 +-]+`, sehingga `search` memangkasnya menjadi angka "08123456" dan memanggil `findByPhoneContainingOrderByName`, yang menemukan `081234560001`. "08123 x" memuat huruf, sehingga diperlakukan sebagai nama dan dicari dengan `findByNameContainingIgnoreCaseOrderByName`, yang tidak menemukan siapa pun kecuali ada nama yang memuat "08123 x".

5. Tanpa transaksi, kasir A membaca sisa tagihan (Rp 34.000), kasir B membacanya sesaat kemudian (masih Rp 34.000), keduanya melihat bahwa Rp 34.000 mereka tidak melebihi sisa tagihan, dan keduanya menyisipkan pembayaran: pesanan dibayar dua kali. Dengan `@Transactional` pada basis data yang mengisolasi transaksi dan kunci pada baris pesanan (Subbab 13.11), B menunggu A lalu melihat sisa tagihan 0. Di sini SQLite hanya mengizinkan satu proses menulis pada satu waktu dan *pool*-nya punya satu koneksi, sehingga kedua pemanggilan `pay` berjalan bergantian.

6. *Record*-nya:

```
public record PriceForm(
        @NotBlank @Size(max = 60) String description,
        @NotNull @Min(value = 500, message = "At least Rp 500.") Long price,
        boolean active) {
}
```

Berkas-berkasnya: `Service` mendapat metode `change(String description, long price, boolean active)`; sebuah `PriceService` (daftar semua, ambil satu, ubah, dengan `@org.springframework.stereotype.Service` yang ditulis lengkap); `PriceController` untuk `GET /prices`, `GET /prices/{id}/edit`, dan `POST /prices/{id}`; templat `prices/list.html` dan `prices/form.html`; sebuah tautan di `fragments.html`. Tidak ada hapus: layanan yang pernah dipesan harus tetap ada, sehingga formulir menawarkan "aktif" sebagai gantinya. Karena setiap pesanan menyimpan totalnya sendiri, mengubah harga tidak pernah mengubah nota lama.

7. Di `OrderServiceTest`:

```
    @Test
    void aWithdrawnServiceIsRefused() {
        Service old = new Service("S0009", "Blanket (per item)", 25000);
        old.withdraw();                // an assumed method: active = false
        when(customers.existsById(1L)).thenReturn(true);
        when(services.findById("S0009")).thenReturn(Optional.of(old));
        var errors = service.check(new OrderService.NewOrder(1L, "S0009",
                BigDecimal.ONE, false, null, null));
        assertEquals("The selected service is invalid.",
                errors.get("service_id").get(0));
    }
```

`Service` dari Bab 11 tidak punya metode yang menonaktifkan layanan, sehingga tes ini mengandaikan metode `withdraw` yang toh diperlukan halaman harga di Soal 6 (metode itu menetapkan `active` menjadi `false`).

Daftar Pustaka

[1] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley, 2002, ch. 9, “Service Layer.”

[2] Broadcom, “Spring Boot Reference Documentation,” version 4.1. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.spring.io/spring-boot/

[3] Broadcom, “Spring Data JPA Reference Documentation: Query Methods and Entity Graphs.” Accessed: Sep. 25, 2026. [Online]. Available: https://docs.spring.io/spring-data/jpa/reference/

[4] Eclipse Foundation, *Jakarta Validation 3.1 Specification*, 2024. Accessed: Sep. 25, 2026. [Online]. Available: https://jakarta.ee/specifications/bean-validation/3.1/

[5] R. Fielding, M. Nottingham, and J. Reschke, “HTTP Semantics,” RFC 9110, IETF, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9110

[6] The Thymeleaf Team, “Tutorial: Thymeleaf + Spring,” Thymeleaf 3.1. Accessed: Sep. 25, 2026. [Online]. Available: https://www.thymeleaf.org/documentation.html

[7] Broadcom, “Spring Framework Reference: Web on Servlet Stack (Spring MVC), Validation and Exceptions,” version 7.0. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.spring.io/spring-framework/reference/web/webmvc.html

[8] endoflife.date, “Spring Boot,” release and support dates. Accessed: Sep. 25, 2026. [Online]. Available: https://endoflife.date/spring-boot

[9] Xerial, “SQLite JDBC Driver,” release 3.53.2.0. Accessed: Sep. 25, 2026. [Online]. Available: https://github.com/xerial/sqlite-jdbc

[10] Hibernate, “Hibernate ORM Community Dialects: SQLiteDialect.” Accessed: Sep. 25, 2026. [Online]. Available: https://github.com/hibernate/hibernate-orm/tree/main/hibernate-community-dialects

Bab 14

Spring Boot III: Spring Security, Desain REST API, dan Pengujian Otomatis

Bab 13 memberi aplikasi laundry Spring Boot halaman untuk konter dan lapisan service di bawah halaman maupun API. Semua di dalamnya masih terbuka bagi siapa pun yang dapat menjangkau server: orang lewat yang tersambung ke Wi-Fi toko dapat membuka antrean, dan program apa pun dapat menerima pesanan melalui API. Bab ini menutup pintu-pintunya lalu memeriksa, secara otomatis, bahwa pintu itu tetap tertutup. Di akhir bab Anda akan mampu menyimpan kata sandi dengan aman dan menjelaskan mengapa *hash* saja tidak cukup tanpa garam (*salt*) dan *cost*; melindungi API JSON dengan token *bearer* bergaya Laravel Sanctum dan halaman dengan formulir login dan sesi; mengonfigurasi dua rantai filter Spring Security dan mengikuti sebuah permintaan melewatinya; membatasi sebuah operasi hanya untuk pemilik, baik di URL maupun di service; mempertahankan keputusan-keputusan di balik REST API Bab 12 (URL, metode, kode status, amplop) dan menyebut apa yang masih kurang; serta menulis tes pada empat tingkat, dari tes unit biasa sampai tes seluruh aplikasi, dan mengetahui apa yang dapat dan tidak dapat ditemukan setiap tingkat.

14.1 Titik Berangkat Kita

Bab 12 membuat API Spring Boot berbicara dengan JSON yang sama seperti API Laravel di Bab 9 dan 10, dan Bab 13 memasang lapisan service di bawahnya dan halaman di atasnya. Satu bagian API Laravel sengaja ditinggalkan: pintu-pintunya. Di Bab 10, API Laravel menjawab setiap permintaan tanpa token yang sah dengan `401 {"message": "Unauthenticated."}`, aplikasi ponsel para sopir masuk dengan `POST /api/tokens` dan menerima token, dan hanya pemilik yang boleh melakukan hal-hal tertentu. Raka, keponakan Dimas yang menulis aplikasi ponsel para sopir, langsung melihat bedanya: terhadap server Spring, aplikasinya berjalan tanpa login sama sekali. Itu praktis untuk demonstrasi dan tidak dapat diterima bagi toko yang menyimpan nomor telepon dan alamat pelanggannya.

Bu Wati punya dua keinginan. Para kasir sebaiknya login di konter dengan akun masing-masing, agar kelak toko dapat melihat siapa yang menerima pesanan mana; dan menghapus pelanggan, operasi jarang yang membuang data seseorang, sebaiknya hanya untuk dirinya. Semua hal lain yang dilakukan kasir hari ini boleh tetap dilakukan kasir.

Kode bab ini ada di `Chapter14/spring`. Ia bermula sebagai salinan Bab 13 dan menambahkan paket `security`, dua entitas (`Staff` dan `ApiToken`), controller token, halaman login, dan tes pada empat tingkat.

14.2 Gagasannya dalam Bahasa Sehari-hari

Sekuriti dalam aplikasi web menjawab dua pertanyaan untuk setiap permintaan, dalam urutan ini: *siapa yang bertanya?* dan *bolehkah ia melakukan ini?* Yang pertama adalah **autentikasi**:

permintaan membuktikan sebuah identitas, dengan kata sandi, *cookie* yang diterbitkan server setelah kata sandi, atau token. Yang kedua adalah **kendali akses**: berdasarkan identitas itu, aplikasi memutuskan apakah operasi diizinkan, di sini menurut *peran* staf. Kedua jawaban menghasilkan dua penolakan klasik HTTP: 401 ketika pertanyaan pertama tidak terjawab ("saya tidak tahu siapa Anda"), 403 ketika jawaban kedua adalah tidak ("saya tahu siapa Anda, dan Anda tidak boleh") [1].

Dua jenis klien menjangkau laundry, dan keduanya membuktikan identitas dengan cara berbeda.

Peramban di konter mendapat formulir login. Setelah e-mail dan kata sandi yang benar, server membuat *sesi*, menyimpan fakta "sesi ini milik Kasir 1" di memorinya, dan memberi peramban sebuah *cookie* berisi pengenal acak sesi itu. Peramban mengirim *cookie* itu bersama setiap permintaan berikutnya secara otomatis. Halaman menjadi praktis dengan cara ini, dan kepraktisan itu membawa bahaya yang disebut *cross-site request forgery*, dibahas di Subbab 14.5.4.

Program, seperti aplikasi ponsel Raka atau integrasi kantor akuntan, tidak memerlukan formulir dan *cookie*. Program mengirim e-mail dan kata sandinya sekali ke `POST /api/tokens` dan menerima *token* acak yang panjang, yang kemudian ditaruhnya di *header* setiap permintaan: `Authorization: Bearer 12|5Rk...`. Server tidak menyimpan sesi: setiap permintaan membawa buktinya sendiri. Aplikasi menyimpan token alih-alih kata sandi, dan mengeluarkan satu ponsel berarti menghapus satu token tanpa memengaruhi perangkat lain.

Spring Security mengimplementasikan keduanya dengan *rantai filter*: daftar filter yang dilewati setiap permintaan sebelum sampai ke controller [2]. Satu filter membaca *cookie* sesi, satu lagi membaca formulir login, filter buatan kita membaca token *bearer*; filter berikutnya membandingkan identitas yang mereka temukan dengan aturan konfigurasi dan meloloskan permintaan atau menjawab 401 atau 403. Sebuah aplikasi boleh punya beberapa rantai, masing-masing untuk sebagian URL-nya, dan inilah yang kita perlukan: satu rantai untuk `/api/**` dengan token dan tanpa sesi, satu untuk semua yang lain dengan formulir login dan sesi.

> **DEFINISI. Rantai filter sekuriti** adalah daftar terurut filter *servlet* yang dijalankan Spring Security untuk setiap permintaan yang cocok dengan rantai itu. Rantai pertama yang *matcher*-nya menerima permintaan akan menanganinya; filter-filter di rantai itu menetapkan siapa pemanggilnya dan memutuskan apakah permintaan boleh sampai ke aplikasi.

14.3 Akun dan Kata Sandi

14.3.1 Jangan pernah menyimpan kata sandi

Aturan pertama tentang akun adalah bahwa basis data tidak boleh memuat kata sandi. Basis data bisa bocor: lewat cadangan yang disalin ke laptop, kesalahan dalam kueri, atau serangan. Banyak orang memakai ulang kata sandi, sehingga daftar kata sandi yang bocor dari sebuah laundry membuka e-mail dan rekening bank mereka. Yang disimpan aplikasi sebagai gantinya adalah *hash kata sandi*: keluaran fungsi satu arah yang mudah dihitung dari kata sandi dan praktis mustahil dibalik. Saat login, aplikasi meng-*hash* apa yang diketik dan membandingkan hasilnya dengan *hash* yang tersimpan.

Hash kriptografis biasa seperti SHA-256 tidak cukup, karena dua alasan. Dua pelanggan dengan kata sandi yang sama akan punya *hash* yang sama, dan penyerang dapat menyiapkan tabel *hash* jutaan kata sandi umum lebih dulu. Jawaban untuk keduanya adalah *garam* (*salt*), nilai acak yang disimpan di samping *hash* dan dicampurkan ke dalamnya, sehingga kata sandi yang sama

menghasilkan *hash* berbeda dan tidak ada tabel yang dapat disiapkan lebih dulu. Alasan kedua adalah kecepatan: kartu grafis modern menghitung miliaran *hash* SHA-256 per detik, sehingga penyerang yang punya *hash* dan garamnya masih dapat mencoba setiap kata sandi pendek. Karena itu fungsi *hash* kata sandi seperti bcrypt, scrypt, Argon2, dan PBKDF2 sengaja dibuat *lambat*, dengan *cost* yang dapat diatur. Panduan OWASP mencantumkannya dan menganjurkan faktor kerja yang sesuai [3].

Cost bcrypt adalah pangkat dua. Dengan *cost* c, satu pemeriksaan menjalankan sekitar

$$N = 2^c \tag{14.1}$$

putaran penyiapan kunci yang mahal, sehingga setiap langkah c melipatgandakan pekerjaan bagi server maupun penyerang. `BCryptPasswordEncoder` Spring memakai $c = 10$ secara bawaan [2]; bawaan Laravel adalah 12 [4]; ASP.NET Core Identity memakai PBKDF2 dengan HMAC-SHA512 dan 100.000 iterasi [5]. *Cost* yang membuat satu login memakan beberapa puluh milidetik tidak terasa bagi kasir dan membuat tebak-tebakan menjadi mahal.

Spring Security membungkus ini dalam antarmuka `PasswordEncoder`. Konfigurasi di Subbab 14.5.1 meminta satu dari `PasswordEncoderFactories.createDelegatingPasswordEncoder()`. *Encoder* itu menghasilkan nilai tersimpan seperti `{bcrypt}$2a$10$N9qo8uLOickgx2ZMRZoMye...`: bagian dalam kurung kurawal menyebut algoritmenya, `$2a$` adalah versi bcrypt, `10` *cost*-nya, lalu 22 karakter garam dan 31 karakter *hash*. Karena algoritmenya disebut dalam nilai tersimpan, versi aplikasi berikutnya dapat mengganti bawaannya menjadi Argon2 dan tetap memeriksa setiap *hash* lama, memutakhirkan masing-masing saat pemiliknya login lagi.

14.3.2 Staf dan peran

Kode 14.1 dan **Kode 14.2** adalah akun-akunnya.

Kode 14.1 `Role.java`: dua peran sudah cukup bagi toko.

```
package com.profitina.laundry.domain;

/** Pemilik menjalankan toko; kasir menerima pesanan dan uang. */
public enum Role {
    OWNER, CASHIER
}
```

Kode 14.2 `Staff.java`: orang yang boleh login, dengan *hash* kata sandi dan peran.

```
package com.profitina.laundry.domain;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.EnumType;
import jakarta.persistence.Enumerated;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

/** Orang yang boleh login. Kata sandinya sendiri tidak pernah disimpan. */
@Entity
@Table(name = "staff")
public class Staff {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true, length = 120)
    private String email;

    @Column(nullable = false, length = 80)
    private String name;

    @Column(nullable = false, length = 100)
    private String passwordHash;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 10)
    private Role role;

    protected Staff() {
    }

    public Staff(String email, String name, String passwordHash, Role role) {
        this.email = email;
        this.name = name;
        this.passwordHash = passwordHash;
        this.role = role;
    }
```

Kode 14.2 (Lanjutan) *Staff.java: orang yang boleh login, dengan hash kata sandi dan peran.*

```
    public Long getId() { return id; }
    public String getEmail() { return email; }
    public String getName() { return name; }
    public String getPasswordHash() { return passwordHash; }
    public Role getRole() { return role; }
}
```

`Staff` adalah entitas yang terpisah dari `Customer`: pelanggan tidak pernah login, dan mencampur keduanya akan memberi setiap baris pelanggan kolom kata sandi dan peran yang tidak bermakna. Field-nya bernama `passwordHash`, bukan `password`, agar tidak ada yang tergoda menyimpan teks aslinya di sana. Peran disimpan sebagai teks (`@Enumerated(EnumType.STRING)`), sehingga kolomnya berbunyi `OWNER` alih-alih `0`, dan menambah peran kelak tidak mengubah makna baris yang sudah ada. **Kode 14.3** mencari staf menurut e-mail tanpa memedulikan huruf besar-kecil, karena orang sama seringnya mengetik `Owner@...` dan `owner@...`.

Kode 14.3 `StaffRepository.java`: mencari staf menurut e-mail.

```
package com.profitina.laundry.repository;

import com.profitina.laundry.domain.Staff;
import java.util.Optional;
import org.springframework.data.jpa.repository.JpaRepository;

public interface StaffRepository extends JpaRepository<Staff, Long> {

    Optional<Staff> findByEmailIgnoreCase(String email);
}
```

14.3.3 Memberi tahu Spring Security tentang staf kita

Formulir login halaman sepenuhnya ditangani Spring Security; ia hanya perlu tahu cara menemukan sebuah akun. Itulah tugas `UserDetailsService` (**Kode 14.4**).

Kode 14.4 `StaffDetailsService.java`: dari e-mail ke pandangan Spring Security tentang seorang pengguna.

```
package com.profitina.laundry.security;

import com.profitina.laundry.domain.Staff;
import com.profitina.laundry.repository.StaffRepository;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Component;

/** Memberi tahu formulir login cara mencari staf menurut e-mail. */
@Component
public class StaffDetailsService implements UserDetailsService {

    private final StaffRepository staff;

    public StaffDetailsService(StaffRepository staff) {
        this.staff = staff;
    }

    @Override
    public UserDetails loadUserByUsername(String email) {
        Staff s = staff.findByEmailIgnoreCase(email)
                .orElseThrow(() -> new UsernameNotFoundException(email));
        return User.withUsername(s.getEmail())
                .password(s.getPasswordHash())
                .roles(s.getRole().name())       // menjadi ROLE_OWNER dst.
                .build();
    }
}
```

`User.withUsername(...).roles("OWNER")` membuat pengguna yang memegang izin `ROLE_OWNER`, dalam istilah Spring sebuah `GrantedAuthority`. Pemeriksaan peran Spring Security seperti `hasRole("OWNER")` menambahkan awalan `ROLE_` sendiri, sehingga di tempat lain peran ditulis tanpanya. Ketika formulir dikirim, Spring Security memanggil `loadUserByUsername`, menanyakan kepada *password encoder* apakah kata sandi yang diketik cocok dengan *hash* tersimpan, dan, bila cocok, memulai sesi. Ketika tidak ada staf dengan e-mail itu, metode melempar `UsernameNotFoundException`; Spring Security mengubah "e-mail tidak ada" maupun "kata sandi salah" menjadi pesan yang sama, sehingga formulir tidak memberi tahu penyerang e-mail mana yang ada.

Dua akun demo dibuat pada nyala pertama oleh **Kode 14.5**.

Kode 14.5 `DemoStaffSeeder.java`: pemilik dan satu kasir, dengan kata sandi yang di-*hash*.

```
package com.profitina.laundry.security;

import com.profitina.laundry.domain.Role;
import com.profitina.laundry.domain.Staff;
import com.profitina.laundry.repository.StaffRepository;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Component;

/** Dua akun demo, dibuat sekali. Ganti untuk pemakaian sungguhan. */
@Component
public class DemoStaffSeeder implements ApplicationRunner {

    private final StaffRepository staff;
    private final PasswordEncoder encoder;

    public DemoStaffSeeder(StaffRepository staff, PasswordEncoder encoder) {
        this.staff = staff;
        this.encoder = encoder;
    }

    @Override
    public void run(ApplicationArguments args) {
        if (staff.count() > 0) {
            return;
        }
        staff.save(new Staff("owner@profitina-laundry.example", "Owner",
                encoder.encode("Owner#2026"), Role.OWNER));
        staff.save(new Staff("kasir1@profitina-laundry.example", "Kasir 1",
                encoder.encode("Kasir#2026"), Role.CASHIER));
    }
}
```

Kata sandi di-*hash* saat masuk oleh *encoder* yang sama yang memeriksanya. Alamatnya berakhiran `.example`, domain yang dicadangkan untuk dokumentasi, sehingga tidak ada kotak surat sungguhan yang terlibat. Kata sandi demo tercetak di buku ini dan di README: hal pertama yang dilakukan toko sungguhan adalah menggantinya.

14.4 Token untuk API

14.4.1 Gagasan token Sanctum

Bab 10 memakai token API Laravel Sanctum, dan versi Spring menyalin desainnya agar aplikasi Raka tidak menyadari server mana yang diajaknya bicara. Token punya dua bagian yang dipisahkan garis tegak, `12|5RkX...`: nomor baris di tabel `api_tokens`, dan rahasia acak sepanjang 40 karakter. Tabel menyimpan pemilik baris, nama perangkat, waktu pembuatan dan pemakaian terakhir, serta *hash* SHA-256 dari rahasianya, tidak pernah rahasia itu sendiri [4]. Ketika sebuah permintaan tiba dengan token, server mencari barisnya menurut nomor, meng-*hash* rahasia yang diberikan, dan membandingkan kedua *hash*.

Mengapa *hash* yang cepat dapat diterima di sini padahal Subbab 14.3.1 bersikeras memakai yang lambat untuk kata sandi? Karena rahasianya tidak dipilih manusia. Tiga puluh bita acak membawa

$$H = 30 \times 8 = 240 \text{ bit} \tag{14.2}$$

keacakan. Menebaknya berarti mencoba sekitar 2^{239} nilai, yang tidak mungkin dilakukan secepat apa pun, sehingga tidak ada yang perlu dilindungi oleh *hash* yang lambat. *Hash* itu hanya memastikan bahwa salinan basis data tidak memuat token yang dapat dipakai.

14.4.2 Tabel token dan service token

Kode 14.6 adalah satu baris tabel itu, dan **Kode 14.7** repository-nya.

Kode 14.6 `ApiToken.java`: kunci API milik satu perangkat.

```
package com.profitina.laundry.domain;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.FetchType;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Table;
import java.time.LocalDateTime;

/** Kunci API milik satu perangkat, seperti yang disimpan Laravel Sanctum. */
@Entity
@Table(name = "api_tokens")
public class ApiToken {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.EAGER, optional = false)
    @JoinColumn(name = "staff_id")
    private Staff staff;

    @Column(nullable = false, length = 60)
    private String device;

    @Column(nullable = false, length = 64)
    private String tokenHash;              // SHA-256 dari bagian rahasia

    @Column(nullable = false)
    private LocalDateTime createdAt;

    private LocalDateTime lastUsedAt;

    protected ApiToken() {
    }
```

Kode 14.6 (Lanjutan) *ApiToken.java: kunci API milik satu perangkat.*

```
    public ApiToken(Staff staff, String device, String tokenHash,
                    LocalDateTime createdAt) {
        this.staff = staff;
        this.device = device;
        this.tokenHash = tokenHash;
        this.createdAt = createdAt;
    }

    public void used(LocalDateTime when) {
        lastUsedAt = when;
    }

    public Long getId() { return id; }
    public Staff getStaff() { return staff; }
    public String getDevice() { return device; }
    public String getTokenHash() { return tokenHash; }
    public LocalDateTime getCreatedAt() { return createdAt; }
    public LocalDateTime getLastUsedAt() { return lastUsedAt; }
}
```

Kode 14.7 `ApiTokenRepository.java`: metode-metode bakunya sudah cukup.

```
package com.profitina.laundry.repository;

import com.profitina.laundry.domain.ApiToken;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ApiTokenRepository extends JpaRepository<ApiToken, Long> {
}
```

Staf pemiliknya dimuat langsung (`FetchType.EAGER`) karena setiap pemakaian token perlu tahu milik siapa token itu dan apa perannya. **Kode 14.8** mengerjakan pekerjaannya.

Kode 14.8 `TokenService.java`: menerbitkan, mencari, dan mencabut token, hanya menyimpan *hash*.

```
package com.profitina.laundry.security;

import com.profitina.laundry.domain.ApiToken;
import com.profitina.laundry.domain.Staff;
import com.profitina.laundry.repository.ApiTokenRepository;
import com.profitina.laundry.repository.StaffRepository;
import com.profitina.laundry.service.RuleException;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.time.Clock;
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;
import java.util.Base64;
import java.util.HexFormat;
import java.util.Optional;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

/**
 * Token API gaya Laravel Sanctum: "id|secret" ditampilkan sekali,
 * tabel hanya menyimpan SHA-256 dari rahasianya.
 */
@Service
@Transactional
public class TokenService {

    /** Token baru dan staf pemiliknya. */
    public record Issued(String token, Staff staff) {
    }

    private static final SecureRandom RANDOM = new SecureRandom();

    private final StaffRepository staff;
    private final ApiTokenRepository tokens;
    private final PasswordEncoder encoder;
    private final Clock clock;
```

Kode 14.8 (Lanjutan) *TokenService.java: menerbitkan, mencari, dan mencabut token, hanya menyimpan hash.*

```
    public TokenService(StaffRepository staff, ApiTokenRepository tokens,
                        PasswordEncoder encoder, Clock clock) {
        this.staff = staff;
        this.tokens = tokens;
        this.encoder = encoder;
        this.clock = clock;
    }

    /** E-mail dan sandi masuk, token keluar; jawaban sama untuk tiap kegagalan. */
    public Issued issue(String email, String password, String device) {
        Staff who = staff.findByEmailIgnoreCase(email)
                .filter(s -> encoder.matches(password, s.getPasswordHash()))
                .orElseThrow(() -> new RuleException("email",
                        "These credentials do not match our records."));
        byte[] raw = new byte[30];
        RANDOM.nextBytes(raw);
        String secret = Base64.getUrlEncoder().withoutPadding()
                .encodeToString(raw);                  // 40 karakter
        ApiToken saved = tokens.save(new ApiToken(who, device,
                sha256(secret), now()));
        return new Issued(saved.getId() + "|" + secret, who);
    }

    /** Token di balik "id|secret", bila ada dan rahasianya cocok. */
    public Optional<ApiToken> find(String plain) {
        int bar = plain.indexOf('|');
        if (bar < 1) {
            return Optional.empty();
        }
        long id;
        try {
            id = Long.parseLong(plain.substring(0, bar));
        } catch (NumberFormatException e) {
            return Optional.empty();
        }
        byte[] given = sha256(plain.substring(bar + 1))
                .getBytes(StandardCharsets.US_ASCII);
        Optional<ApiToken> token = tokens.findById(id).filter(t ->
                MessageDigest.isEqual(given,           // perbandingan waktu-konstan
                        t.getTokenHash().getBytes(StandardCharsets.US_ASCII)));
        token.ifPresent(t -> t.used(now()));
        return token;
    }
```

Kode 14.8 (Lanjutan) *TokenService.java: menerbitkan, mencari, dan mencabut token, hanya menyimpan hash.*

```
    public void revoke(Long tokenId) {
        tokens.deleteById(tokenId);
    }

    static String sha256(String text) {
        try {
            return HexFormat.of().formatHex(MessageDigest.getInstance("SHA-256")
                    .digest(text.getBytes(StandardCharsets.UTF_8)));
        } catch (NoSuchAlgorithmException e) {
            throw new IllegalStateException(e);      // setiap JDK memilikinya
        }
    }

    private LocalDateTime now() {
        return LocalDateTime.now(clock).truncatedTo(ChronoUnit.SECONDS);
    }
}
```

Tiga rincian dalam `TokenService` adalah keputusan sekuriti, bukan gaya.

- `issue` memberi jawaban yang sama, “These credentials do not match our records.”, entah e-mailnya tidak dikenal atau kata sandinya salah. Ini kalimat yang dipakai Laravel, dan kalimat itu tidak memberi tahu penyerang apa pun tentang e-mail mana yang milik staf. Jawabannya adalah `RuleException` pada field `email`, sehingga penangan galat API dari Bab 13 mengubahnya menjadi 422 yang biasa dengan `errors`.
- Bita acaknya berasal dari `SecureRandom`, pembangkit Java yang kuat secara kriptografis. `java.util.Random` menghasilkan angka yang dapat ditebak dari beberapa keluaran sebelumnya dan tidak boleh dipakai untuk rahasia.
- `find` membandingkan *hash* dengan `MessageDigest.isEqual`, yang memakan waktu sama entah karakter pertama yang berbeda atau yang terakhir. `equals` biasa berhenti pada perbedaan pertama, dan dengan mengukur berapa lama server memerlukan waktu untuk menolak, penyerang secara teori dapat menemukan sebuah *hash* satu karakter demi satu karakter. Ini disebut *timing attack*.

`find` juga mencatat `lastUsedAt`, sehingga pemilik kelak dapat melihat ponsel mana yang masih memakai API dan membuang token yang berbulan-bulan tidak dipakai.

14.4.3 Filternya

Kode 14.9 menjalankan token itu.

Kode 14.9 `BearerTokenFilter.java`: dari token *bearer* ke staf yang terautentikasi.

```
package com.profitina.laundry.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.List;
import org.springframework.security.authentication
        .UsernamePasswordAuthenticationToken;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.filter.OncePerRequestFilter;

/**
 * "Authorization: Bearer 12|secret" -> staf pemilik token 12.
 * Bukan @Component: filter ini hanya boleh berjalan di rantai API.
 */
public class BearerTokenFilter extends OncePerRequestFilter {

    private final TokenService tokens;

    public BearerTokenFilter(TokenService tokens) {
        this.tokens = tokens;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain chain)
            throws ServletException, IOException {
        String header = request.getHeader("Authorization");
        if (header != null && header.startsWith("Bearer ")) {
            tokens.find(header.substring(7).trim()).ifPresent(t -> {
                var role = new SimpleGrantedAuthority(
                        "ROLE_" + t.getStaff().getRole().name());
                SecurityContextHolder.getContext().setAuthentication(
                        UsernamePasswordAuthenticationToken.authenticated(
                                t.getStaff().getEmail(), t.getId(),
                                List.of(role)));
            });
        }
        chain.doFilter(request, response);    // tanpa token: tetap anonim
    }
}
```

`OncePerRequestFilter` adalah kelas dasar Spring untuk filter yang harus berjalan tepat sekali per permintaan. Filter ini membaca *header* `Authorization`; bila diawali `Bearer` dan tokennya sah, ia menaruh identitas terautentikasi di `SecurityContextHolder`, penyimpanan per permintaan tempat bagian lain Spring Security, dan controller, mengetahui siapa yang bertanya. *Principal* identitas itu adalah e-mail, *credentials*-nya nomor token (diperlukan untuk mengeluarkan perangkat ini), dan satu-satunya izinnya adalah peran beserta awalannya. Bila tidak ada *header*, atau tokennya tidak sah, filter tidak berbuat apa-apa dan meneruskan permintaan: ia tidak memutuskan apa yang boleh dilakukan permintaan anonim. Keputusan itu milik aturan rantai, yang berada di satu tempat dan lebih mudah dibaca daripada pernyataan `if` yang tersebar.

Kelas ini tidak beranotasi `@Component`, dan komentarnya menjelaskan alasannya. Spring Boot mendaftarkan setiap *bean* filter yang ditemukannya di *servlet container*, tempat filter itu akan berjalan untuk *setiap* permintaan, termasuk halaman. Dibuat dengan `new` di dalam rantai API, filter ini hanya berjalan di sana.

14.4.4 Masuk dan keluar

Kode 14.10 adalah pintu depan API.

Kode 14.10 `TokenApiController.java`: `POST /api/tokens` memberi token, `DELETE /api/tokens/current` mengambilnya kembali.

```
package com.profitina.laundry.api;

import com.profitina.laundry.security.TokenService;
import jakarta.validation.Valid;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;
import org.springframework.http.HttpStatus;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

/** POST /api/tokens dan DELETE /api/tokens/current, seperti di Bab 10. */
@RestController
@RequestMapping("/api/tokens")
public class TokenApiController {

    public record TokenRequest(@NotBlank @Email String email,
                               @NotBlank String password,
                               @NotBlank @Size(max = 60) String device) {
    }

    public record UserView(String name, String role) {
    }

    public record TokenView(String token, UserView user) {
    }

    private final TokenService tokens;

    public TokenApiController(TokenService tokens) {
        this.tokens = tokens;
    }
```

Kode 14.10 (Lanjutan) *TokenApiController.java: POST /api/tokens memberi token, DELETE /api/tokens/current mengambilnya kembali.*

```
    /** Token ditampilkan sekali; hanya hash-nya yang disimpan. */
    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public TokenView store(@Valid @RequestBody TokenRequest in) {
        TokenService.Issued t = tokens.issue(in.email(), in.password(),
                in.device());
        return new TokenView(t.token(), new UserView(t.staff().getName(),
                t.staff().getRole().name().toLowerCase()));
    }

    /** Mengeluarkan perangkat ini: tokennya langsung tidak berlaku. */
    @DeleteMapping("/current")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void destroy(Authentication who) {
        tokens.revoke((Long) who.getCredentials());
    }
}
```

Jawaban atas login yang berhasil berstatus 201, karena sebuah token, *resource* baru, telah dibuat:

```
{"token":"3|kq1V0cS2...","user":{"name":"Kasir 1","role":"cashier"}}
```

Berbeda dari setiap jawaban lain API ini, jawaban ini tidak dibungkus `{"data": ...}`; controller Laravel Bab 10 mengembalikan objek polos yang sama, dan kontraknya mengikuti Laravel. Token hanya muncul dalam jawaban ini. Bila aplikasi kehilangannya, aplikasi login lagi dan menerima token baru. `DELETE /api/tokens/current` membaca nomor token dari *credentials* yang disimpan filter, menghapus barisnya, dan menjawab 204 tanpa badan; sejak saat itu token tersebut tidak membuka apa pun.

14.5 Dua Rantai Filter

14.5.1 Konfigurasinya

Kode 14.11 adalah seluruh konfigurasi sekuriti.

Kode 14.11 `SecurityConfig.java`: satu rantai untuk API, satu untuk halaman.

```
package com.profitina.laundry.security;

import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.annotation.Order;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.method.configuration.*;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.factory.PasswordEncoderFactories;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication
        .UsernamePasswordAuthenticationFilter;

/**
 * Dua rantai filter: API JSON (token bearer, tanpa sesi, tanpa CSRF)
 * dan halaman web (formulir login, cookie sesi, token CSRF di formulir).
 */
@Configuration
@EnableMethodSecurity
public class SecurityConfig {

    @Bean
    PasswordEncoder passwordEncoder() {
        return PasswordEncoderFactories.createDelegatingPasswordEncoder();
    }

    @Bean
    @Order(1)
    SecurityFilterChain api(HttpSecurity http, TokenService tokens)
            throws Exception {
        http.securityMatcher("/api/**")
            .authorizeHttpRequests(auth -> auth
                .requestMatchers(HttpMethod.POST, "/api/tokens").permitAll()
                .requestMatchers(HttpMethod.DELETE, "/api/customers/*")
                    .hasRole("OWNER")
                .anyRequest().authenticated())
            .addFilterBefore(new BearerTokenFilter(tokens),
                    UsernamePasswordAuthenticationFilter.class)
            .sessionManagement(s -> s
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .csrf(csrf -> csrf.disable())      // tanpa cookie, tanpa risiko CSRF
            .exceptionHandling(e -> e
```

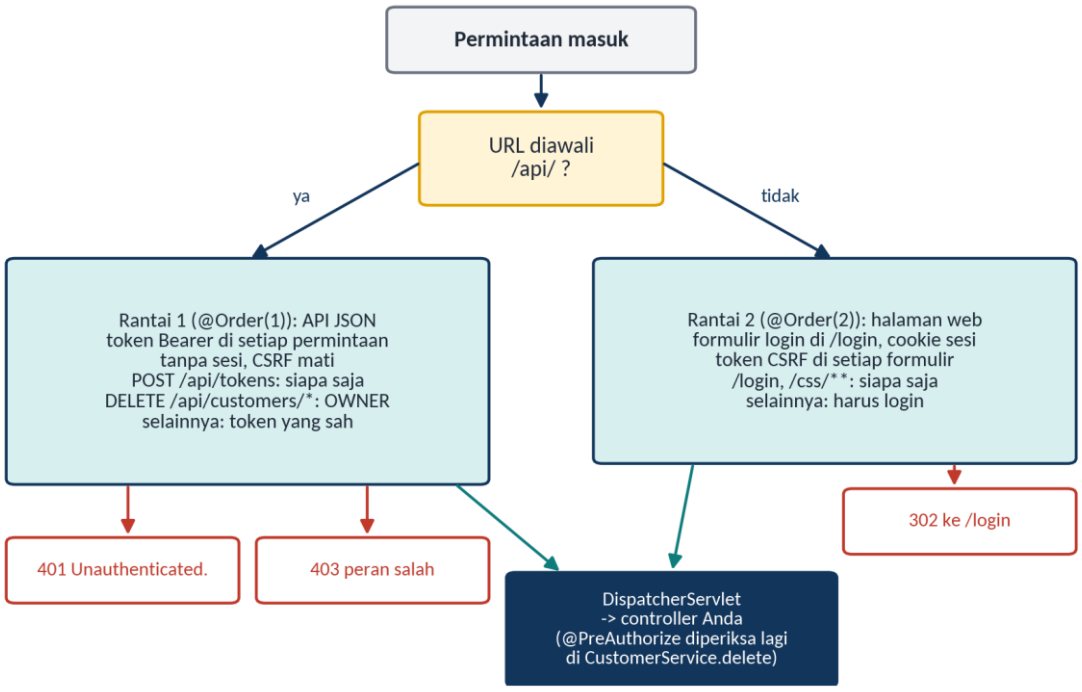
Kode 14.11 (Lanjutan) *SecurityConfig.java: satu rantai untuk API, satu untuk halaman.*

```
                .authenticationEntryPoint((request, response, ex) ->
                    json(response, 401, "Unauthenticated."))
                .accessDeniedHandler((request, response, ex) ->
                    json(response, 403, "This action is unauthorized.")));
        return http.build();
    }

    @Bean
    @Order(2)
    SecurityFilterChain web(HttpSecurity http) throws Exception {
        http.authorizeHttpRequests(auth -> auth
                .requestMatchers("/login", "/css/**", "/error").permitAll()
                .anyRequest().authenticated())
            .formLogin(form -> form
                .loginPage("/login")
                .usernameParameter("email")
                .defaultSuccessUrl("/", false))
            .logout(logout -> logout.logoutSuccessUrl("/login?logout"));
        return http.build();
    }

    /** Bentuk {"message": ...} yang sama dengan setiap galat API lain. */
    private static void json(HttpServletResponse response, int status,
                             String message) throws IOException {
        response.setStatus(status);
        response.setContentType("application/json");
        response.getWriter().write("{\"message\":\"" + message + "\"}");
    }
}
```

Ada dua *bean* bertipe `SecurityFilterChain`, dan **Gambar 14.1** memperlihatkan bagaimana sebuah permintaan menemukan rantainya.



Gambar 14.1 Dua rantai filter: permintaan di bawah `/api/` masuk ke rantai tanpa status dengan token *bearer* dan penolakan JSON; yang lain masuk ke rantai dengan formulir login, *cookie* sesi, dan perlindungan CSRF.

`@Order(1)` menaruh rantai API di depan. `securityMatcher("/api/**")`-nya mengklaim setiap URL yang diawali `/api/`; URL lain jatuh ke rantai web, yang tidak punya *matcher* sehingga menerima semua yang tersisa. Bila urutannya terbalik, rantai web juga akan mengklaim permintaan API dan menjawab aplikasi Raka dengan *redirect* ke halaman login.

Aturan rantai API dibaca dari atas ke bawah, dan aturan pertama yang cocok yang memutuskan:

1. `POST /api/tokens` terbuka bagi semua orang; seseorang tidak dapat diwajibkan punya token untuk memperoleh token.
2. `DELETE /api/customers/*` memerlukan peran `OWNER`.
3. Setiap permintaan lain memerlukan pemanggil yang terautentikasi, dengan peran apa pun.

Rantai ini menambahkan `BearerTokenFilter` sebelum `UsernamePasswordAuthenticationFilter` milik Spring, sehingga identitas sudah diketahui sebelum aturan mana pun diperiksa. `SessionCreationPolicy.STATELESS` memberi tahu Spring Security agar tidak pernah membuat sesi untuk API, sehingga setiap permintaan harus membawa tokennya. Perlindungan CSRF dimatikan untuk rantai ini saja, dengan alasan yang diberikan di Subbab 14.5.4. Terakhir, kedua penangan menulis penolakan kontrak: 401 dengan `{"message":"Unauthenticated."}` dan 403 dengan `{"message":"This action is unauthorized."}`, teks yang persis dikirim Laravel. Tanpanya Spring Security akan menjawab 401 dengan badan kosong dan 403 dengan halaman galatnya sendiri.

Rantai web lebih pendek. `/login`, *style sheet*, dan halaman galat terbuka; setiap halaman lain memerlukan staf yang sudah login. `formLogin` menyalakan pemrosesan login Spring Security: `POST /login` ditangani Spring Security, dengan membaca field `email` dan `password`; keberhasilan membawa ke halaman yang diinginkan pengguna sebelum login, atau ke `/`; kegagalan ke `/login?error`. `logout` menangani `POST /logout` dan kembali ke `/login?logout`. `@EnableMethodSecurity` pada kelas ini menyalakan pemeriksaan metode di Subbab 14.5.5.

14.5.2 Mengikuti permintaan dengan tangan

Sebelum memercayai sebuah konfigurasi, ikuti beberapa permintaan melewatinya di atas kertas. **Tabel 14.1** melakukannya untuk enam permintaan.

Tabel 14.1 Enam permintaan diikuti melewati kedua rantai.

Permintaan	Rantai	Identitas yang ditemukan	Aturan yang memutuskan	Jawaban
`GET /api/orders`, tanpa *header*	API	tidak ada	permintaan lain: terautentikasi	401 JSON
`GET /api/orders`, token kasir	API	Kasir 1, `ROLE_CASHIER`	permintaan lain: terautentikasi	200
`DELETE /api/customers/2`, token kasir	API	Kasir 1, `ROLE_CASHIER`	`DELETE /api/customers/*`: pemilik	403 JSON
`DELETE /api/customers/2`, token pemilik	API	Owner, `ROLE_OWNER`	`DELETE /api/customers/*`: pemilik	204, atau 409 bila ia punya pesanan
`GET /orders`, tanpa *cookie*	web	tidak ada	permintaan lain: terautentikasi	302 ke `/login`
`POST /customers`, sesi, tanpa field CSRF	web	Kasir 1	filter CSRF, sebelum aturan mana pun	403

Baris keempat memperlihatkan pemisahan yang berguna. Sekuriti memutuskan apakah pemilik *boleh* menghapus; service memutuskan apakah pelanggan ini *dapat* dihapus. Pelanggan yang punya pesanan ditolak dengan 409 dan "Sari Wulandari has orders and cannot be deleted." siapa pun yang meminta.

14.5.3 Halaman login

Kode 14.12 dan **Kode 14.13** menggambar formulir login; Spring Security menangani sisanya.

Kode 14.12 `LoginController.java`: hanya `GET`; Spring Security menangani `POST`.

```
package com.profitina.laundry.web;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class LoginController {

    /** Spring Security sendiri menangani POST /login; kita menggambar formnya. */
    @GetMapping("/login")
    public String login() {
        return "login";
    }
}
```

Kode 14.13 `login.html`: e-mail dan kata sandi, dengan pesan untuk login yang gagal dan untuk logout.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head th:replace="~{fragments :: head('Log in')}"></head>
<body>
<main>
  <h1>Profitina Laundry</h1>
  <p class="bad" th:if="${param.error}">Wrong e-mail or password.</p>
  <p class="ok" th:if="${param.logout}">You have logged out.</p>
  <form method="post" th:action="@{/login}">
    <label>E-mail <input name="email" type="email" autofocus required></label>
    <label>Password <input name="password" type="password" required></label>
    <button>Log in</button>
  </form>
</main>
</body>
</html>
```

`${param.error}` membaca *query string*, sehingga `/login?error` menampilkan pesannya. *Fragment* navigasi mendapat tombol logout (**Kode 14.14**). Logout adalah `POST` di dalam formulir, bukan tautan, dengan alasan yang sama bahwa menghapus tidak pernah memakai `GET`.

Kode 14.14 `fragments.html`: navigasi kini diakhiri formulir logout.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head th:fragment="head(title)">
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title th:text="${title} + ' | Profitina Laundry'">Profitina Laundry</title>
  <link rel="stylesheet" th:href="@{/css/app.css}">
</head>
<body>
<header th:fragment="nav">
  <strong>Profitina Laundry</strong>
  <nav>
    <a th:href="@{/orders(status='queued')}">Queue</a>
    <a th:href="@{/orders}">All orders</a>
    <a th:href="@{/orders/new}">New order</a>
    <a th:href="@{/customers}">Customers</a>
  </nav>
  <form method="post" th:action="@{/logout}"><button>Log out</button></form>
</header>
<div th:fragment="flash">
  <p class="ok" th:if="${message}" th:text="${message}">Saved.</p>
  <p class="bad" th:if="${error}" th:text="${error}">Not allowed.</p>
</div>
</body>
</html>
```

14.5.4 CSRF: mengapa halaman memerlukannya dan API tidak

Peramban menempelkan *cookie* sesi toko pada *setiap* permintaan ke alamat toko, halaman mana pun yang memicu permintaan itu. Misalkan Kasir 1 sedang login dan, di tab lain, membuka halaman di situs lain yang memuat formulir tersembunyi yang mengirim ke `http://laundry.local/customers/2/delete` dan sebaris skrip yang mengirimkannya. Peramban mengirim *cookie*-nya; server melihat permintaan dari kasir yang sudah login. Inilah **cross-site request forgery** (CSRF).

Pertahanannya adalah rahasia yang tidak dapat diketahui situs lain. Rantai web Spring Security memberi setiap sesi token CSRF dan menolak setiap `POST`, `PUT`, `PATCH`, atau `DELETE` yang tidak membawanya [2]. Thymeleaf menyisipkannya secara otomatis: formulir yang ditulis dengan `th:action` mendapat input tersembunyi `_csrf` berisi token itu, itulah sebabnya setiap formulir di Bab 13 dan 14 memakai `th:action` alih-alih `action` biasa [6]. Formulir di situs lain tidak dapat membaca token itu, sehingga permintaan palsunya gagal dengan 403.

Rantai API mematikan perlindungan ini, dan ini aman karena alasan yang jelas: API tidak memakai *cookie*. Token berpindah di *header* yang ditetapkan program secara eksplisit; peramban tidak pernah menambahkannya sendiri, sehingga halaman asing tidak dapat membuat peramban mengirimnya. Bila suatu hari API menerima *cookie* sesi, perlindungan CSRF harus dinyalakan kembali.

14.5.5 Garis pertahanan kedua di service

Aturan URL mencegah kasir menghapus pelanggan melalui API. Tetapi controller halaman dari Bab 13 juga punya tombol hapus, di `POST /customers/{id}/delete`, dan rantai web tidak punya aturan tentangnya. Menambahkan aturan URL untuk setiap halaman yang menghapus sesuatu mudah terlupa. Sebaliknya, service adalah satu-satunya tempat yang dilewati setiap penghapusan, sehingga pemeriksaannya juga tempatnya di sana (**Kode 14.15**).

Kode 14.15 `CustomerService.java`: `delete` hanya untuk pemilik, siapa pun yang memanggilnya.

```java
package com.profitina.laundry.service;

import com.profitina.laundry.domain.Customer;
import com.profitina.laundry.repository.CustomerRepository;
import com.profitina.laundry.repository.OrderRepository;
import java.time.Clock;
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;
import java.util.List;
import java.util.Optional;
import org.springframework.data.domain.Sort;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

/** Semua yang dilakukan toko terhadap pelanggan, di satu tempat. */
@Service
@Transactional
public class CustomerService {

    private final CustomerRepository customers;
    private final OrderRepository orders;
    private final Clock clock;

    public CustomerService(CustomerRepository customers,
                           OrderRepository orders, Clock clock) {
        this.customers = customers;
        this.orders = orders;
        this.clock = clock;
    }

    /** Menurut nama, atau menurut telepon bila teksnya angka. */
    @Transactional(readOnly = true)
    public List<Customer> search(String text) {
        String q = text == null ? "" : text.trim();
        if (q.isEmpty()) {
            return customers.findAll(Sort.by("name"));
        }
        String digits = q.replaceAll("[^0-9]", "");
        return q.matches("[0-9 +-]+")
                ? customers.findByPhoneContainingOrderByName(digits)
                : customers.findByNameContainingIgnoreCaseOrderByName(q);
    }
```

Kode 14.15 (Lanjutan) *CustomerService.java: delete hanya untuk pemilik, siapa pun yang memanggilnya.*

```
    @Transactional(readOnly = true)
    public Optional<Customer> findByPhone(String phone) {
        return customers.findByPhone(phone.replaceAll("[^0-9]", ""));
    }

    @Transactional(readOnly = true)
    public Customer get(Long id) {
        return customers.findById(id)
                .orElseThrow(() -> new NotFoundException("customer", id));
    }

    public Customer create(String name, String phone, String address) {
        if (customers.existsByPhone(phone)) {     // Laravel: aturan unique
            throw new RuleException("phone",
                    "The phone has already been taken.");
        }
        return customers.save(new Customer(name.trim(), phone,
                blankToNull(address), now()));
    }

    public Customer update(Long id, String name, String phone,
                           String address) {
        Customer customer = get(id);
        if (customers.existsByPhoneAndIdNot(phone, id)) {
            throw new RuleException("phone",
                    "The phone has already been taken.");
        }
        customer.change(name.trim(), phone, blankToNull(address));
        return customer;            // disimpan saat commit (dirty checking)
    }

    @PreAuthorize("hasRole('OWNER')")
    public void delete(Long id) {
        Customer customer = get(id);
        if (orders.existsByCustomerId(id)) {
            throw new ConflictException(customer.getName()
                    + " has orders and cannot be deleted.");
        }
        customers.delete(customer);
    }

    private LocalDateTime now() {
        return LocalDateTime.now(clock).truncatedTo(ChronoUnit.SECONDS);
    }
```

Kode 14.15 (Lanjutan) *CustomerService.java: delete hanya untuk pemilik, siapa pun yang memanggilnya.*

```
    private static String blankToNull(String s) {
        return s == null || s.isBlank() ? null : s.trim();
    }
}
```

`@PreAuthorize("hasRole('OWNER')")` membuat Spring memeriksa peran pemanggil sebelum metode berjalan dan melempar `AccessDeniedException` bila tidak cocok; rantai API mengubahnya menjadi 403 JSON, rantai web menjadi halaman galat 403-nya. Seperti `@Transactional`, anotasi ini bekerja melalui *proxy* di sekeliling *bean*, sehingga berlaku untuk pemanggilan dari *bean* lain, persis cara controller memanggil service. **Kode 14.16** menambahkan `DELETE` pada API.

Kode 14.16 `CustomerApiController.java`: `DELETE /api/customers/{id}` yang baru.

```
package com.profitina.laundry.api;

import com.profitina.laundry.domain.Customer;
import com.profitina.laundry.service.CustomerService;
import jakarta.validation.Valid;
import java.util.List;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/customers")
public class CustomerApiController {

    private final CustomerService customers;

    public CustomerApiController(CustomerService customers) {
        this.customers = customers;
    }

    @GetMapping
    public DataView<List<CustomerView>> index(
            @RequestParam(required = false) String phone) {
        List<Customer> found = phone == null ? customers.search("")
                : customers.findByPhone(phone).stream().toList();
        return new DataView<>(found.stream().map(CustomerView::of).toList());
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public DataView<CustomerView> store(
            @Valid @RequestBody CreateCustomerRequest in) {
        return new DataView<>(CustomerView.of(
                customers.create(in.name(), in.phone(), in.address())));
    }
```

Kode 14.16 (Lanjutan) *CustomerApiController.java: DELETE /api/customers/{id} yang baru.*

```
    /** Hanya pemilik (SecurityConfig dan CustomerService sama-sama begitu). */
    @DeleteMapping("/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void destroy(@PathVariable Long id) {
        customers.delete(id);
    }
}
```

Aturan URL dan aturan metode sengaja bertumpang tindih. Inilah *defense in depth*, pertahanan berlapis: bila perubahan kelak memindahkan URL atau menambah pemanggil baru, salah satu dari kedua pemeriksaan tetap berlaku. Panduan OWASP tentang kendali akses menganjurkan persis ini: tolak secara bawaan, dan periksa di server, di setiap pintu masuk [7].

14.6 Mendesain REST API

Dengan pintu-pintu terpasang, saatnya mundur selangkah dan memeriksa API itu sendiri. Bab 12 tidak menciptakan bentuknya: bab itu menyalin API Laravel agar kedua server dapat diperiksa

oleh satu kontrak. Keputusan di balik bentuk itu adalah keputusan yang dihadapi setiap perancang API, dan mengetahui alasannya memungkinkan kita mempertahankannya, atau mengubahnya dengan sadar.

14.6.1 *Resource*, URL, dan metode

REST, gaya yang dijelaskan Fielding, menata API di sekitar *resource*, hal-hal yang punya alamat, yang dibaca dan diubah klien dengan metode seragam HTTP [1]. *Resource* laundry adalah daftar harga, pelanggan, pesanan, pembayaran sebuah pesanan, dan laporan harian; token juga *resource*. **Tabel 14.2** adalah API lengkap setelah bab ini.

Tabel 14.2 API Profitina Laundry setelah Bab 14.

Metode dan URL	Makna	Berhasil	Siapa
`POST /api/tokens`	masuk, membuat token	201	siapa pun
`DELETE /api/tokens/current`	mengeluarkan perangkat ini	204	staf
`GET /api/services`	daftar harga yang aktif	200	staf
`GET /api/services/{id}`	satu layanan	200	staf
`GET /api/customers?phone=`	mencari pelanggan menurut telepon	200	staf
`POST /api/customers`	pelanggan baru	201	staf
`DELETE /api/customers/{id}`	membuang pelanggan tanpa pesanan	204	pemilik
`GET /api/orders?status=&page=`	satu halaman pesanan	200	staf
`POST /api/orders`	pesanan baru	201	staf
`GET /api/orders/{id}`	satu pesanan dengan pembayarannya	200	staf
`PATCH /api/orders/{id}/status`	langkah berikutnya	200	staf
`POST /api/orders/{id}/payments`	sebuah pembayaran	201	staf
`GET /api/reports/daily?date=`	angka-angka satu hari	200	staf

Tiga konvensi berlaku di seluruh tabel. URL menamai benda dengan kata benda jamak (`/orders`, bukan `/getOrders`); metodenya menyatakan apa yang terjadi pada benda itu. *Resource* milik *resource* lain disarangkan di bawahnya (`/orders/7/payments`), dan sarangnya berhenti pada satu tingkat. Filter dan halaman adalah parameter *query* (`?status=queued&page=2`), karena keduanya memilih dari sebuah *resource* alih-alih menamai *resource* baru.

Perubahan status adalah satu-satunya tempat desain harus memilih. Bisa saja `PATCH /api/orders/7` dengan `{"status": "finished"}`, pembaruan sebagian atas pesanan. Kontraknya memakai *sub-resource*, `PATCH /api/orders/7/status`, yang membuat jelas di log dan di aturan sekuriti bahwa permintaan ini mengubah status dan bukan yang lain. Sebagian perancang lebih suka URL aksi seperti `POST /api/orders/7/finish`; kurang REST, lebih eksplisit. Setiap pilihan dapat dipertahankan; yang penting adalah memilih sekali lalu menerapkannya di mana-mana.

14.6.2 Metode aman dan idempoten

HTTP memberi metode dua sifat yang diandalkan klien dan *proxy* [1]. Metode yang **aman** (*safe*) tidak mengubah apa pun: `GET` boleh diulang, di-*cache*, dan dimuat lebih dulu dengan bebas. Metode yang **idempoten** punya efek yang sama entah dikirim sekali atau beberapa kali: `PUT` dan

`DELETE` idempoten, `POST` tidak, dan `PATCH` idempoten hanya bila permintaan tertentu itu idempoten.

Lihat metode-metode laundry dengan sifat ini di benak. `DELETE /api/customers/5` yang dikirim dua kali menghapus sekali dan menjawab 404 pada kali kedua: keadaan server tetap sama. `PATCH /api/orders/7/status` dengan `{"status": "finished"}` idempoten dalam efeknya: pada kali kedua pesanan sudah finished, dan service menolak dengan 422 ("An order that is finished can only become departed.") tanpa mengubah apa pun. Persis inilah alasan kontrak mewajibkan klien menyebut status yang diinginkan alih-alih "maju ke langkah berikutnya": ponsel sopir yang mengulang permintaan setelah kehilangan sinyal tidak akan melompati satu langkah.

`POST /api/orders/7/payments` tidak idempoten. Bila ponsel mengirim pembayaran Rp 20.000, kehilangan sambungan sebelum jawaban tiba, lalu mengirimnya lagi, pesanan dibayar dua kali, kecuali aturan sisa tagihan kebetulan menolaknya. Subbab 14.8 menjelaskan obat bakunya, kunci idempotensi.

14.6.3 Kode status dan badan galat

Tabel 14.3 mencantumkan setiap kode status yang dipakai API.

Tabel 14.3 Kode status API dan kapan masing-masing dipakai.

Kode	Makna di API ini	Badan
200 OK	pembacaan atau perubahan status berhasil	`{"data": ...}`
201 Created	token, pelanggan, pesanan, atau pembayaran dibuat	*resource* baru
204 No Content	logout atau penghapusan berhasil	tidak ada
401 Unauthorized	tidak ada token yang sah	`{"message": "Unauthenticated."}`
403 Forbidden	token sah, tetapi perannya tidak boleh melakukan ini	`{"message": "This action is unauthorized."}`
404 Not Found	record tidak ada	`{"message": "No order 999."}`
409 Conflict	keadaan record tidak mengizinkan permintaan itu	`{"message": "Order #7 has left the shop and is closed."}`
422 Unprocessable Content	masukan melanggar aturan	`{"message": "The given data was invalid.", "errors": {...}}`

Perbedaan 422 dan 409 layak dinyatakan dengan tepat, karena keduanya sering tertukar. 422 berarti "yang Anda kirim salah": jumlah 0, antar-jemput tanpa alamat, pembayaran berlebih; klien dapat membetulkan permintaannya dan mengirim lagi. 409 berarti "yang Anda kirim mungkin baik-baik saja, tetapi *resource*-nya berada dalam keadaan yang tidak mengizinkannya": pesanan yang sudah departed tidak dapat menerima pembayaran berapa pun. RFC 9110 mendefinisikan 409 sebagai konflik dengan keadaan *resource* sasaran saat ini dan 422 sebagai isi yang bentuknya benar tetapi tidak dapat diproses [1].

Badan galat selalu punya `message` yang dapat dibaca manusia, dan untuk 422 objek `errors` yang dapat dipakai program, berkunci nama-nama field yang sama dengan yang dikirim klien. Aplikasi Raka menaruh setiap pesan di bawah input pemiliknya, seperti halaman-halaman Bab 13. Alternatifnya adalah format baku *problem details* RFC 9457, dengan `type`, `title`, `status`, dan `detail` [8]; Spring mendukungnya langsung. Laundry tidak memakainya karena kontraknya mengikuti bentuk Laravel, dan satu bentuk untuk ketiga server lebih berharga bagi Raka daripada bentuk baku di salah satunya.

14.6.4 Amplop dan halaman

Setiap jawaban yang berhasil dibungkus: `{"data": {...}}` untuk satu record, `{"data": [...], "meta": {"current_page": 1, "per_page": 15, "total": 42, "last_page": 3}}` untuk satu halaman. Amplop memakan beberapa bita dan memberi ruang untuk tumbuh: fakta penomoran halaman, dan mungkin kelak tautan atau peringatan, dapat ditambahkan di samping `data` tanpa merusak klien yang hanya membaca `data`. Sebaliknya, larik JSON polos tidak akan pernah dapat menerima field saudara. Jawaban token di Subbab 14.4.4 adalah satu-satunya pengecualian, warisan dari Laravel.

Nomor halaman dengan ukuran halaman tetap (*offset pagination*) sederhana dan membuat pengguna dapat melompat ke halaman 3. Kelemahannya muncul pada daftar yang berubah selagi dibaca: bila pesanan baru tiba ketika ponsel berada di halaman 1, halaman 2 dimulai satu pesanan lebih lambat dan satu pesanan muncul dua kali. *Cursor pagination*, "15 pesanan sesudah pesanan 812", menghindarinya dan cocok untuk gulir tanpa akhir; untuk layar konter yang mencantumkan paling banyak beberapa puluh pesanan terbuka, nomor halaman sudah memadai.

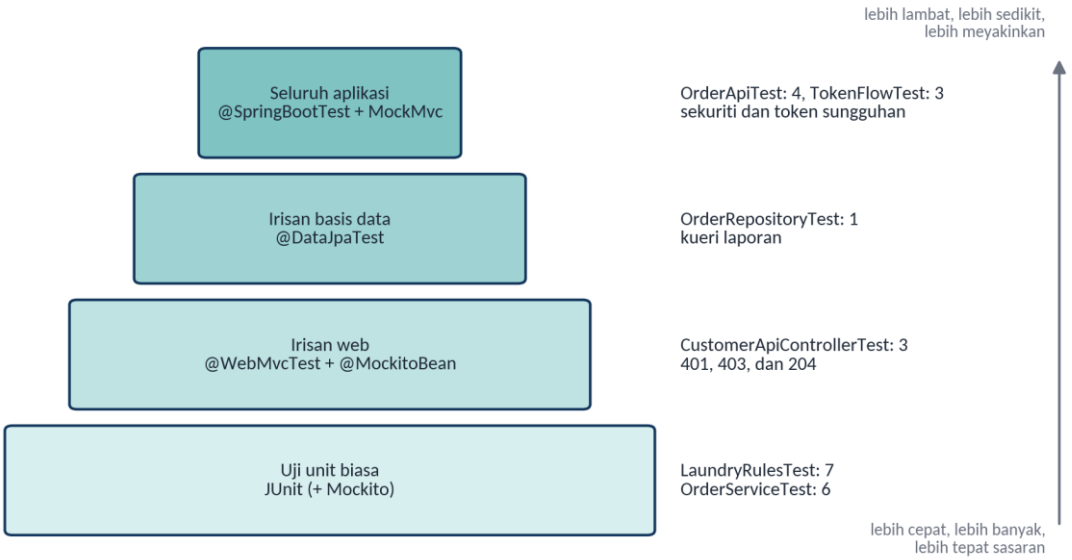
14.6.5 Yang masih kurang dari API: versi

URL-nya tidak punya nomor versi. Selama aplikasi ponsel dan server dimutakhirkan bersama, itu dapat diterima. Begitu Raka menerbitkan aplikasinya dan sebagian sopir tidak memutakhirkannya, setiap perubahan JSON harus *kompatibel ke belakang*: field boleh ditambah, tidak boleh diganti nama atau dibuang, dan field wajib baru dalam permintaan merusak aplikasi lama. Ketika perubahan yang merusak tidak terhindarkan, pendekatan yang lazim adalah melayani bentuk lama dan baru berdampingan di bawah `/api/v1/...` dan `/api/v2/...`, atau memilih versi dengan *header*, lalu memensiunkan yang lama setelah tanggal yang diumumkan. Spring Framework 7 punya dukungan bawaan untuk *versioning* API semacam ini [9]. Laundry mencatatnya sebagai perbaikan di masa depan: menambahkan `/v1` sekarang ke ketiga kerangka kerja akan mengubah kontrak tanpa ada pengguna yang memerlukannya.

14.7 Tes Otomatis pada Empat Tingkat

14.7.1 Piramidanya

Sebuah tes membuktikan satu hal tentang kode, dan tes yang menyalakan lebih banyak bagian aplikasi membuktikan lebih banyak, tetapi berjalan lebih lambat dan, ketika gagal, menunjuk penyebabnya dengan kurang tepat. Karena itu rangkaian tes yang sehat punya banyak tes kecil di bawah dan sedikit tes besar di atas, *piramida pengujian* dalam **Gambar 14.2**.



Gambar 14.2 Empat tingkat tes dalam proyek Bab 14, dari tes unit biasa di bawah sampai tes seluruh aplikasi di atas.

Tabel 14.4 menamai tingkat-tingkat itu dan apa yang dapat dan tidak dapat ditangkap masing-masing.

Tabel 14.4 Empat tingkat tes, dengan 24 run di bab ini.

Tingkat	Menyalakan	Kelas tes	Run	Menangkap	Melewatkan
Unit	hanya kelasnya	`LaundryRulesTest`, `OrderServiceTest`	13	harga dan aturan yang salah	*wiring*, SQL, HTTP
Slice web	controller, JSON, sekuriti	`CustomerApiControllerTest`	3	kode status, JSON, aturan akses	service, SQL
Slice JPA	entitas, repository, H2	`OrderRepositoryTest`	1	kueri dan pemetaan yang salah	HTTP, aturan
Seluruh aplikasi	semuanya, dengan H2	`OrderApiTest`, `TokenFlowTest`	7	*wiring* yang salah, alur ujung ke ujung	sedikit, tetapi lambat

Tes *slice* hanya menyalakan bagian Spring Boot yang diperlukan satu lapisan [10]. Tes ini lebih cepat daripada menyalakan semuanya dan gagal karena lebih sedikit alasan yang tidak berkaitan. **Kode 14.17** menambahkan *starter* yang menyediakan *slice* dan pembantu tes sekuriti; di Spring Boot 4 setiap teknologi membawa *starter* tesnya sendiri.

Kode 14.17 `pom.xml`: sekuriti, dan satu *starter* tes per *slice*.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>4.1.1</version>
    <relativePath/>
  </parent>
  <groupId>com.profitina</groupId>
  <artifactId>laundry</artifactId>
  <version>14.0.0</version>
  <properties>
    <java.version>25</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-webmvc</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-validation</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>
    <!-- SQLite untuk toko, seperti dijanjikan Bab 11 -->
    <dependency>
      <groupId>org.xerial</groupId>
      <artifactId>sqlite-jdbc</artifactId>
      <version>3.53.2.0</version>
    </dependency>
    <dependency>
```

Kode 14.17 (Lanjutan) *pom.xml: sekuriti, dan satu starter tes per slice.*

```
      <groupId>org.hibernate.orm</groupId>
      <artifactId>hibernate-community-dialects</artifactId>
      <version>${hibernate.version}</version>
    </dependency>
    <!-- H2 di memori, hanya untuk tes -->
    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-webmvc-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

Tingkat unit sudah tersedia, ditulis dengan JUnit 6 dan Mockito [11]: tes aturan dari Bab 11 dengan tujuh run (enam harga dan satu alur kerja), dan tes service dari Bab 13 dengan enam. Tingkat-tingkat lainnya menyusul.

14.7.2 Lapisan web saja

Kode 14.18 menguji controller API pelanggan bersama konfigurasi sekuriti yang sesungguhnya, dan tidak ada yang lain.

Kode 14.18 `CustomerApiControllerTest.java`: lapisan web dan aturan sekuriti, dengan service tiruan.

```
package com.profitina.laundry.api;

import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.verifyNoInteractions;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

import com.profitina.laundry.security.SecurityConfig;
import com.profitina.laundry.security.TokenService;
import com.profitina.laundry.service.CustomerService;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.webmvc.test.autoconfigure.WebMvcTest;
import org.springframework.context.annotation.Import;
import org.springframework.security.test.context.support.WithMockUser;
import org.springframework.test.context.bean.override.mockito.MockitoBean;
import org.springframework.test.web.servlet.MockMvc;

/** Lapisan web saja, dengan aturan keamanan yang sesungguhnya. */
@WebMvcTest(CustomerApiController.class)
@Import(SecurityConfig.class)
class CustomerApiControllerTest {

    @Autowired
    MockMvc mvc;

    @MockitoBean
    CustomerService customers;

    @MockitoBean
    TokenService tokens;                    // SecurityConfig memerlukannya

    @Test
    void withoutATokenTheApiSays401() throws Exception {
        mvc.perform(get("/api/customers"))
           .andExpect(status().isUnauthorized())
           .andExpect(jsonPath("$.message").value("Unauthenticated."));
    }
```

Kode 14.18 (Lanjutan) *CustomerApiControllerTest.java: lapisan web dan aturan sekuriti, dengan service tiruan.*

```
    @Test
    @WithMockUser(roles = "CASHIER")
    void aCashierMayNotDelete() throws Exception {
        mvc.perform(delete("/api/customers/2"))
           .andExpect(status().isForbidden())
           .andExpect(jsonPath("$.message")
                   .value("This action is unauthorized."));
        verifyNoInteractions(customers);
    }

    @Test
    @WithMockUser(roles = "OWNER")
    void theOwnerDeletes() throws Exception {
        mvc.perform(delete("/api/customers/2"))
           .andExpect(status().isNoContent());
        verify(customers).delete(2L);
    }
}
```

`@WebMvcTest(CustomerApiController.class)` menyalakan Spring MVC dengan controller yang satu ini, konfigurasi JSON, dan rantai filter sekuriti, tetapi tanpa service, repository, atau basis data. `@Import(SecurityConfig.class)` memastikan rantai kita yang dipakai, bukan bawaan

Spring Boot. `@MockitoBean` menaruh *mock* Mockito ke dalam *application context* menggantikan `CustomerService`, yang diperlukan controller, dan `TokenService`, yang diperlukan rantai API. `MockMvc` mengirim permintaan melewati filter dan controller tanpa jaringan.

`@WithMockUser(roles = "CASHIER")` menjalankan satu tes seolah-olah seorang kasir sedang login, tanpa token atau kata sandi. Ketiga tes mengikuti tiga baris pertama **Tabel 14.1** dan memeriksa seluruh jawaban, status dan JSON. `verifyNoInteractions(customers)` di tes kedua membuktikan bahwa permintaan kasir dihentikan *sebelum* controller: service tidak pernah dipanggil. Di *slice* ini `CustomerService` adalah *mock*, sehingga `@PreAuthorize` miliknya tidak berjalan; aturan URL sendirian harus menolak, dan tes ini menunjukkan bahwa ia memang menolak.

14.7.3 Kueri basis data

Kode 14.19 menguji kueri repository pada basis data sungguhan yang kosong.

Kode 14.19 `OrderRepositoryTest.java`: kueri laporan dan `existsByCustomerId` terhadap H2.

```
package com.profitina.laundry.repository;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

import com.profitina.laundry.domain.Customer;
import com.profitina.laundry.domain.Order;
import com.profitina.laundry.domain.OrderStatus;
import com.profitina.laundry.domain.Service;
import java.math.BigDecimal;
import java.time.LocalDateTime;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.data.jpa.test.autoconfigure.DataJpaTest;

/** Kueri saja, pada basis data memori yang baru untuk tiap tes. */
@DataJpaTest
class OrderRepositoryTest {

    @Autowired CustomerRepository customers;
    @Autowired ServiceRepository services;
    @Autowired OrderRepository orders;

    @Test
    void theDailyCountsAndTheCustomerCheck() {
        LocalDateTime nine = LocalDateTime.of(2026, 9, 25, 9, 0);
        Customer sari = customers.save(
                new Customer("Sari Wulandari", "081200000101", null, nine));
        Service wash = services.save(
                new Service("S0001", "Wash, dry & fold (per kg)", 8000));
        orders.save(new Order(sari, wash, new BigDecimal("4.5"), false, null,
                null, nine));
        Order b = new Order(sari, wash, new BigDecimal("3"), false, null,
                null, nine.plusHours(1));
        b.advance(nine.plusHours(5));                  // queued -> finished
        orders.save(b);
```

Kode 14.19 (Lanjutan) *OrderRepositoryTest.java: kueri laporan dan existsByCustomerId terhadap H2.*

```
        var counts = orders.countByStatusBetween(nine.toLocalDate()
                .atStartOfDay(), nine.toLocalDate().plusDays(1).atStartOfDay());
        assertEquals(2, counts.size());
        assertEquals(1, counts.stream()
                .filter(c -> c.getStatus() == OrderStatus.FINISHED)
                .mapToLong(OrderRepository.StatusCount::getN).sum());
        assertEquals(60_000, orders.billedBetween(nine.minusHours(9),
                nine.plusHours(15)));
        assertTrue(orders.existsByCustomerId(sari.getId()));
    }
}
```

`@DataJpaTest` menyalakan JPA, repository, dan basis data tertanam, menjalankan setiap tes dalam transaksi dan membatalkannya (*rollback*) di akhir, sehingga tes tidak dapat saling mengganggu. Tes ini menyimpan satu pelanggan, satu layanan, dan dua pesanan, menyelesaikan salah satunya, lalu memeriksa hitungan per status, nilai yang ditagih hari itu ($36.000 + 24.000 = 60.000$ menurut aturan harga Subbab 13.4.3), dan kueri yang melindungi pelanggan yang punya pesanan. Kesalahan dalam JPQL `countByStatusBetween`, atau galat pemetaan di `Order`, gagal di sini dengan pesan yang tepat, sedangkan tes seluruh aplikasi hanya akan melaporkan angka yang salah dalam sebuah laporan.

14.7.4 Seluruh aplikasi

Kode 14.20 menyalakan semuanya: *application context*, rantai sekuriti yang sesungguhnya, service, repository, dan basis data H2 baru dengan akun-akun demo.

Kode 14.20 `TokenFlowTest.java`: masuk, pakai token, keluar, lalu lihat token itu ditolak.

```
package com.profitina.laundry.security;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

import com.jayway.jsonpath.JsonPath;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.webmvc.test.autoconfigure.AutoConfigureMockMvc;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;

/** Seluruh aplikasi: akun asli, hash asli, token asli. */
@SpringBootTest
@AutoConfigureMockMvc
class TokenFlowTest {

    @Autowired
    MockMvc mvc;

    private String login(String password, int expect) throws Exception {
        return mvc.perform(post("/api/tokens")
                        .contentType(MediaType.APPLICATION_JSON)
                        .content("""
                            {"email": "kasir1@profitina-laundry.example",
                             "password": "%s", "device": "test"}"""
                               .formatted(password)))
                .andExpect(status().is(expect))
                .andReturn().getResponse().getContentAsString();
    }

    @Test
    void aTokenOpensTheApiUntilItIsRevoked() throws Exception {
        String body = login("Kasir#2026", 201);
        String bearer = "Bearer " + JsonPath.read(body, "$.token");
        mvc.perform(get("/api/orders").header("Authorization", bearer))
           .andExpect(status().isOk());
        mvc.perform(delete("/api/tokens/current")
                        .header("Authorization", bearer))
           .andExpect(status().isNoContent());
        mvc.perform(get("/api/orders").header("Authorization", bearer))
           .andExpect(status().isUnauthorized());
    }
```

Kode 14.20 (Lanjutan) *TokenFlowTest.java: masuk, pakai token, keluar, lalu lihat token itu ditolak.*

```
    @Test
    void aWrongPasswordGetsNoToken() throws Exception {
        String body = login("kasir#2026", 422);
        org.junit.jupiter.api.Assertions.assertEquals(
                "These credentials do not match our records.",
                JsonPath.read(body, "$.errors.email[0]"));
    }

    @Test
    void aMadeUpTokenIsRefused() throws Exception {
        mvc.perform(get("/api/orders")
                        .header("Authorization", "Bearer 1|not-the-secret"))
           .andExpect(status().isUnauthorized());
    }
}
```

`@SpringBootTest` dengan `@AutoConfigureMockMvc` menyalakan aplikasi lengkap dan memberi tes `MockMvc` yang melewati semua filternya. Tes pertama adalah riwayat hidup sebuah token: login

dengan kata sandi kasir yang asli, permintaan yang berhasil, logout, dan token yang sama ditolak. `JsonPath.read` mengambil token dari jawabannya. Tes kedua memeriksa bahwa kata sandi yang salah (huruf `k` kecil) menerima 422 dengan kalimat Laravel; yang ketiga, bahwa token dengan rahasia karangan tidak membuka apa pun. Tes-tes ini menangkap kesalahan yang tidak dapat ditangkap tes yang lebih kecil: filter yang didaftarkan di rantai yang salah, *password encoder* yang meng-*hash* secara berbeda saat mengisi data dan saat memeriksa, kolom token yang terlalu pendek untuk *hash*-nya.

`OrderApiTest` dari Bab 12 juga menyalakan seluruh aplikasi; setelah bab ini tes itu akan menerima 401 untuk setiap permintaan, karena tidak mengirim token. Satu anotasi pada kelasnya, `@WithMockUser(roles = "CASHIER")`, membuatnya berjalan sebagai kasir, dan keempat tesnya lulus tanpa perubahan. Tes itu memeriksa alur pesanan; `TokenFlowTest` memeriksa token; tidak satu pun harus mengerjakan tugas yang lain.

Jalankan semuanya dengan

```
mvn test
```

Surefire, pelari tes Maven, melaporkan 24 run tes dalam enam kelas, dengan satu baris per kelas dan sebuah ringkasan; laporan untuk setiap kelas ditulis ke `target/surefire-reports`. Di toko yang rutin melakukan perubahan, perintah ini dijalankan sebelum setiap rilis, dan sebaiknya pada setiap *push* ke repositori bersama.

14.8 Membuatnya Lebih Baik

Kunci idempotensi. Biarkan aplikasi ponsel mengirim *header* acak `Idempotency-Key` bersama setiap `POST`; server menyimpan kunci itu bersama jawabannya dan, ketika kunci yang sama datang lagi, mengembalikan jawaban tersimpan alih-alih membuat pembayaran kedua. Banyak API pembayaran bekerja dengan cara ini.

Masa berlaku token dan daftar perangkat. Kini token hidup sampai dicabut. Halaman tempat pemilik melihat setiap token dengan perangkat dan `lastUsedAt`-nya, dan aturan yang membuang token yang menganggur 90 hari, akan menutup celah yang ditinggalkan ponsel yang hilang.

Batasi percobaan login. `POST /api/tokens` dan formulir login menerima tebakan tanpa batas. Menghitung percobaan gagal per e-mail dan per alamat, dan menunda atau menolak setelah lima kali dalam semenit, membuat tebak-tebakan tidak praktis; *throttling* login Laravel di Bab 10 mengerjakan ini.

Mencatat siapa mengerjakan apa. Dengan adanya akun, `Order` dan `Payment` dapat menyimpan staf yang membuatnya. Service membaca namanya dari `SecurityContextHolder`, atau lebih baik, menerimanya sebagai parameter dari controller, agar tetap dapat diuji.

HTTPS. Semua di bab ini mengandaikan kata sandi, *cookie*, dan token berpindah dalam keadaan terenkripsi. Di belakang *reverse proxy* seperti nginx atau Caddy dengan sertifikat, tetapkan `server.forward-headers-strategy=native` agar Spring tahu bahwa permintaan aslinya aman.

14.9 Memakainya Sungguhan

Hampir setiap aplikasi usaha kecil memerlukan potongan-potongan bab ini dalam susunan yang sama: akun staf dengan *hash* kata sandi yang lambat, formulir login dan sesi untuk orang di depan layar, token untuk program, daftar peran yang pendek, dan pemeriksaan di URL dan di service. Konter apotek akan menambah peran apoteker, yang satu-satunya boleh menyerahkan obat

resep; kantin sekolah akan memberi orang tua token untuk aplikasi yang menampilkan saldo anaknya, setiap token terbatas pada satu anak. Ketika jumlah peran tumbuh melampaui segelintir, saatnya menyimpan *izin* ("boleh menghapus pelanggan", "boleh mengubah harga") dan memberi peran kumpulan izin, alih-alih menaburkan nama peran di seluruh kode.

Jangan membangun penyimpanan kata sandi atau skema token sendiri melampaui yang diperlihatkan bab ini. Pakai *password encoder* kerangka kerja, `SecureRandom`, dan perbandingan waktu-konstan, dan ikuti *cheat sheet* OWASP [7], [3]. Sebelum sistem dengan data pelanggan sungguhan dipakai, mintalah seseorang yang tidak ikut menulisnya mencoba membobolnya.

Satu kebiasaan kecil lagi membuat perbedaan besar: tulislah tes untuk setiap penolakan, bukan hanya untuk jalur yang berhasil. Tes yang memeriksa bahwa kasir *tidak* boleh menghapus, bahwa token yang dicabut *tidak* lagi berlaku, dan bahwa kata sandi yang salah *tidak* menghasilkan token, adalah tes yang akan gagal pada hari seseorang tanpa sengaja melonggarkan konfigurasi sekuriti. Pintu yang terbuka tidak menimbulkan keluhan dari siapa pun; hanya tes yang memperhatikannya.

14.10 Profitina dalam Praktik

Profitina memakai dua jenis pintu yang sama. Dasbor webnya memasukkan orang dengan sesi, dan API-nya menerbitkan token untuk program: server MCP yang dipakai aplikasi AI untuk membaca laporan visibilitas pengguna, dan skrip yang ditulis pelanggan terhadap API publik. Token ditampilkan sekali dan disimpan dalam bentuk *hash*, masing-masing dapat dicabut sendiri-sendiri, dan akses diperiksa dua kali, di rute dan di service yang menjalankan operasinya. Rangkaian tes otomatis platform itu, jauh lebih dari seribu tes, mengikuti piramida bab ini: banyak tes cepat untuk aturan murni seperti penskoran dan batas koin, lebih sedikit tes untuk *endpoint* tunggal, dan sekumpulan kecil alur lengkap seperti "daftar, verifikasi, jalankan pemindaian, lihat laporan", yang semuanya harus lulus sebelum rilis.

14.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Menyimpan kata sandi dengan SHA-256, atau sebagai teks biasa.** Pakai `PasswordEncoder`; *hash* cepat atas kata sandi manusia dapat dibalik dengan menebak.
- **Mendaftarkan filter buatan sendiri sebagai `@Component`.** Filter itu lalu berjalan untuk setiap permintaan, di luar rantainya. Buat dengan `new` di rantai yang memerlukannya.
- **Rantai dalam urutan yang salah.** Rantai tanpa *matcher* yang ditaruh di depan mengklaim setiap permintaan. Beri rantai yang spesifik `@Order(1)` dan `securityMatcher`.
- **Mematikan CSRF di mana-mana "karena API memerlukannya".** Hanya rantai yang tidak memakai *cookie* yang boleh tanpa perlindungan CSRF.
- **`action=` biasa di formulir Thymeleaf.** Field CSRF hanya ditambahkan dengan `th:action`; formulirnya lalu gagal dengan 403.
- **Jawaban berbeda untuk "e-mail tidak ada" dan "kata sandi salah".** Jawaban itu memberi tahu penyerang akun mana yang ada.
- **Memeriksa peran hanya di templat halaman.** Menyembunyikan tombol bukanlah kendali akses. Periksa di URL, di service, atau keduanya.
- **`hasRole("ROLE_OWNER")`.** `hasRole` menambahkan awalannya sendiri; tulis `hasRole("OWNER")`.

- **Tes yang rusak setelah sekuriti ditambahkan.** Tes seluruh aplikasi tanpa identitas kini mendapat 401. Beri `@WithMockUser` atau token sungguhan, dan tambahkan tes yang juga memeriksa penolakannya.
- **Hanya tes seluruh aplikasi.** Tes itu lambat dan gagal dengan samar. Taruh sebagian besar tes di dasar piramida.

14.12 Ringkasan Bab

- Autentikasi menetapkan siapa yang bertanya; kendali akses memutuskan apa yang boleh dilakukannya. Identitas yang tidak ada menghasilkan 401, penolakan 403.
- Kata sandi disimpan sebagai *hash* bergaram yang sengaja lambat; pekerjaan bcrypt berlipat dua untuk setiap langkah *cost*-nya (Persamaan (14.1)). *Encoder* delegasi Spring menyebut algoritmenya dalam nilai tersimpan.
- Token API bergaya Sanctum berbentuk `id|secret`; tabel hanya menyimpan SHA-256 dari rahasia 240 bit (Persamaan (14.2)), dibandingkan dalam waktu konstan.
- Dua rantai filter (**Gambar 14.1**): rantai API tanpa status, membaca token *bearer*, dan menjawab 401 dan 403 dalam JSON kontrak; rantai web memakai formulir login, sesi, dan token CSRF (**Tabel 14.1**).
- Menghapus pelanggan dibatasi untuk pemilik di URL dan dengan `@PreAuthorize` di service: pertahanan berlapis.
- Desain API mengikuti REST: kata benda di URL, metode dengan sifat aman dan idempoten yang jelas, sekumpulan kecil kode status dengan 422 untuk masukan yang salah dan 409 untuk keadaan yang bertentangan, serta amplop untuk data dan halaman (**Tabel 14.2**, **Tabel 14.3**). *Versioning* adalah hal utama yang masih kurang.
- Tes datang dalam empat tingkat (**Gambar 14.2**, **Tabel 14.4**): tes unit, *slice* web dengan aturan sekuriti sungguhan, *slice* JPA, dan seluruh aplikasi; 24 run seluruhnya.

14.13 Soal Latihan

1. Memakai Persamaan (14.1), berapa kali lebih banyak pekerjaan pemeriksaan bcrypt pada *cost* 12 dibanding *cost* 10? Bila pemeriksaan pada *cost* 10 memakan 60 ms di server toko, kira-kira berapa lama pada *cost* 12, dan akankah kasir menyadarinya?
2. Untuk setiap permintaan, sebutkan rantainya, aturan yang memutuskan, dan statusnya: (a) `POST /api/tokens` dengan kata sandi yang salah; (b) `GET /api/services` dengan token `abc`; (c) `GET /customers` di peramban yang sudah logout; (d) `POST /api/orders/7/payments` dengan token yang sudah dicabut.
3. Jelaskan mengapa rantai API boleh mematikan perlindungan CSRF tetapi rantai web tidak. Apa yang harus berubah bila aplikasi Raka memakai *cookie* sesi alih-alih token?
4. Seorang rekan mengusulkan menyimpan token asli di `api_tokens` "agar pemilik dapat melihatnya lagi". Berikan dua alasan menentang, dan usulkan apa yang semestinya ditampilkan halaman pemilik.
5. Apakah `DELETE /api/customers/5` idempoten di API ini? Jelaskan apa jawaban server pada kali pertama dan kedua, dan beri argumen apakah jawaban-jawaban itu penting bagi idempotensi.
6. Tulis tes *slice* web yang memeriksa bahwa kasir *boleh* membuat pelanggan melalui `POST /api/customers` dan menerima 201.

7. Tambahkan aturan "hanya pemilik yang boleh membaca laporan harian" dengan dua cara: sebagai aturan URL dan dengan `@PreAuthorize`. Tes mana dari bab ini yang gagal, dan apa yang harus diubah di dalamnya?

14.14 Penyelesaian Soal Latihan

1. Menurut Persamaan (14.1), $2^{12}/2^{10} = 4$ kali lebih banyak pekerjaan, sehingga sekitar $4 \times 60 = 240$ ms. Kasir login sekali per giliran kerja, sehingga seperempat detik tidak terasa; bagi penyerang, banyaknya tebakan per detik terbagi empat.

2. (a) Rantai API; `POST /api/tokens` terbuka, sehingga permintaan sampai ke controller, dan `TokenService.issue` melempar `RuleException`: 422 dengan "These credentials do not match our records." (b) Rantai API; filter tidak menemukan garis tegak di `abc`, sehingga pemanggil tetap anonim; "permintaan lain: terautentikasi" yang memutuskan: 401 JSON. (c) Rantai web; tanpa sesi; "permintaan lain: terautentikasi" yang memutuskan: *redirect* 302 ke `/login`. (d) Rantai API; baris token sudah dihapus, sehingga `find` tidak mengembalikan apa pun dan pemanggil anonim: 401 JSON.

3. CSRF bekerja karena peramban menambahkan *cookie* ke permintaan secara otomatis, situs mana pun yang memicunya. Rantai API sama sekali tidak membaca *cookie*; buktinya adalah *header* yang hanya dapat ditetapkan kode yang memegang token, dan halaman asing tidak dapat membuat peramban menambahkannya. Rantai web bersandar pada *cookie* sesi, sehingga halaman asing dapat membuat peramban mengirim permintaan dengan *cookie* itu; token CSRF-lah yang tidak dapat disediakan halaman asing. Bila aplikasi Raka memakai *cookie* sesi, rantai API harus menyalakan perlindungan CSRF dan aplikasi harus mengambil dan mengirim token itu bersama setiap permintaan yang mengubah data, atau *cookie*-nya memerlukan atribut `SameSite=Strict` dan tinjauan yang cermat.

4. Basis data yang memuat token yang dapat dipakai sama berbahayanya dengan daftar kata sandi bagi siapa pun yang memperoleh cadangannya; dan pemilik tidak perlu melihat rahasia, cukup tahu perangkat mana yang punya akses. Halamannya semestinya mencantumkan nomor setiap token, perangkat, pemilik, waktu pembuatan, dan pemakaian terakhir, dengan tombol untuk mencabutnya. Bila sebuah perangkat kehilangan tokennya, perangkat itu login lagi dan menerima token baru.

5. Ya. Permintaan pertama menghapus pelanggan dan menjawab 204; yang kedua tidak menemukan pelanggan dan menjawab 404 ("No customer 5."). Idempotensi menyangkut efek pada keadaan server, bukan jawaban yang identik: setelah satu permintaan maupun dua, pelanggan itu sudah tidak ada dan tidak ada hal lain yang berubah. Karena itu klien yang mengulang `DELETE` setelah kehilangan jawaban sebaiknya memperlakukan 404 sebagai "sudah selesai".

6. Di `CustomerApiControllerTest`, dengan *import* untuk `post`, `MediaType`, dan `jsonPath`:

```
    @Test
    @WithMockUser(roles = "CASHIER")
    void aCashierMayAddACustomer() throws Exception {
        when(customers.create("Rina", "081300000001", null)).thenReturn(
                new Customer("Rina", "081300000001", null,
                        LocalDateTime.of(2026, 9, 25, 9, 0)));
        mvc.perform(post("/api/customers")
                        .contentType(MediaType.APPLICATION_JSON)
                        .content("""
                            {"name": "Rina", "phone": "0813-0000-0001"}"""))
           .andExpect(status().isCreated())
           .andExpect(jsonPath("$.data.phone").value("081300000001"));
    }
```

Telepon dipangkas menjadi angka oleh *record* permintaan sebelum service dipanggil, sehingga *mock* diajari nilai yang hanya berisi angka. CSRF tidak mengganggu, karena rantai API mematikannya.

7. Sebagai aturan URL, tambahkan `.requestMatchers(HttpMethod.GET, "/api/reports/**").hasRole("OWNER")` sebelum `anyRequest()` di rantai API. Dengan `@PreAuthorize`, anotasikan `ReportService.daily` dengan `@PreAuthorize("hasRole('OWNER')")`. Dengan cara mana pun, `OrderApiTest.theDailyReportNeedsADate` berjalan sebagai kasir dan kini menerima 403 alih-alih 422 atau 200. Beri tes yang satu itu `@WithMockUser(roles = "OWNER")` sendiri, dan tambahkan tes yang memeriksa bahwa kasir menerima 403 dengan “This action is unauthorized.”. Melakukan keduanya, seperti untuk menghapus pelanggan, juga melindungi laporan dari halaman kelak yang menampilkannya.

Daftar Pustaka

[1] R. Fielding, M. Nottingham, and J. Reschke, “HTTP Semantics,” RFC 9110, IETF, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9110

[2] Broadcom, “Spring Security Reference: Servlet Applications, Architecture, Password Storage and CSRF,” version 7.0. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.spring.io/spring-security/reference/

[3] OWASP Foundation, “Password Storage Cheat Sheet,” OWASP Cheat Sheet Series. Accessed: Sep. 25, 2026. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

[4] Laravel, “Laravel Sanctum” and “Hashing,” Laravel 13.x documentation. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/sanctum

[5] Microsoft, “Introduction to Identity on ASP.NET Core” and “PasswordHasher,” ASP.NET Core 10 documentation. Accessed: Sep. 25, 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/security/authentication/identity

[6] The Thymeleaf Team, “Thymeleaf + Spring Security integration basics.” Accessed: Sep. 25, 2026. [Online]. Available: https://www.thymeleaf.org/doc/articles/springsecurity.html

[7] OWASP Foundation, “Authorization Cheat Sheet,” OWASP Cheat Sheet Series. Accessed: Sep. 25, 2026. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

[8] M. Nottingham, E. Wilde, and S. Dalal, “Problem Details for HTTP APIs,” RFC 9457, IETF, Jul. 2023. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9457

[9] Broadcom, “Spring Framework Reference: Web on Servlet Stack, API Versioning,” version 7.0. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.spring.io/spring-framework/reference/web/webmvc-versioning.html

[10] Broadcom, “Spring Boot Reference Documentation: Testing,” version 4.1. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.spring.io/spring-boot/reference/testing/

[11] JUnit Team, “JUnit User Guide,” JUnit 6. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.junit.org/

Bab 15

Membandingkan Tiga Kerangka Kerja: Kinerja, Produktivitas, Ekosistem, Memilih

Profitina Laundry kini ada tiga kali: dalam ASP.NET Core, dalam Laravel, dan dalam Spring Boot. Setelah membangun toko yang sama tiga kali, Anda dapat membandingkan kerangka kerja berdasarkan pengalaman, bukan kabar burung, dan bab ini menunjukkan cara melakukannya dengan jujur. Di akhir bab Anda akan mampu memetakan setiap konsep satu kerangka kerja ke dua lainnya; membaca *benchmark* publik secara kritis dan mengetahui apa yang dikatakan dan tidak dikatakannya tentang aplikasi Anda; mengukur sendiri sebuah kerangka kerja dan memeriksa angkanya dengan hukum antrean yang sederhana; membandingkan produktivitas, ekosistem, dan bawaan sekuriti berdasarkan bukti; membaca kalender dukungan .NET, Java, Spring Boot, PHP, dan Laravel; serta memilih kerangka kerja untuk proyek nyata dengan alasan yang dapat Anda tuliskan.

15.1 Titik Berangkat Kita

Keponakan Bu Wati, Dimas, tahun ini lulus diploma informatika. Ia membantu di laundry selama liburan dan melihat ketiga versi program konter berjalan di komputer yang sama. Kini keluarga seorang temannya, yang menjalankan usaha katering kecil dengan dua belas karyawan, memintanya membangun sistem pesanan, dan ia harus memilih satu kerangka kerja. Setiap teman sekelasnya punya pendapat. Yang satu berkata ASP.NET Core paling cepat, jadi pasti yang terbaik. Yang lain berkata Laravel paling mudah di-*hosting*, karena setiap paket *hosting* murah menyediakan PHP. Yang ketiga berkata Spring Boot dipakai bank, jadi pasti yang paling serius. Ketiga pernyataan itu mengandung sebagian kebenaran, dan tidak satu pun merupakan alasan.

Bab ini memberi Dimas, dan Anda, cara untuk memutuskan. Mula-mula ketiga kerangka kerja dijajarkan konsep demi konsep (Subbab 15.3), karena kejutan terbesar setelah membangun aplikasi yang sama tiga kali adalah betapa *miripnya* mereka. Lalu kita melihat bukti yang biasa dikutip orang: *benchmark* kinerja (Subbab 15.4), ukuran produktivitas (Subbab 15.5), survei tentang apa yang dipakai pengembang (Subbab 15.6), dan kalender yang menyatakan berapa lama setiap versi menerima perbaikan (Subbab 15.7). Bawaan sekuriti mendapat perbandingannya sendiri (Subbab 15.8). Subbab 15.9 mengubah semuanya menjadi prosedur pengambilan keputusan.

15.2 Gagasannya dalam Bahasa Sehari-hari

Memilih kerangka kerja berarti memilih sesuatu yang akan dijalani tim selama bertahun-tahun. Karena itu pertanyaan yang berguna bukan “mana yang terbaik?”, melainkan “mana yang terbaik *untuk proyek ini, tim ini, dan tempat ini*?” **Tabel 15.1** mendaftar pertanyaan yang dijawab bab ini dan jenis bukti yang diperlukan masing-masing.

Tabel 15.1 Apa yang dibandingkan, dan jenis bukti yang diperlukan setiap pertanyaan.

Pertanyaan	Jenis bukti	Di mana dalam bab ini
Apakah cukup cepat?	*benchmark*, dan pengukuran sendiri atas beban kerja sendiri	Subbab 15.4
Seberapa besar kerja sebuah fitur?	ukuran kode fitur yang sama, perkakas, kurva belajar	Subbab 15.5
Bisakah kita merekrut dan mendapat bantuan?	survei pengembang, *registry* paket, pasar kerja dan *hosting* setempat	Subbab 15.6
Berapa lama sebuah versi didukung?	kalender rilis dan dukungan resmi	Subbab 15.7
Apakah aman secara bawaan?	bawaan untuk kata sandi, XSS, CSRF, SQL	Subbab 15.8
Cocokkah dengan tim dan server kita?	apa yang dikuasai tim, di mana aplikasi harus berjalan	Subbab 15.9

Setiap jenis bukti punya jebakannya sendiri. *Benchmark* mengukur satu beban kerja di satu mesin; survei mengukur siapa yang menjawabnya; kalender dapat berubah; pengukuran Anda sendiri hanya sebaik metodenya. Aturan yang mengalir sepanjang bab ini sederhana: *sebutkan dari mana setiap angka berasal, kapan diukur, dan apa yang diukurnya.*

15.3 Satu Aplikasi, Tiga Kosakata

15.3.1 Konsep yang sama

Bab 2 memperkenalkan gagasan yang dimiliki bersama semua kerangka kerja web modern: *routing*, controller, model dan ORM, injeksi dependensi, templat. Setelah membangun laundry tiga kali, kita dapat mengisi tabelnya dengan nama-nama sebenarnya (**Tabel 15.2**).

Tabel 15.2 Konsep yang sama dengan tiga nama.

Konsep	ASP.NET Core 10	Laravel 13	Spring Boot 4.1
Bahasa, *runtime*	C# 14, .NET 10	PHP 8.5	Java 25, JVM
Proyek baru	`dotnet new`	`laravel new`, `composer create-project`	Spring Initializr
Pembantu baris perintah	CLI `dotnet`	`php artisan`	`mvn` atau `gradle`
Routing	atribut `[HttpGet("...")]` atau `MapGet`	`routes/web.php`, `routes/api.php`	`@GetMapping` pada metode
Controller	kelas turunan `Controller`	kelas di `app/Http/Controllers`	`@Controller`, `@RestController`
Validasi masukan	*data annotation*, `[ApiController]`	kelas Form Request, `validate()`	Jakarta Bean Validation, `@Validated`
Gaya ORM	EF Core, *unit of work* (`DbContext`)	Eloquent, *active record*	JPA/Hibernate, repository
Perubahan skema	migrasi EF Core	migrasi Laravel	Flyway atau Liquibase
Injeksi dependensi	bawaan, *constructor injection*	*service container*, *constructor injection*	kontainer Spring, *constructor injection*
Templat	Razor (`.cshtml`)	Blade (`.blade.php`)	Thymeleaf (`.html`)
Keluaran JSON	`System.Text.Json`, *record*	API Resource	Jackson 3, *record*
Galat dalam satu bentuk JSON	`IExceptionHandler`, `ModelState`	badan 422, 404, dan 401 bawaan kerangka kerja	`@RestControllerAdvice`
Login dan peran	Identity atau *authentication handler*, `[Authorize(Roles=...)]`	*guard*, Gate, *middleware* `can`	rantai filter Spring Security, `@PreAuthorize`
Pengujian	xUnit, `WebApplicationFactory`	PHPUnit atau Pest, `RefreshDatabase`	JUnit 6, `@WebMvcTest`, `@DataJpaTest`

Sebagian besar baris hanya berbeda ejaan. Dua baris menunjukkan perbedaan desain yang sesungguhnya, dan keduanya patut dipahami.

***Active record* versus *unit of work*.** Dalam Eloquent milik Laravel, sebuah objek tahu cara menyimpan dirinya: `$order->update([...])` langsung menulis ke basis data. Dalam EF Core dan JPA, objeknya polos; objek terpisah (`DbContext`, `EntityManager` JPA) melacak objek-objek yang dimuat dalam satu *unit of work* dan menulis semua perubahannya ketika Anda memanggil `SaveChanges` atau ketika transaksi di-*commit* (*dirty checking* di Subbab 13.5). *Active record* lebih cepat untuk memulai dan terbaca wajar dalam aplikasi kecil; *unit of work* memudahkan menyatukan beberapa perubahan dan menguji aturan tanpa basis data. Keduanya mencapai hasil yang sama di toko kita, tetapi Anda harus tahu mana yang sedang dipakai: di JPA, lupa bahwa sebuah entitas *managed* berarti perubahan yang tidak disengaja bisa tersimpan; di Eloquent, lupa bahwa `update` langsung menyimpan berarti separuh perubahan bisa tersimpan sebelum galat.

Seberapa banyak yang diatur lewat konvensi. Laravel menemukan tabel `orders` untuk model `Order` dan kolom-kolomnya berdasarkan nama, dan rute tinggal di satu berkas yang dapat Anda baca dari atas ke bawah. Spring Boot mengonfigurasi apa pun yang ditemukannya di *classpath* (tambahkan *starter* Security, dan setiap halaman memerlukan login) dan menemukan controller dengan memindai anotasi. ASP.NET Core menyisakan lebih banyak di `Program.cs`, yang Anda

baca dari atas ke bawah dan yang menyatakan berurutan apa saja yang dipakai aplikasi. Tidak ada yang lebih baik; masing-masing menyembunyikan hal yang berbeda, dan minggu pertama dengan kerangka kerja baru sebagian besar dihabiskan untuk mempelajari apa yang dikerjakannya tanpa diminta.

15.3.2 Riwayat satu permintaan

Kemiripannya lebih dalam daripada nama. Ikuti `POST /api/orders` dari Bab 16 melewati setiap kerangka kerja, dan perhentian yang sama muncul dalam urutan yang sama.

1. **Server web menerima bita-bitanya.** ASP.NET Core punya server sendiri, Kestrel, di dalam aplikasi. Spring Boot menanamkan Tomcat. Laravel berjalan di dalam PHP, yang dijalankan server web (nginx dengan PHP-FPM, Apache, atau `php artisan serve` selama pengembangan) untuk setiap permintaan.
2. **Serangkaian *middleware* berjalan.** Di ASP.NET Core rangkaian itu adalah urutan pemanggilan `app.Use...` di `Program.cs`: penanganan eksepsi, autentikasi, pemeriksaan akses, lalu *endpoint*. Di Laravel, *middleware* global dan *middleware* rute (`auth:sanctum`, `can:`) yang dideklarasikan di `bootstrap/app.php` dan berkas rute. Di Spring Boot, filter-filter *servlet*, di antaranya rantai filter Spring Security dari Bab 14. Di ketiganya, sebuah *middleware* dapat menjawab sendiri (401, atau *redirect* ke halaman login) sehingga permintaan tidak pernah sampai ke controller.
3. ***Routing* memilih satu penangan** berdasarkan metode dan jalur: atribut pada metode C#, satu baris di `routes/api.php`, `@PostMapping` pada metode Java.
4. **Masukan diikat dan divalidasi.** Badan JSON menjadi *record* `OrderRequest` di C# dan Java, atau larik yang sudah divalidasi dalam Form Request Laravel. Masukan yang tidak sah berhenti di sini dengan 422 dan semua galatnya sekaligus: Form Request Laravel melakukannya sendiri, sedangkan ASP.NET Core dan Spring Boot diajari berbuat sama di Bab 12 dan 16.
5. **Kode kita berjalan:** metode controller yang tipis memanggil service, service menerapkan aturan (`LaundryRules` di C# dan Java, `PriceCalculator` dan `OrderWorkflow` di PHP) di dalam transaksi, dan repository atau model berbicara dengan basis data melalui ORM.
6. **Hasilnya diserialkan** menjadi JSON (`System.Text.Json`, API Resource, Jackson) atau dirender oleh templat, dan eksepsi di sepanjang jalan diubah menjadi badan galat kontrak oleh satu penangan pusat.

Begitu Anda dapat menyebut keenam perhentian ini, mempelajari kerangka kerja baru sebagian besar tinggal soal mencari tahu apa nama setiap perhentian di sana dan di mana konfigurasinya. Ini juga memberi tahu ke mana harus melihat ketika ada yang salah: 401 berasal dari perhentian 2, 404 untuk URL yang Anda yakin ada berasal dari perhentian 3, 422 dari perhentian 4, 409 dari perhentian 5.

15.3.3 Membandingkan pengujian

Piramida pengujian Bab 14 ada di ketiga kerangka kerja, dengan nama berbeda untuk tingkat yang sama. Uji unit biasa untuk aturan sama sekali tidak memerlukan kerangka kerja: JUnit di Java, xUnit atau NUnit di C#, PHPUnit atau Pest di PHP. Pengujian lapisan web mengirim permintaan tanpa membuka port jaringan: `MockMvc` dengan `@WebMvcTest` di Spring Boot, `WebApplicationFactory` dengan server uji di memori pada ASP.NET Core, dan di Laravel pembantu `getJson` dan `postJson` dari *test case*, yang memanggil aplikasi secara langsung. Pengujian basis data memakai basis data sekali pakai: `@DataJpaTest` melakukan *rollback* sesudah setiap pengujian, *trait* `RefreshDatabase` di Laravel memigrasi basis data yang bersih, dan pengujian EF Core biasanya memakai SQLite di memori. Pembantu pengujian Laravel paling

ringkas (satu pengujian API utuh muat dalam beberapa baris, seperti `CapstoneTest` di Bab 16); *slice* Spring Boot paling tepat mengenai apa yang dinyalakan; `WebApplicationFactory` ASP.NET Core menyalakan aplikasi sungguhan dengan service yang diganti, di antara keduanya. Tidak satu pun dari ketiganya membuat pengujian sulit, dan di tidak satu pun pengujian bersifat pilihan.

15.3.4 Endpoint yang sama tiga kali

Untuk membandingkan kode dan kecepatan secara adil, kita memerlukan satu pekerjaan kecil yang ditulis dengan cara yang sama di ketiganya. Bab ini memakai penawaran harga: `GET /api/quote?price=8000&unit=kg&quantity=2.5&pickup=true` mengembalikan total Persamaan (15.1), aturan Bab 8 dan 11 yang dinyatakan ulang di Subbab 13.4.3, dengan b perseratus yang ditagih (paling sedikit 300 untuk layanan per kilogram, satuan utuh untuk yang lain):

$$T = \left\lfloor \frac{p \cdot b + 50}{100} \right\rfloor + \begin{cases} 10000 & \text{antar-jemput dan subtotal} < 100000 \\ 0 & \text{selainnya} \end{cases} \tag{15.1}$$

Kode 15.1 adalah versi ASP.NET Core, sebuah *minimal API*: tanpa kelas controller, hanya sebuah rute dan sebuah fungsi.

Kode 15.1 `PerfLab/Program.cs`: *endpoint* penawaran harga sebagai *minimal API* ASP.NET Core.

```
// Endpoint kecil yang sama di ketiga kerangka kerja: harga satu pesanan.
// GET /api/quote?price=8000&unit=kg&quantity=2.5&pickup=true
//   -> {"price":8000,"unit":"kg","quantity":2.5,"pickup":true,"total":34000}
using System.Text.Json.Serialization;

var builder = WebApplication.CreateSlimBuilder(args);
builder.Logging.ClearProviders();                // ukur pekerjaannya, bukan log
var app = builder.Build();

app.MapGet("/api/quote",
    (long price, string unit, decimal quantity, bool pickup) =>
{
    // Aturan Bab 8 dan 11, dalam perseratus satuan.
    long h = decimal.ToInt64(quantity * 100m);
    long billed = unit == "kg" ? Math.Max(h, 300) : (h + 99) / 100 * 100;
    long subtotal = (price * billed + 50) / 100;
    long total = subtotal + (pickup && subtotal < 100_000 ? 10_000 : 0);
    return Results.Json(new Quote(price, unit, quantity, pickup, total),
                        QuoteJson.Default.Quote);
});

app.Run();

public record Quote(long Price, string Unit, decimal Quantity, bool Pickup,
                    long Total);

// Kode JSON untuk Quote dibangkitkan saat kompilasi (tanpa refleksi).
[JsonSerializable(typeof(Quote))]
[JsonSourceGenerationOptions(
    PropertyNamingPolicy = JsonKnownNamingPolicy.CamelCase)]
public partial class QuoteJson : JsonSerializerContext
{
}
```

`CreateSlimBuilder` menyalakan ASP.NET Core dengan lebih sedikit fitur bawaan, dan kelas `QuoteJson` memakai pembangkit kode sumber `System.Text.Json`, yang menulis kode JSON saat kompilasi alih-alih menelusuri properti *record* saat program berjalan. Keduanya adalah optimasi

yang dianjurkan tim .NET untuk layanan kecil yang cepat. **Kode 15.2** adalah *endpoint* yang sama di Laravel, memakai ulang kelas `PriceCalculator` dari Bab 8, dan **Kode 15.3** di Spring Boot.

Kode 15.2 `routes/api.php`: *endpoint* penawaran harga sebagai rute Laravel.

```php
<?php

use App\Domain\PriceCalculator;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

// Endpoint kecil yang sama dengan PerfLab (ASP.NET Core) dan QuoteController.
Route::get('/quote', function (Request $request) {
    $price = (int) $request->query('price');
    $unit = (string) $request->query('unit');
    $quantity = (string) $request->query('quantity');
    $pickup = $request->boolean('pickup');
    return response()->json([
        'price' => $price, 'unit' => $unit, 'quantity' => (float) $quantity,
        'pickup' => $pickup,
        'total' => PriceCalculator::total($unit === 'kg', $price, $quantity,
            $pickup),
    ]);
});
```

Kode 15.3 `QuoteController.java`: *endpoint* penawaran harga sebagai controller Spring Boot.

```java
package com.profitina.perflab;

import java.math.BigDecimal;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

/** Endpoint kecil yang sama dengan PerfLab (ASP.NET Core) dan routes/api.php. */
@RestController
public class QuoteController {

    public record Quote(long price, String unit, BigDecimal quantity,
                        boolean pickup, long total) { }

    @GetMapping("/api/quote")
    public Quote quote(@RequestParam long price, @RequestParam String unit,
                       @RequestParam BigDecimal quantity,
                       @RequestParam(defaultValue = "false") boolean pickup) {
        // Aturan Bab 8 dan 11, dalam perseratus satuan.
        long h = quantity.movePointRight(2).longValueExact();
        long billed = "kg".equals(unit) ? Math.max(h, 300)
                : (h + 99) / 100 * 100;
        long subtotal = (price * billed + 50) / 100;
        long total = subtotal + (pickup && subtotal < 100_000 ? 10_000 : 0);
        return new Quote(price, unit, quantity, pickup, total);
    }
}
```

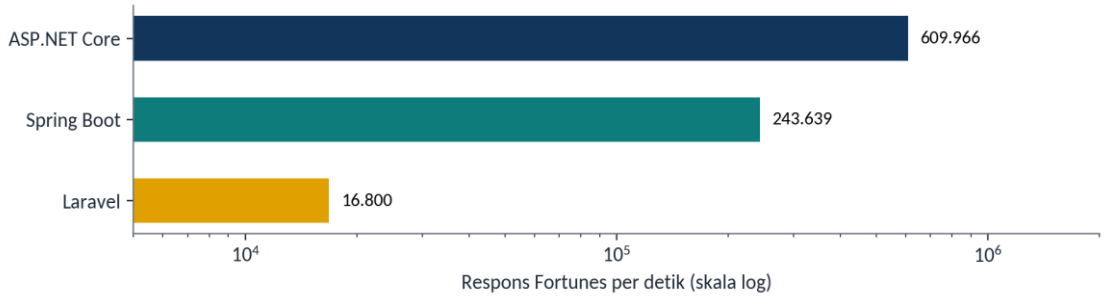
Baca ketiganya berdampingan dan kemiripannya mencolok: satu rute, tiga parameter, satu aturan, satu objek JSON. Perbedaannya lebih terletak pada bahasa daripada pada kerangka kerjanya: jumlah tiba sebagai `decimal` di C#, `BigDecimal` di Java, atau string di PHP, lalu ketiganya menghitung dalam perseratus utuh dengan bilangan bulat, sehingga tidak ada pembulatan *floating-point* yang dapat membuat salah satunya berselisih satu rupiah.

15.4 Kinerja

15.4.1 Apa yang diukur benchmark publik

Perbandingan publik kerangka kerja web yang paling terkenal adalah TechEmpower Framework Benchmarks. Selama lebih dari satu dasawarsa proyek itu menjalankan enam uji kecil yang sama (JSON polos, satu kueri basis data, banyak kueri, pembaruan, teks polos, dan "Fortunes") terhadap ratusan implementasi kerangka kerja di perangkat keras yang identik dan menerbitkan hasilnya sebagai *round*. Round 23, terbit pada Februari 2025, berjalan di server baru dengan prosesor Intel Xeon Gold 6330 56 inti dan jaringan 40 Gbit/s [1]. Ternyata itulah *round* terakhir: pada 24 Maret 2026 TechEmpower mengumumkan penghentian proyek dan mengarsipkan repositorinya, sambil mempertahankan hasil-hasil lama sebagai potret [2]. Karena itu Round 23 masih merupakan yang "terbaru" dan akan tetap demikian, hal yang patut diingat ketika seseorang mengutip angka yang terdengar lebih baru.

Uji *Fortunes* paling mendekati halaman sungguhan: server membaca semua baris sebuah tabel kecil melalui ORM kerangka kerja, menambahkan satu baris di memori, mengurutkannya, lalu merender tabel HTML melalui mesin templat kerangka kerja dengan *escaping* yang benar [1]. **Gambar 15.1** memperlihatkan *throughput* Fortunes Round 23 untuk ketiga kerangka kerja, sebagaimana dihimpun Pham dari hasil resmi [3].



Gambar 15.1 TechEmpower Round 23 (Februari 2025), uji Fortunes: respons per detik dari implementasi ASP.NET Core, Spring, dan Laravel terbaik, dalam skala logaritmik.

Perbedaannya besar: ASP.NET Core menjawab sekitar 2,5 kali lebih banyak permintaan per detik daripada Spring, dan Spring sekitar 14,5 kali lebih banyak daripada Laravel. Sebelum menyimpulkan apa pun, bacalah apa sebenarnya angka-angka itu.

- **Itu entri terbaik setiap keluarga, disetel oleh para ahli.** Entri ASP.NET Core tercepat memakai bagian tumpukan yang paling ramping dengan akses basis data mentah; entri Laravel berjalan pada susunan PHP produksi. Aplikasi Anda yang ditulis seperti di buku ini akan lebih lambat di ketiganya.
- **Perangkat kerasnya jauh lebih besar daripada server sebuah laundry.** Dengan 609.966 respons per detik, ASP.NET Core dapat melayani semua pesanan Profitina Laundry dalam setahun dalam waktu jauh di bawah satu detik.
- **Sumbu logaritmik menyembunyikan seberapa banyak itu cukup.** 16.800 respons per detik milik Laravel tetap sekitar satu juta halaman per menit. Sebuah toko, sekolah, atau klinik paling-paling mengirim beberapa permintaan per detik di saat tersibuknya.

> **WAWASAN KUNCI:** *Benchmark* menjawab "seberapa cepat kerangka kerja ini *bisa*?", bukan "seberapa cepat aplikasi *saya* nanti?" Untuk kebanyakan aplikasi bisnis, ketiga kerangka kerja jauh lebih cepat daripada yang diperlukan, dan yang menentukan kecepatan adalah basis data, jaringan, dan kode yang Anda tulis.

Kinerja tetap penting dalam dua keadaan: ketika aplikasi benar-benar melayani lalu lintas berat (situs *e-commerce* nasional, *payment gateway*), di mana kerangka kerja yang lebih cepat berarti lebih sedikit server, dan ketika anggarannya hanya satu server kecil, di mana jejak memori yang lebih kecil membuat lebih banyak aplikasi muat di dalamnya.

15.4.2 Mengukur sendiri

Alih-alih mempercayai angka orang lain, ukurlah beban kerja Anda sendiri di mesin Anda sendiri. **Kode 15.4** adalah pembangkit beban kecil: sejumlah *pekerja* masing-masing mengirim permintaan berturut-turut selama waktu tertentu, lalu program melaporkan permintaan per detik dan latensi yang tidak dilampaui 50% dan 99% permintaan (*p50* dan *p99*).

Kode 15.4 `LoadGen/Program.cs`: pembangkit beban yang melaporkan *throughput* dan persentil latensi.

```
// Pembangkit beban kecil: N pekerja mengirim permintaan GET selama waktu
// tertentu, lalu kita laporkan permintaan per detik dan persentil latensi.
// dotnet run -c Release -- <url> <detik> <pekerja>
using System.Diagnostics;

string url = args.Length > 0 ? args[0]
    : "http://localhost:4150/api/quote?price=8000&unit=kg&quantity=2.5&pickup=true";
int seconds = args.Length > 1 ? int.Parse(args[1]) : 10;
int workers = args.Length > 2 ? int.Parse(args[2]) : 16;

using var http = new HttpClient(new SocketsHttpHandler
{
    MaxConnectionsPerServer = workers,
    PooledConnectionLifetime = TimeSpan.FromMinutes(5)
});

// Pemanasan: biarkan JIT mengompilasi jalur sibuk di kedua sisi dahulu.
for (int i = 0; i < 2000; i++) (await http.GetAsync(url)).Dispose();

var latencies = new List<double>[workers];
long errors = 0;
var stop = DateTime.UtcNow.AddSeconds(seconds);
var clock = Stopwatch.StartNew();
await Task.WhenAll(Enumerable.Range(0, workers).Select(async w =>
{
    var mine = latencies[w] = new List<double>(200_000);
    while (DateTime.UtcNow < stop)
    {
        long start = Stopwatch.GetTimestamp();
        using var res = await http.GetAsync(url);
        await res.Content.ReadAsByteArrayAsync();
        if (!res.IsSuccessStatusCode) Interlocked.Increment(ref errors);
        mine.Add(Stopwatch.GetElapsedTime(start).TotalMilliseconds);
    }
}));
double elapsed = clock.Elapsed.TotalSeconds;

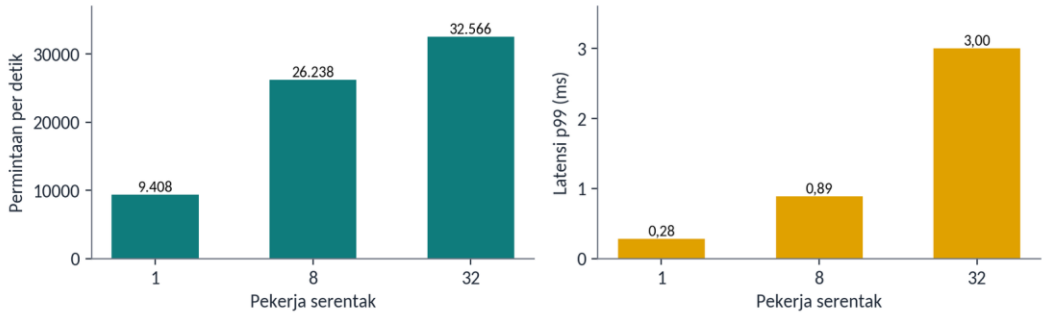
var all = latencies.SelectMany(l => l).Order().ToArray();
double P(double q) => all[(int)Math.Min(all.Length - 1, q * all.Length)];
Console.WriteLine($"workers {workers,3}: {all.Length / elapsed,9:F0} req/s   "
    + $"p50 {P(0.50):F2} ms   p99 {P(0.99):F2} ms   errors {errors}");
```

Dua rincian membuat pengukurannya adil. Pemanasan mengirim 2.000 permintaan sebelum mengukur, karena .NET, JVM, dan *opcode cache* PHP semuanya menjalankan kode lebih lambat pada beberapa kali pertama selagi mengompilasi dan menyimpannya. Selain itu, setiap pekerja mencatat setiap latensi, sehingga persentil dihitung dari semua permintaan, tidak dirata-ratakan.

Mesin tempat buku ini ditulis hanya dapat menjalankan versi ASP.NET Core (yang lain memerlukan paket yang tidak dapat diunduhnya). PerfLab dibangun dalam mode Release lalu dinyalakan; ia menjawab permintaan pertamanya 446 ms setelah dijalankan, dan penawarannya benar: `{"price":8000,"unit":"kg","quantity":2.5,"pickup":true,"total":34000}`, minimum 3 kg ditambah biaya antar-jemput. Kemudian LoadGen berjalan masing-masing 10 detik dengan 1, 8, dan 32 pekerja, di mesin virtual dua inti yang sama:

```
workers   1:      9408 req/s   p50 0.09 ms   p99 0.28 ms   errors 0
workers   8:     26238 req/s   p50 0.28 ms   p99 0.89 ms   errors 0
workers  32:     32566 req/s   p50 0.84 ms   p99 3.00 ms   errors 0
```

Gambar 15.2 menggambar angka yang sama.



Gambar 15.2 Pengukuran kami sendiri atas *endpoint* penawaran ASP.NET Core di mesin virtual 2 vCPU (25 September 2026): *throughput* mendatar setelah 8 pekerja sementara latensi p99 terus naik.

Bentuk kurvanya adalah pelajaran yang paling berguna. Dari 1 ke 8 pekerja, *throughput* hampir tiga kali lipat (9.408 menjadi 26.238 permintaan per detik), karena satu pekerja membiarkan prosesor menganggur selagi menunggu setiap jawaban. Dari 8 ke 32 pekerja, empat kali lebih banyak, *throughput* hanya naik seperempat, menjadi 32.566: kedua inti hampir jenuh, dibagi antara server dan pembangkit beban itu sendiri. Melewati titik itu, tambahan pekerja terutama membuat setiap permintaan menunggu lebih lama dalam antrean: latensi p99 naik dari 0,89 ms menjadi 3,00 ms, lebih dari tiga kali.

15.4.3 Memeriksa angka dengan hukum Little

Teori antrean memberi pemeriksaan yang dapat dikerjakan siapa pun dengan tangan. *Hukum Little* menyatakan bahwa dalam sistem yang stabil, rata-rata banyaknya permintaan di dalam sistem, L, sama dengan laju kedatangannya, λ, dikali rata-rata waktu yang dihabiskan setiap permintaan di dalam sistem, W:

$$L = \lambda W \tag{15.2}$$

Dalam uji beban tertutup setiap pekerja selalu punya tepat satu permintaan yang sedang berjalan, sehingga L adalah banyaknya pekerja dan λ adalah *throughput* terukur; Persamaan (15.2) lalu memberi rata-rata latensi $W = L/\lambda$. Untuk satu pekerja, $W = 1/9408$ s ≈ 0.106 ms; untuk 8 pekerja, $W = 8/26238$ s ≈ 0.305 ms; untuk 32 pekerja, $W = 32/32566$ s ≈ 0.983 ms. Setiap rata-rata berada sedikit di atas median terukur (0,09, 0,28, dan 0,84 ms) dan jauh di bawah p99, persis seperti sebaran latensi yang berekor panjang. Seandainya Persamaan (15.2) memberi rata-rata *di bawah* median, atau jauh di atas p99, pengukurannya pasti keliru di suatu tempat, misalnya ada pekerja yang berhenti lebih awal. Satu baris perhitungan telah mengonfirmasi *benchmark* itu.

Hukum yang sama menjawab pertanyaan perencanaan. Bila usaha katering itu mengharapkan paling banyak 5 permintaan per detik pada siang hari dan masing-masing memerlukan sekitar 20

ms termasuk basis data, maka rata-rata $L = 5 \times 0.02 = 0.1$ permintaan sedang dilayani setiap saat: satu server kecil dengan kerangka kerja mana pun sebagian besar waktunya menganggur.

15.4.4 Waktu nyala dan memori

Dua angka lain penting untuk pemasangan kecil. *Waktu nyala* menentukan seberapa cepat aplikasi hidup kembali setelah pembaruan, atau seberapa cepat salinan baru menyala ketika lalu lintas naik: PerfLab menjawab setelah sekitar setengah detik; aplikasi Spring Boot dengan JPA biasanya memerlukan beberapa detik di JVM (jauh lebih singkat bila dikompilasi lebih dahulu dengan GraalVM, yang didukung Spring Boot); aplikasi PHP sama sekali tidak punya waktu nyala, karena setiap permintaan mulai dari awal di server yang sudah berjalan. *Memori* menentukan berapa aplikasi yang muat di satu server: satu permintaan PHP memakai beberapa megabita lalu melepaskannya, sedangkan *runtime* .NET dan JVM mempertahankan proses yang hangat berukuran puluhan sampai ratusan megabita. Ukurlah keduanya dengan cara yang sama seperti *throughput*, di mesin Anda, dengan aplikasi Anda; README kode bab ini menunjukkan caranya.

Cara mengukur memori tidak rumit. Di Windows, buka Task Manager, tab *Details*, dan lihat kolom *Memory* untuk proses `dotnet`, `java`, atau `php-cgi` setelah aplikasi menerima beberapa ratus permintaan. Di macOS pakai Activity Monitor, dan di Linux perintah `ps -o rss,cmd -C dotnet` (atau `java`, `php-fpm`) menampilkan memori residen dalam kilobita. Catat angkanya dua kali: sesaat setelah aplikasi menyala, dan setelah uji beban selesai. Selisihnya memberi gambaran berapa yang dipakai aplikasi ketika sibuk, dan angka kedua itulah yang menentukan berapa banyak aplikasi muat di satu VPS kecil.

15.5 Produktivitas

Produktivitas lebih sulit diukur daripada kecepatan, dan klaim tentangnya layak diperlakukan sama hati-hatinya. Buku ini menyediakan tiga ukuran yang jujur.

Ukuran kode untuk fitur yang sama. Menghitung aplikasi utuh tidak banyak artinya, karena setiap versi *capstone* membawa tambahan yang berbeda (dua *store* di ASP.NET Core, halaman Bab 13 di Spring Boot). Ukuran yang lebih adil adalah besar satu *fitur yang ditambahkan ke ketiganya dengan cara yang sama*. Bab 16 menambahkan dua: menghapus pelanggan, hanya oleh pemilik dan tidak pernah bila pelanggan itu punya pesanan, dan papan publik yang menampilkan antrean dengan nama depan. **Tabel 15.3** menghitung baris yang diperlukan masing-masing, tanpa baris kosong dan komentar.

Tabel 15.3 Baris yang diperlukan untuk dua fitur yang sama pada *capstone* Bab 16, dihitung pada 25 September 2026.

Fitur	ASP.NET Core 10	Laravel 13	Spring Boot 4.1
Hapus pelanggan khusus pemilik	sekitar 20: service 10, controller 6, kueri *store* 4	sekitar 12: controller 8, rute 3, *gate* 1	sekitar 17: service 9, controller 5, aturan sekuriti 2, repository 1
Papan publik, controller dan tampilan	18 + 42	17 + 37, ditambah 1 rute	21 + 35, ditambah 1 aturan sekuriti

Papan memakan biaya yang kurang lebih sama di mana-mana, karena sebagian besarnya HTML. Fitur hapus memperlihatkan watak tiap kerangka kerja: *route-model binding* Laravel, *gate* satu baris, dan `abort(409, ...)` menjadikannya yang terpendek; Spring Boot memakai barisnya untuk dua pemeriksaan yang berdiri sendiri (aturan URL dan `@PreAuthorize`) dan untuk metode repository yang tidak diperlukan oleh `$customer->orders()->exists()` milik Eloquent; ASP.NET

Core menulis kueri *store*-nya sendiri karena *capstone*-nya mendukung dua *store*. Baris kode adalah ukuran kasar (satu baris PHP dan satu baris Java tidak memuat makna yang sama), tetapi hasilnya mendukung apa yang tersirat dari membangun toko tiga kali: untuk aplikasi semacam ini tidak ada kerangka kerja yang jauh lebih ringkas, dan perbedaannya lebih berasal dari pilihan desain daripada dari bahasanya.

Perkakas yang menuliskan kode untuk Anda. `php artisan make:model Order -mcr` membuat model, migrasi, dan *resource controller* dalam satu perintah; `dotnet new`, `dotnet ef migrations add`, dan Spring Initializr mengerjakan hal serupa. Ketiganya didukung sangat baik di VS Code, dan Java serta C# punya bantuan tambahan berupa kompilator yang memeriksa tipe sebelum program berjalan, yang menangkap banyak kesalahan yang di PHP baru dilaporkan ketika barisnya dijalankan (PHPStan dan Larastan menutup sebagian besar celah itu bagi Laravel).

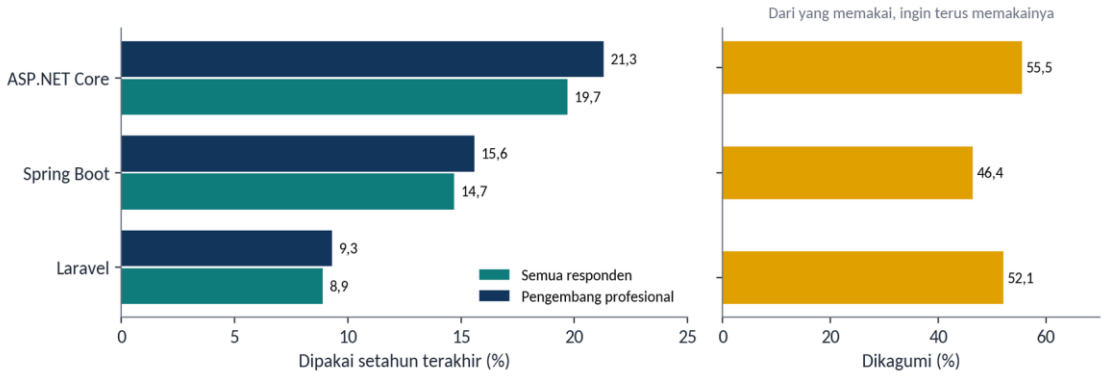
Kurva belajar. Laravel biasanya paling cepat dimulai bagi orang yang mengenal PHP dari pelajaran pemrograman web, karena *routing*, model, dan templatnya terlihat dan jumlahnya sedikit. ASP.NET Core dan Spring Boot menuntut lebih banyak di awal (bahasa yang dikompilasi, injeksi dependensi di mana-mana, lebih banyak konfigurasi untuk dipahami), dan membalasnya pada basis kode yang lebih besar dengan tipe dan perkakas yang lebih kuat. Ini sesuai dengan pengalaman Bab 3 sampai 14: bab Laravel pertama paling cepat menghasilkan halaman yang berjalan, dan bab-bab Spring Boot paling banyak memerlukan penjelasan tentang apa yang terjadi tanpa diminta.

Bahasanya sendiri. Ketiga bahasa saling mendekat dalam beberapa tahun terakhir, dan *capstone* memakai fitur modern yang sama di ketiganya. *Record* menggambarkan data dalam satu baris: `record OrderRequest(...)` di C#, `record CreateOrderRequest(...)` di Java; PHP punya kelas *readonly* dan *constructor promotion* untuk tujuan yang sama. Enumerasi bernilai menggantikan string ajaib: `enum OrderStatus` di ketiganya, dan *backed enum* PHP (`case Queued = 'queued'`) bahkan mengubah ke dan dari teks yang disimpan dengan sendirinya. *Pattern matching* menggantikan rentetan `if`: status berikutnya adalah ekspresi `switch` di `LaundryRules.Advance` C# dan `LaundryRules.advance` Java, dan `match` di metode `next()` enum PHP yang dipanggil `OrderWorkflow::advance`; C# memperingatkan dan Java menolak mengompilasi bila ada kasus yang terlewat, sedangkan PHP melempar `UnhandledMatchError` ketika barisnya dijalankan. Nilai *null* dinyatakan eksplisit di C# (`string?`), dideklarasikan dengan `?string` di PHP, dan ditangani dengan `Optional` serta anotasi di Java. Perbedaan yang tersisa menyangkut *kapan* kesalahan ditemukan: C# dan Java memeriksa tipe saat program dikompilasi, PHP saat barisnya dijalankan (kecuali dipakai penganalisis statis seperti PHPStan). Bagi tim yang suka kesalahannya ditangkap kompilator, perbedaan itu lebih penting daripada *benchmark* mana pun.

15.6 Ekosistem dan Komunitas

15.6.1 Apa yang dipakai pengembang

Survei pengembang publik terbesar adalah milik Stack Overflow. Edisi 2025 menanyakan kerangka kerja dan teknologi web mana yang dipakai responden secara intensif dalam setahun terakhir; 23.678 responden menjawab pertanyaan itu, 19.460 di antaranya pengembang profesional [4]. **Gambar 15.3** memperlihatkan ketiga kerangka kerja buku ini.



Gambar 15.3 Stack Overflow Developer Survey 2025: persentase responden yang memakai setiap kerangka kerja dalam setahun terakhir, dan persentase penggunanya yang ingin terus memakainya ("dikagumi").

ASP.NET Core dipakai 19,7% dari semua responden dan 21,3% dari pengembang profesional, Spring Boot 14,7% dan 15,6%, Laravel 8,9% dan 9,3% [4]. Sebagai pembanding, Node.js dipakai 48,7% dan Express 19,9%, sehingga ketiga kerangka kerja kita adalah pilihan arus utama yang besar. Angka "dikagumi" menanyakan kepada pengguna sebuah teknologi apakah mereka ingin melanjutkannya: 55,5% untuk ASP.NET Core, 52,1% untuk Laravel, dan 46,4% untuk Spring Boot [4].

Bacalah survei seperti membaca *benchmark*: siapa yang menjawab? Responden Stack Overflow adalah pengunjung situs berbahasa Inggris yang memilih sendiri untuk ikut, dan keikutsertaan survei sangat berbeda menurut negara dan jenis pemberi kerja. Survei itu tidak memberi tahu Anda apa yang diminta pasar kerja di Surabaya atau Makassar, juga tidak memberi tahu berapa banyak usaha kecil Indonesia yang menjalankan PHP di *shared hosting*. Untuk pertanyaan-pertanyaan itu, carilah bukti setempat: lowongan kerja di kota Anda, paket *hosting* yang dijual penyedia lokal, dan apa yang benar-benar dipakai perusahaan tempat Anda ingin bekerja.

15.6.2 Paket, hosting, dan bantuan

Setiap kerangka kerja hidup dalam ekosistem paket yang besar: NuGet untuk .NET, Packagist (melalui Composer) untuk PHP, Maven Central untuk Java. Untuk aplikasi toko, ketiganya menyediakan apa yang diperlukan: *driver* basis data, nota PDF, kode QR, ekspor Excel, klien *payment gateway*. Dua perbedaan praktis tersisa. PHP berjalan di hampir setiap paket *hosting*, termasuk *shared hosting* termurah, sedangkan ASP.NET Core dan Spring Boot biasanya memerlukan server virtual atau platform kontainer, yang sedikit lebih mahal dan menuntut sedikit lebih banyak pengetahuan. Selain itu, literatur tanya-jawab dan tutorial besar untuk ketiganya, tetapi untuk Laravel banyak yang ditulis bagi pemula, sedangkan untuk Spring Boot dan ASP.NET Core banyak yang mengandaikan sistem yang lebih besar.

Bagi pembaca di Indonesia, pertanyaan *hosting* sering menjadi penentu yang diam-diam. Paket *shared hosting* dengan panel kontrol yang dijual banyak penyedia lokal hampir selalu menyediakan PHP dan MySQL, kadang juga Composer dan akses SSH terbatas, tetapi jarang mengizinkan proses yang terus berjalan seperti Kestrel atau Tomcat. Artinya aplikasi Laravel dapat dipasang di paket seperti itu, sedangkan aplikasi ASP.NET Core atau Spring Boot memerlukan server virtual (VPS), baik dari penyedia lokal maupun penyedia awan global, dengan sistem operasi yang Anda kelola sendiri: memasang *runtime*, menyiapkan layanan agar menyala ulang otomatis, dan memasang *reverse proxy* dengan HTTPS. Pekerjaan itu tidak sulit, dan Bab 6 sudah menjalaninya untuk ASP.NET Core, tetapi harus ada orang yang mengerjakannya dan

memeliharanya. Sebelum memutuskan, tanyakan kepada klien di mana situs mereka sekarang berjalan, siapa yang memeliharanya, dan berapa anggaran bulanannya.

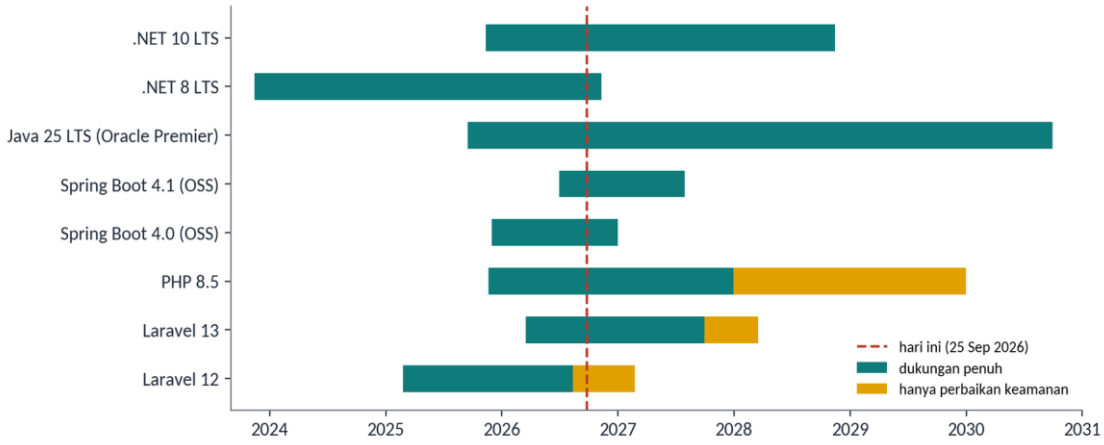
Pertanyaan tenaga kerja pun sebaiknya dijawab dengan bukti setempat. Cara sederhana yang dapat dilakukan siapa saja: buka dua atau tiga situs lowongan kerja yang populer di kota Anda, cari setiap nama kerangka kerja, dan catat banyaknya lowongan beserta jenis perusahaannya pada hari yang sama. Angka yang Anda kumpulkan sendiri, dengan tanggal dan sumbernya, lebih berguna untuk keputusan Anda daripada angka global mana pun, asalkan Anda ingat bahwa angka itu hanya potret satu hari.

15.7 Kalender Dukungan

Versi kerangka kerja yang tidak lagi menerima perbaikan keamanan tidak boleh berhadapan dengan internet. Karena itu setiap tim perlu tahu, sebelum mulai, berapa lama versi yang dipilih akan didukung dan seberapa sering harus meningkatkan versi. **Tabel 15.4** mengumpulkan tanggal-tanggalnya per 25 September 2026, dan **Gambar 15.4** menggambarnya.

Tabel 15.4 Tanggal rilis dan dukungan, dihimpun dari endoflife.date pada 25 September 2026.

Versi	Dirilis	Dukungan penuh sampai	Perbaikan keamanan sampai
.NET 10 (LTS)	11 Nov 2025	14 Nov 2028	14 Nov 2028
.NET 8 (LTS)	14 Nov 2023	10 Nov 2026	10 Nov 2026
Java 25 (LTS, Oracle Premier)	16 Sep 2025	30 Sep 2030	30 Sep 2030
Spring Boot 4.1 (sumber terbuka)	30 Jun 2026	31 Jul 2027	31 Jul 2027
Spring Boot 4.0 (sumber terbuka)	30 Nov 2025	31 Des 2026	31 Des 2026
PHP 8.5	20 Nov 2025	31 Des 2027	31 Des 2029
Laravel 13	17 Mar 2026	30 Sep 2027	17 Mar 2028
Laravel 12	24 Feb 2025	13 Agu 2026	24 Feb 2027



Gambar 15.4 Jendela dukungan versi-versi yang dipakai buku ini; garis putus-putus adalah 25 September 2026.

Pola di balik tanggal-tanggal itu lebih berguna daripada tanggalnya sendiri.

- **.NET** merilis versi mayor baru setiap November; versi bernomor genap adalah *Long Term Support* (LTS) dengan tiga tahun dukungan [5]. .NET 10 didukung sampai November 2028; .NET 8, yang dipakai banyak proyek yang sudah ada, berakhir pada 10 November 2026, sehingga proyek yang memakainya harus meningkatkan versi sekarang.

- **Java** merilis setiap enam bulan dan menetapkan versi LTS setiap dua tahun; Java 25 (September 2025) adalah LTS yang berlaku [6]. Dukungan JDK panjang, tetapi aplikasi Spring juga bergantung pada kalender Spring sendiri.
- **Spring Boot** merilis versi minor setiap enam bulan (Mei atau Juni dan November), dan masing-masing menerima perbaikan sumber terbuka selama sekitar tiga belas bulan [7], [8]. Dukungan komersial memperpanjangnya bagi pelanggan yang membayar. Dalam praktik, aplikasi Spring Boot sebaiknya pindah ke versi minor berikutnya paling tidak setahun sekali: dukungan sumber terbuka 4.0 berakhir pada 31 Desember 2026, sehingga kode kita memakai 4.1.
- **PHP** memberi setiap versi dua tahun dukungan aktif disusul dua tahun perbaikan keamanan [9].
- **Laravel** merilis satu versi mayor setahun, sekitar kuartal pertama, dan menyediakan perbaikan *bug* selama 18 bulan dan perbaikan keamanan selama dua tahun [10]. Laravel 13 memerlukan PHP 8.3 atau lebih baru dan mendukung sampai PHP 8.5.

Pegangan praktisnya: tim .NET meningkatkan versi setiap dua atau tiga tahun, tim Laravel setiap tahun, tim Spring Boot setiap enam sampai dua belas bulan, tetapi peningkatan minor Spring Boot biasanya kecil. Rencanakan pekerjaan peningkatan versi sejak awal proyek, apa pun pilihan Anda.

15.8 Aman secara Bawaan

Ketiga kerangka kerja membuat hal yang aman menjadi hal yang mudah, dan mereka sepakat pada sebagian besar bawaan (**Tabel 15.5**).

Tabel 15.5 Bawaan sekuriti dibandingkan.

Urusan	ASP.NET Core	Laravel	Spring Boot
Injeksi SQL	EF Core memparameterkan setiap nilai	Eloquent dan *query builder* mengikat setiap nilai	JPA dan Spring Data mengikat setiap nilai
XSS di templat	Razor mengodekan `@value`	Blade `{{ }}` meng-*escape*	Thymeleaf `th:text` meng-*escape*
CSRF pada formulir	token *antiforgery*, ditambahkan *tag helper*	token `@csrf`, diperiksa *middleware*	token ditambahkan Thymeleaf, diperiksa filter sekuriti
Hash kata sandi (bawaan)	Identity: PBKDF2 dengan HMAC-SHA512, 100.000 iterasi	bcrypt, *cost* 12	*encoder* delegasi: bcrypt, *cost* 10
Galat validasi	dokumen *problem* 400 otomatis dengan `[ApiController]`; *capstone* mematikannya dan menjawab 422	JSON 422 dengan semua galat, secara bawaan	penangan milik aplikasi; 422 sejak Bab 12

Bawaan kata sandi berasal dari kode dan dokumentasi kerangka kerjanya sendiri: `PasswordHasher` ASP.NET Core Identity (format versi 3) memakai PBKDF2 dengan HMAC-SHA512, garam 128 bit, dan 100.000 iterasi [11]; konfigurasi *hashing* Laravel memakai bcrypt dengan 12 putaran kecuali `BCRYPT_ROUNDS` menyatakan lain; `BCryptPasswordEncoder` Spring Security memakai kekuatan 10 bila dibuat melalui *encoder* delegasi (Subbab 14.3.1 memperlihatkan arti satu langkah *cost*). Tempat-tempat berbahayanya pun sama di ketiganya: SQL mentah yang dirangkai dengan penggabungan string (`DB::raw`, `FromSqlRaw`, kueri *native*), templat yang mencetak HTML tanpa *escape* (`@Html.Raw`, `{!! !!}`, `th:utext`), dan perlindungan CSRF yang dimatikan secara menyeluruh. Mengenal pintu-pintu darurat kerangka kerja Anda adalah bagian dari mengenal kerangka kerja itu.

15.9 Memilih

Bukti-bukti tadi kini muat dalam prosedur singkat. Prosedur ini ditulis untuk sistem katering Dimas, tetapi berlaku untuk proyek apa pun.

1. **Tuliskan batasan yang tidak dapat digeser.** Di mana aplikasi harus berjalan (*shared hosting* PHP, server Windows perusahaan, platform kontainer di awan)? Haruskah ia tersambung ke sistem yang sudah ada (platform Java seperusahaan, login Active Directory)? Sebuah batasan sering memutuskan sendiri: klien dengan jajaran Windows Server dan SQL Server condong ke ASP.NET Core; klien yang sudah membayar *hosting* PHP condong ke Laravel; bank yang sistem lainnya berbasis Spring condong ke Spring Boot.
2. **Hitung apa yang sudah dikuasai tim.** Kerangka kerja yang dikuasai tim dengan baik lebih cepat dan lebih aman daripada yang "lebih baik" tetapi masih dipelajari. Dengan satu pengembang, pilih apa yang paling dikuasai pengembang itu, kecuali ada batasan yang melarangnya.
3. **Periksa bebannya.** Perkirakan menit tersibuk dengan hukum Little (Persamaan (15.2)). Bila ketiganya cukup cepat dengan selisih lebar (keadaan yang lazim), kecepatan gugur dari pertimbangan. Bila tidak, ukur beban kerja Anda sendiri seperti di Subbab 15.4.2.
4. **Periksa kalendernya.** Pilih versi yang masih punya dukungan paling tidak setahun, dan rencanakan peningkatannya.
5. **Bangun satu fitur nyata sebagai percobaan (*spike*).** Dua hari membangun formulir pesanan dengan calon terkuat mengungkap lebih banyak daripada dua minggu membaca.
6. **Tuliskan keputusannya** beserta alasannya, agar orang berikutnya memahaminya dan tahu kapan harus meninjaunya kembali.

Tabel 15.6 menerapkan prosedur ini pada empat keadaan yang realistis.

Tabel 15.6 Empat keadaan dan kerangka kerja yang disarankan prosedur ini.

Keadaan	Pilihan yang mungkin	Alasan utama
Usaha katering, satu pengembang yang menguasai PHP, *hosting* murah	Laravel	penguasaan tim dan *hosting*
Grup rumah sakit dengan server Windows dan SQL Server	ASP.NET Core	platform dan staf yang sudah ada
Bank koperasi yang menambah layanan pada sistem berbasis Spring	Spring Boot	cocok dengan platform dan aturan sekuritinya
Perusahaan rintisan yang memperkirakan lalu lintas API berat, tim menguasai C# dan Java	ASP.NET Core atau Spring Boot, setelah pengukuran	kinerja per server, tipe yang kuat

Bagi Dimas prosedurnya singkat. Usaha katering itu punya satu pengembang (dirinya), yang kini sudah membangun aplikasi yang sama di ketiga kerangka kerja dan paling menguasai PHP dari diplomanya; keluarga itu sudah menyewa *hosting* PHP untuk situs webnya; menit tersibuknya adalah pesanan makan siang lewat telepon, beberapa permintaan per detik. Jadi Laravel, dengan catatan untuk meninjau ulang keputusan itu bila suatu hari usaha itu memerlukan aplikasi ponsel dengan lalu lintas berat.

Sebagai latihan kedua, terapkan prosedur yang sama pada keadaan lain yang akrab: sebuah pesantren ingin mencatat pembayaran SPP dan uang makan santri, dengan satu staf tata usaha yang pernah belajar Java di SMK, sebuah komputer server tua di kantor yayasan, dan internet yang kadang putus. Batasannya adalah server di tempat dan sambungan yang tidak andal, jadi aplikasi harus berjalan di jaringan lokal tanpa bergantung pada layanan awan. Penguasaan tim mengarah ke Java. Bebannya sangat kecil. Kalendernya memihak Java 25 LTS dengan Spring Boot

4.1 yang diperbarui setiap tahun. Maka Spring Boot adalah pilihan yang wajar, dengan catatan bahwa seseorang harus dijadwalkan memperbarui versinya setiap tahun; bila itu tidak mungkin, ASP.NET Core dengan .NET LTS yang didukung tiga tahun menjadi pesaing serius karena pemeliharaannya lebih jarang. Perhatikan bahwa keputusan ini sama sekali tidak menyentuh *benchmark*.

15.10 Memakainya Sungguhan

Dua kebiasaan dari bab ini layak dipertahankan apa pun pilihan Anda. Pertama, simpan aturan usaha dalam kode biasa yang tidak bergantung pada kerangka kerja, seperti `LaundryRules` di C# dan Java serta `PriceCalculator` dengan `OrderWorkflow` di PHP. Aturan adalah bagian yang harus ditulis ulang bila pindah dari satu kerangka kerja ke yang lain; bila aturan itu kecil, murni, dan teruji, perpindahan sebagian besar tinggal pekerjaan rutin pada lapisan luar. Kedua, gambarkan API Anda sebagai kontrak yang tidak bergantung pada kerangka kerja: URL, kolom, kode status, badan galat. Bab 16 menunjukkan bahwa satu kontrak dapat diimplementasikan tiga kali dan diperiksa oleh satu program; kontrak yang sama memungkinkan aplikasi ponsel atau sistem mitra bekerja dengan aplikasi Anda apa pun bahan pembuatnya, dan memungkinkan Anda suatu hari mengganti kerangka kerja di belakangnya tanpa ada yang menyadari.

Keputusan hanya berguna bila dapat ditemukan dan dipahami kemudian. Banyak tim menyimpan *architecture decision record* singkat untuk setiap pilihan penting, satu halaman di repositori proyek dengan empat judul: konteks (apa yang harus dikerjakan proyek, untuk siapa, di mana berjalannya), pilihan yang dipertimbangkan, keputusan, dan akibatnya, termasuk apa yang akan membuat tim meninjau ulang. Untuk sistem katering, catatan Dimas akan berbunyi: konteks, usaha dua belas orang yang menerima pesanan lewat telepon, satu pengembang, *hosting* PHP sewaan; pilihan, ketiga kerangka kerja buku ini; keputusan, Laravel 13 di atas PHP 8.5; akibat, peningkatan versi tahunan di kuartal pertama, dan peninjauan bila suatu hari direncanakan aplikasi ponsel dengan lalu lintas berat. Menuliskannya memakan dua puluh menit, dan menyelamatkan pengembang berikutnya dari mengulang seluruh diskusi, atau dari membatalkan keputusan yang baik tanpa tahu mengapa keputusan itu diambil.

15.11 Profitina dalam Praktik

Profitina sendiri tidak dibangun dengan satu pun dari ketiga kerangka kerja buku ini: ia aplikasi Python dengan FastAPI, dilayani Uvicorn, dengan PostgreSQL di produksi. Alasannya mengikuti prosedur Subbab 15.9. Pekerjaan intinya melibatkan menjalankan model AI secara lokal dan mengolah teks, bidang tempat pustaka Python paling kuat; timnya menguasai Python dengan baik; dan FastAPI memberi model permintaan dan jawaban yang bertipe, validasi otomatis, dan galat bergaya *problem*, gagasan yang sama dengan ketiga kerangka kerja di sini. Keputusannya dituliskan beserta untung-ruginya, dan bagian-bagian yang tidak boleh bergantung pada kerangka kerja mana pun (aturan skor, aturan langganan) berupa fungsi biasa dengan pengujiannya sendiri. Memilih dengan baik tidak terlalu menyangkut nama kerangka kerjanya, melainkan kecocokannya dengan tim, pekerjaan, dan tempatnya.

15.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Memilih berdasarkan grafik *benchmark*.** Hasil peringkat teratas adalah entri terbaik sebuah kerangka kerja di perangkat keras besar. Ukur beban kerja Anda sendiri, dan periksa dulu apakah kecepatan penting sama sekali (Persamaan (15.2)).
- **Mengutip angka tanpa sumber dan tanggal.** "Laravel dipakai 9%" tidak bermakna tanpa "dari responden Stack Overflow Developer Survey 2025". Selalu sebutkan di mana dan kapan.
- **Mengukur aplikasi yang masih dingin.** Permintaan pertama ke .NET, JVM, dan PHP lambat selagi kode dikompilasi dan disimpan. Panaskan dulu sebelum mengukur, dan ukur dalam mode Release atau produksi.
- **Menjalankan pembangkit beban di mesin yang jenuh tanpa menyebutkannya.** Pengukuran kami membagi dua inti antara server dan klien; itu wajar untuk menunjukkan metodenya, tidak untuk dibandingkan dengan *benchmark* pusat data.
- **Mengabaikan kalender dukungan.** Memulai proyek pada versi yang kehilangan dukungan tiga bulan lagi berarti peningkatan versi di siklus rilis pertama.
- **Menganggap perbedaan kerangka kerja sebagai risiko utama.** Penguasaan tim, *hosting*, dan mutu kode lebih penting daripada kerangka kerja bagi hampir setiap aplikasi bisnis.
- **Membandingkan versi yang tidak setara.** Membandingkan Laravel terbaru dengan Spring Boot lama (atau sebaliknya) mengukur umur perangkat lunaknya, bukan kerangka kerjanya. Samakan dulu generasinya, lalu catat versi persis di setiap angka yang Anda laporkan.
- **Melupakan biaya pemeliharaan.** Kerangka kerja yang hemat server tetapi tidak dapat dipelihara siapa pun di tim justru mahal. Hitung juga jam kerja manusia untuk memasang, memperbarui, dan memperbaiki.
- **Berganti kerangka kerja untuk memperbaiki masalah desain.** Controller yang gemuk dan aturan yang tersebar tetap gemuk dan tersebar di kerangka kerja baru. Perbaiki pelapisannya dulu.

15.13 Ringkasan Bab

- Setelah membangun aplikasi yang sama tiga kali, ternyata kerangka kerja-kerangka kerja itu terpetakan konsep demi konsep (**Tabel 15.2**); perbedaan desain yang sesungguhnya adalah *active record* versus *unit of work*, dan seberapa banyak yang diatur lewat konvensi.
- TechEmpower Round 23 (Februari 2025) adalah *round* pamungkas *benchmark* itu, yang diarsipkan pada Maret 2026; hasil Fortunes-nya mengurutkan ASP.NET Core, Spring, dan Laravel dalam urutan itu (**Gambar 15.1**), tetapi yang ditunjukkannya adalah entri terbaik yang disetel di perangkat keras sangat besar.
- Pengukuran kami sendiri atas *endpoint* ASP.NET Core memperlihatkan *throughput* yang mendatar di batas mesin sementara latensi naik (**Gambar 15.2**); hukum Little (Persamaan (15.2)) memeriksanya dengan tangan.
- Fitur yang sama memerlukan jumlah baris yang mirip di ketiga kerangka kerja; Laravel paling pendek untuk fitur kecil berkat konvensinya (**Tabel 15.3**).
- Dalam Stack Overflow Developer Survey 2025, ASP.NET Core dipakai 19,7% responden, Spring Boot 14,7%, dan Laravel 8,9%; ketiganya dikagumi oleh sekitar separuh penggunanya (**Gambar 15.3**).
- Kalender dukungan berbeda: .NET LTS selama tiga tahun, Laravel dengan versi mayor tahunan dan dua tahun perbaikan keamanan, Spring Boot dengan versi minor setiap enam bulan dan sekitar tiga belas bulan dukungan sumber terbuka (**Tabel 15.4**, **Gambar 15.4**).

- Bawaan sekuriti kuat dan mirip di ketiganya (**Tabel 15.5**); risikonya ada di pintu-pintu darurat.
- Kerangka kerja dipilih berdasarkan batasan, penguasaan tim, beban, kalender, dan percobaan, lalu keputusannya dituliskan (**Tabel 15.6**).

15.14 Soal Latihan

1. Seorang teman berkata: "Di Round 23 Laravel menangani 16.800 permintaan per detik pada Fortunes, jadi situs Laravel saya di akun *shared hosting* dapat melayani 16.800 pengunjung per detik." Berikan tiga alasan mengapa kesimpulan itu tidak sah.
2. Sebuah uji beban dengan 16 pekerja melaporkan 12.000 permintaan per detik dan latensi p50 1,9 ms. Pakai hukum Little untuk menghitung rata-rata latensinya. Masuk akalkah laporan itu? Bagaimana bila laporannya menyebut p50 3,5 ms?
3. Usaha katering memperkirakan 600 pesanan antara pukul 11.00 dan 13.00, setiap pesanan menimbulkan sekitar 8 permintaan masing-masing 40 ms. Berapa rata-rata permintaan yang sedang berjalan? Apakah ini memengaruhi pilihan kerangka kerja?
4. Petakan bagian-bagian Laravel dari Bab 16 berikut ke padanannya di ASP.NET Core dan Spring Boot: (a) `StoreCustomerRequest`; (b) `OrderResource`; (c) `Gate::define('delete-customers', ...)`; (d) berkas-berkas migrasi; (e) pembungkus `{"data": ...}` yang ditambahkan `JsonResource`.
5. Dengan **Tabel 15.4**, sebuah tim memulai proyek pada 1 Oktober 2026 dan berencana merilisnya pada Maret 2027 serta memeliharanya sampai akhir 2028 tanpa pekerjaan besar. Untuk setiap kerangka kerja, versi mana yang sebaiknya dipakai di awal, dan peningkatan versi mana yang harus direncanakan?
6. Ukur *endpoint* PerfLab di komputer Anda sendiri dengan 1, 4, dan 16 pekerja, dan periksa setiap hasil dengan hukum Little. Lalu ubah PerfLab agar memakai `WebApplication.CreateBuilder` dan penyerial JSON berbasis refleksi, dan ukur lagi. Apa yang berubah, dan mengapa perubahannya mungkin kecil?

15.15 Penyelesaian Soal Latihan

1. (a) Angka Round 23 adalah entri Laravel terbaik, disetel para ahli, bukan aplikasi biasa dengan autentikasi, pencatatan log, dan lebih banyak kueri. (b) Uji itu berjalan di server 56 inti dengan jaringan 40 Gbit/s; akun *shared hosting* hanya mendapat sebagian kecil dari satu server. (c) Permintaan bukan pengunjung: satu tampilan halaman memuat halamannya dan sering beberapa sumber daya lain, dan pengunjung tidak datang semuanya pada detik yang sama. Alasan keempat: Fortunes membaca satu tabel mungil; pekerjaan basis data halaman sungguhan biasanya jauh lebih mahal.

2. $W = L/\lambda = 16/12000$ s ≈ 1.33 ms. Rata-rata 1,33 ms dengan median 1,9 ms *tidak* masuk akal: dengan latensi berekor panjang rata-rata biasanya di atas median, sedangkan di sini jauh di bawahnya. Kemungkinan besar uji itu mengukur *throughput* dan latensi pada rentang waktu berbeda, atau sebagian pekerja menganggur. Dengan p50 3,5 ms rata-ratanya akan jauh di bawah median, lebih tidak masuk akal lagi. Hukum itu tidak membuktikan laporan benar, tetapi cepat menunjukkan bila laporan itu salah.

3. 600 pesanan × 8 permintaan = 4.800 permintaan dalam 7.200 detik, jadi $\lambda \approx 0.67$ permintaan per detik; $L = \lambda W = 0.67 \times 0.04 \approx 0.027$. Rata-rata satu permintaan sedang berjalan kurang dari 3% waktu. Kerangka kerja mana pun di server terkecil sudah lebih dari cukup; kinerja tidak memengaruhi pilihan.

4. (a) `CustomerRequest`, *record* dengan *data annotation* di `Api/Dtos.cs`, di ASP.NET Core; `CreateCustomerRequest`, *record* dengan batasan Bean Validation, di Spring Boot. (b) `OrderView` di keduanya (*record* di `Dtos.cs`, dan *record* dengan metode pabrik di paket `api` Spring). (c) `[Authorize(Roles = "OWNER")]` pada aksi hapus di ASP.NET Core; `.requestMatchers(HttpMethod.DELETE, "/api/customers/*").hasRole("OWNER")` dan `@PreAuthorize("hasRole('OWNER')")` di Spring Boot. (d) Model EF Core di `LaundryDbContext` (dengan `EnsureCreated`, atau migrasi EF Core yang dibangkitkan darinya) di ASP.NET Core; entitas dengan `ddl-auto=update`, atau skrip Flyway, di Spring Boot. (e) *Record* generik `DataView<T>` di ASP.NET Core dan *record* `DataView` di Spring Boot, yang dikembalikan controller di sekeliling setiap record tunggal.

5. .NET: mulai dengan .NET 10 LTS, didukung sampai November 2028, yang mencakup seluruh periode; tidak perlu peningkatan mayor, hanya rilis *patch* bulanan. Spring Boot: mulai dengan 4.1 (4.0 berakhir Desember 2026), lalu rencanakan pindah ke 4.2 (diperkirakan sekitar November 2026) atau yang lebih baru sebelum Juli 2027, dan ke setiap versi minor berikutnya kira-kira setiap enam bulan; pertahankan pula Java 25 LTS. Laravel: mulai dengan Laravel 13 (perbaikan *bug* sampai 30 September 2027, keamanan sampai 17 Maret 2028), rencanakan peningkatan ke Laravel 14 pada 2027 dan mungkin ke 15 sebelum akhir 2028; PHP 8.5 didukung sampai akhir 2029.

6. Angkanya bergantung pada komputer Anda; periksa masing-masing dengan $W = L/\lambda$ terhadap p50 dan p99 yang dilaporkan seperti di Subbab 15.4.3. *Builder* standar menambahkan lebih banyak *middleware* dan pencatatan log, dan JSON berbasis refleksi lebih lambat daripada JSON hasil pembangkit kode, sehingga *throughput* biasanya turun. Perubahannya mungkin kecil karena di komputer kecil pembangkit beban berebut prosesor dengan server, dan karena pekerjaan per permintaan (satu perkalian dan satu objek JSON mungil) begitu kecil sehingga biaya tetap penanganan HTTP yang mendominasi.

Daftar Pustaka

[1] TechEmpower, “Framework Benchmarks Round 23,” blog post and results, Feb.–Mar. 2025. Accessed: Sep. 25, 2026. [Online]. Available: https://www.techempower.com/blog/2025/03/17/framework-benchmarks-round-23/

[2] TechEmpower, “Sunsetting the TechEmpower Framework Benchmarks,” GitHub issue #10932, Mar. 24, 2026. [Online]. Available: https://github.com/TechEmpower/FrameworkBenchmarks/issues/10932

[3] T. A. Pham, “Best popular backend frameworks by performance of throughput benchmark comparison and ranking in 2025,” DEV Community, 2025 (compiled from TechEmpower Round 23 Fortunes results). Accessed: Sep. 25, 2026. [Online]. Available: https://dev.to/tuananhpham/popular-backend-frameworks-performance-benchmark-1bkh

[4] Stack Overflow, “2025 Stack Overflow Developer Survey: Technology, Web frameworks and technologies,” 2025. Accessed: Sep. 25, 2026. [Online]. Available: https://survey.stackoverflow.co/2025/technology

[5] endoflife.date, “Microsoft .NET,” release and support dates. Accessed: Sep. 25, 2026. [Online]. Available: https://endoflife.date/dotnet

[6] endoflife.date, “Oracle JDK,” release and support dates. Accessed: Sep. 25, 2026. [Online]. Available: https://endoflife.date/oracle-jdk

[7] endoflife.date, “Spring Boot,” release and support dates. Accessed: Sep. 25, 2026. [Online]. Available: https://endoflife.date/spring-boot

[8] Broadcom, “Spring Support Policy.” Accessed: Sep. 25, 2026. [Online]. Available: https://spring.io/support-policy/

[9] endoflife.date, “PHP,” release and support dates. Accessed: Sep. 25, 2026. [Online]. Available: https://endoflife.date/php

[10] Laravel, “Release Notes: Versioning Scheme and Support Policy,” Laravel 13.x documentation; and endoflife.date, “Laravel.” Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/releases and https://endoflife.date/laravel

[11] .NET Foundation, “PasswordHasher.cs,” ASP.NET Core source code, `src/Identity/Extensions.Core/src`. Accessed: Sep. 25, 2026. [Online]. Available: https://github.com/dotnet/aspnetcore/blob/main/src/Identity/Extensions.Core/src/PasswordHasher.cs

Bab 16

Evaluasi dan Pembahasan IV

Bab ini menyatukan seluruh buku. Profitina Laundry selesai di ketiga kerangka kerja: empat entitas yang sama (`Customer`, `Service`, `Order`, `Payment`), aturan yang sama, API JSON yang sama dengan login dan peran, dan papan konter publik yang sama. Seperti bab-bab evaluasi sebelumnya, bab ini disusun sebagai soal, masing-masing dikerjakan dalam urutan yang sama: pahami soalnya, kerjakan satu contoh dengan tangan, rancang di atas kertas, bangun penyelesaiannya, dan pelajari satu penyelesaian keliru yang menggoda. Di akhir bab Anda akan mampu menyatakan kontrak API cukup tepat sehingga sebuah program dapat memeriksanya; memverifikasi tiga implementasi yang berdiri sendiri terhadap satu program itu; mengenali tempat-tempat kerangka kerja berbeda dalam bawaannya (nama field, jawaban validasi, penolakan) dan menyesuaikan masing-masing dengan kontrak; memperluas ketiganya dengan fitur yang sama dan membandingkan biayanya; serta menilai dengan jujur apa yang dijalankan, apa yang hanya dikompilasi, dan apa yang masih harus dikerjakan.

16.1 Titik Berangkat Kita

Bab 3 sampai 6 membangun laundry dengan ASP.NET Core, Bab 7 sampai 10 dengan Laravel, dan Bab 11 sampai 14 dengan Spring Boot. Bab 12 menuliskan API yang dipakai aplikasi ponsel Raka sebagai kontrak berisi enam belas pemeriksaan, dan membuat versi Spring lulus dengan cara yang sama seperti versi Laravel. Bab 14 menambahkan login dengan token, peran pemilik, dan tes. Bab 15 membandingkan ketiga kerangka kerja dari luar.

Bu Wati kini punya satu permintaan terakhir, dan satu keputusan. Permintaannya adalah sebuah layar: televisi di samping konter setiap outlet yang menampilkan pesanan mana yang masih antre dan mana yang siap diambil, agar pelanggan berhenti bertanya “punya saya sudah jadi?”. Keputusannya adalah versi mana dari ketiganya yang akan dijalankan outlet-outlet. Raka tidak peduli yang mana, asalkan aplikasinya tetap berjalan. Cara paling bersih untuk menepati janji itu adalah membuat ketiga versi menaati kontrak yang sama, dan membuktikannya dengan satu program.

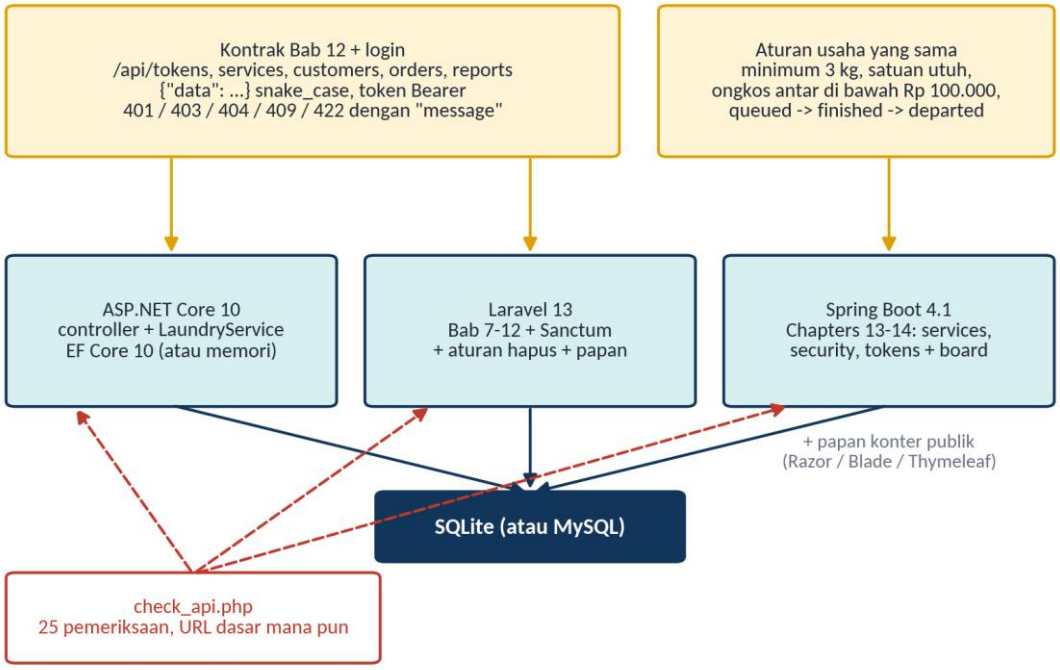
Karena itu bab ini punya lima soal.

- **Soal 16.1**, kontrak dengan login, peran, dan logout, ditulis sebagai program yang memeriksa server mana pun.
- **Soal 16.2**, aturan toko, dikerjakan dengan tangan dan diperiksa dalam tiga bahasa.
- **Soal 16.3**, *capstone* ASP.NET Core, dibangun dari nol di atas kontrak dan dijalankan sungguhan.
- **Soal 16.4**, *capstone* Laravel, melengkapi proyek Bab 10.
- **Soal 16.5**, *capstone* Spring Boot, melengkapi proyek Bab 14.

Kodenya ada di `Chapter16`: `contract` untuk pemeriksa, `aspnet`, `laravel`, dan `spring` untuk ketiga aplikasi, dan `runs` untuk keluaran dari run yang dilakukan saat buku ini ditulis.

16.2 Cara Mendekati Capstone

Capstone adalah tempat banyak keputusan kecil bertemu, dan urutan kerja lebih penting daripada di bab mana pun. **Gambar 16.1** memperlihatkan urutan yang dipakai di sini: kontrak lebih dulu, lalu aturan, lalu aplikasi, dan pemeriksa yang sama di akhir masing-masing.



Gambar 16.1 Satu kontrak, tiga implementasi: pemeriksa `check_api.php` mengirim 25 permintaan yang sama ke setiap server dan membandingkan jawabannya dengan kontrak.

Tiga kebiasaan menopang seluruh bab.

Tulis kontrak sebagai program. Kontrak dalam kalimat ("pesanan yang salah ditolak") dapat dibaca dengan beberapa cara; pemeriksaan ("`POST /api/orders` dengan jumlah 0 menjawab 422, dan `errors` punya kunci `customer_id`, `service_id`, `quantity`, dan `delivery_address`") tidak. Program juga berjalan dalam sedetik dan dapat diulang setelah setiap perubahan.

Jaga aturan bebas dari kerangka kerja. Aturan harga dan alur kerja adalah fungsi kecil yang murni di setiap bahasa (`LaundryRules` di C# dan Java, `PriceCalculator` dan `OrderWorkflow` di PHP). Aturan diperiksa tersendiri, tanpa server web, sebelum aplikasi mana pun memakainya.

Katakan apa yang dijalankan. Mesin tempat buku ini ditulis dapat membangun dan menjalankan aplikasi .NET yang hanya memerlukan *shared framework*, dan punya PHP serta JDK, tetapi tidak dapat mengunduh paket dari NuGet, Composer, atau Maven Central. Semua yang dapat dijalankan telah dijalankan dan keluaran aslinya ditampilkan; semua yang lain diperiksa sejauh yang diizinkan mesin itu, dan setiap subbab menyatakan dengan tepat sejauh mana.

Agar pernyataan-pernyataan itu dapat dibandingkan, bab ini memakai empat tingkat bukti, dari yang paling lemah sampai yang paling kuat. *Diperiksa sintaksnya*: berkas dapat diurai oleh bahasanya (`php -l`, pengurai Java), jadi tidak ada salah ketik yang menghentikan program, tetapi belum ada yang diketahui tentang nama dan tipe. *Dikompilasi*: semua nama, tipe, dan *import* cocok satu sama lain, di sini sebagian terhadap pengganti kecil untuk pustaka yang tidak dapat diunduh. *Dijalankan*: program benar-benar berjalan dan keluarannya dapat dibaca, seperti pemeriksa aturan dan *capstone* ASP.NET Core. *Diperiksa terhadap kontrak*: server yang berjalan

menjawab ke-25 permintaan pemeriksa dengan benar. Hanya tingkat terakhir yang membuktikan bahwa aplikasi Raka akan bekerja; tingkat-tingkat di bawahnya adalah langkah menuju ke sana. Setiap kali bab ini menyebut sebuah hasil, hasil itu ditempatkan pada salah satu tingkat ini, dan Subbab 16.8 merangkumnya untuk ketiga kerangka kerja. Kebiasaan ini berguna jauh di luar buku: laporan "sudah selesai" dari seorang pengembang selalu layak disertai pertanyaan "sampai tingkat mana?"

16.3 Soal 16.1: Kontrak

16.3.1 Soalnya

Tulis program yang memeriksa apakah sebuah server menaati kontrak API Profitina Laundry: enam belas pemeriksaan Bab 12 sebagai kasir yang sudah login, ditambah pintu-pintu Bab 14 (kata sandi yang salah, token yang tidak ada, hak kasir dan hak pemilik, serta logout). Program harus bekerja terhadap server mana pun dari ketiganya, hanya memakai PHP biasa, dan meninggalkan data yang tidak merusak run berikutnya.

16.3.2 Memahaminya dengan tangan

Sebelum menulis program, telusuri satu sesi aplikasi Raka di atas kertas. Kasir login dan menerima token. Ia mendaftarkan pelanggan baru, Sri Rahayu, dengan mengetik telepon "0812-5566-7788"; server menyimpan `081255667788`. Ia menerima sebuah pesanan: 2,5 kg cuci-kering-lipat seharga Rp 8.000, dijemput dari Jl. Kenanga 5. Aturan Bab 8 memberi

$$T = \left\lfloor \frac{8000 \cdot \max(250{,}300) + 50}{100} \right\rfloor + 10000 = 24000 + 10000 = 34000 \tag{16.1}$$

karena minimum 3 kg berlaku dan subtotalnya di bawah Rp 100.000, sehingga biaya antar-jemput dikenakan. Pesanan berstatus `queued` dengan sisa tagihan Rp 34.000. Meminta `departed` sekarang harus gagal, karena satu langkah terlompati; `finished` berhasil dan mencatat `finished_at`; `departed` gagal lagi, karena belum ada yang dibayar. Pembayaran Rp 40.000 ditolak ("Only Rp 34.000 is still owed."), pembayaran Rp 34.000 lewat QRIS diterima, dan kini `departed` berhasil. Pembayaran berikutnya mana pun ditolak dengan 409, karena pesanan sudah keluar toko. Bila pembayaran pada sebuah pesanan berjumlah P, sisa tagihannya adalah

$$B = T - P, \qquad 0 \le B \le T \tag{16.2}$$

dan setiap pemeriksaan dalam sesi ini adalah pernyataan tentang T, B, atau status. Kasir lalu mencoba menghapus data Sri dan ditolak dengan 403; pemilik login dan ditolak dengan 409, karena Sri punya pesanan; pelanggan tanpa pesanan dapat ia hapus. Terakhir kasir logout, dan tokennya berhenti berlaku.

Sesi itu memuat setiap pemeriksaan. Setelah dituliskan, sesi itu juga menjadi urutan program menjalankannya, karena setiap langkah memerlukan apa yang dibuat langkah sebelumnya.

16.3.3 Kontraknya

Tabel 16.1 merangkum kontraknya. Rinciannya, field demi field, adalah milik Bab 12; Bab 14 menjelaskan alasan di baliknya.

Tabel 16.1 Kontrak API Profitina Laundry setelah Bab 16.

Bagian	Aturan
Login	`POST /api/tokens` dengan `email`, `password`, `device`: 201 `{"token", "user": {"name", "role"}}`; kata sandi yang salah menjadi 422 pada `email`
Pintu	tanpa token: 401 `{"message": "Unauthenticated."}`; peran yang tidak boleh: 403 `{"message": "This action is unauthorized."}`
Data	field JSON dalam snake_case; satu record sebagai `{"data": {...}}`; satu halaman sebagai `{"data": [...], "meta": {"current_page", "per_page", "total", "last_page"}}`
Nilai	status dan metode pembayaran dalam huruf kecil; uang dalam rupiah utuh; waktu sebagai `YYYY-MM-DDTHH:MM:SS` dalam waktu toko
Masukan salah	422 `{"message": "The given data was invalid.", "errors": {"field": ["..."]}}`, dengan semua galat sekaligus
Tidak ada, ditutup	404 `{"message": "No order 7."}`; 409 `{"message": "..."}` untuk permintaan yang dilarang keadaan record
Peran	hanya pemilik yang menghapus pelanggan; pelanggan yang punya pesanan tidak dapat dihapus (409)

Akun demo hanya menjadi bagian kontrak lewat baris perintah: versi Spring Boot dan ASP.NET Core membuat `kasir1@profitina-laundry.example` dan `owner@profitina-laundry.example`, sedangkan proyek Laravel Bab 10 mengisi `cashier@laundry.test` dan `owner@laundry.test`. Karena itu pemeriksa menerima keempat nilai itu sebagai argumen, dengan akun Spring dan ASP.NET sebagai bawaan.

16.3.4 Pemeriksanya

Kode 16.1 adalah seluruh programnya.

Kode 16.1 `check_api.php`: 25 pemeriksaan kontrak, untuk server mana pun dari ketiganya.

```php
<?php

// Kontrak capstone sebagai program: 16 pemeriksaan Bab 12, kini dengan
// login, peran, dan keluar. PHP biasa, tanpa paket:
//   php check_api.php <base-url> [cashier-email cashier-password
//                                 owner-email owner-password]

$base = rtrim($argv[1] ?? 'http://127.0.0.1:8080', '/');
[$cashierEmail, $cashierPassword, $ownerEmail, $ownerPassword] = [
    $argv[2] ?? 'kasir1@profitina-laundry.example', $argv[3] ?? 'Kasir#2026',
    $argv[4] ?? 'owner@profitina-laundry.example', $argv[5] ?? 'Owner#2026'];
$token = null;
$passed = $failed = 0;

// Mengirim satu permintaan dengan token kini; hasil [status, badan JSON].
function call(string $method, string $path, ?array $body = null): array
{
    global $base, $token;
    $headers = ['Accept: application/json',
                'Content-Type: application/json'];
    if ($token !== null) {
        $headers[] = "Authorization: Bearer $token";
    }
    $context = stream_context_create(['http' => [
        'method' => $method,
        'header' => implode("\r\n", $headers),
        'content' => $body === null ? '' : json_encode($body),
        'ignore_errors' => true,          // berikan juga badan jawaban 4xx
    ]]);
    $raw = @file_get_contents($base . $path, false, $context);
    if ($raw === false) {
        fwrite(STDERR, "Cannot reach $base$path\n");
        exit(2);
    }
    preg_match('/^HTTP\/\S+ (\d{3})/', $http_response_header[0], $m);
    return [(int) $m[1], json_decode($raw, true)];
}

function check(string $what, bool $ok): void
{
    global $passed, $failed;
    $ok ? $passed++ : $failed++;
    printf("%s  %s\n", $ok ? 'PASS' : 'FAIL', $what);
}
```

Kode 16.1 (Lanjutan) *check_api.php: 25 pemeriksaan kontrak, untuk server mana pun dari ketiganya.*

```
function login(string $email, string $password): array
{
    global $token;
    $token = null;
    return call('POST', '/api/tokens', ['email' => $email,
        'password' => $password, 'device' => 'contract check']);
}

// ---- login -------------------------------------------------------------
[$s, $j] = login($cashierEmail, $cashierPassword . 'x');
check('a wrong password -> 422, errors.email',
    $s === 422 && isset($j['errors']['email']));
[$s, $j] = call('GET', '/api/services');
check('no token -> 401 with a message',
    $s === 401 && is_string($j['message'] ?? null));
[$s, $j] = login($cashierEmail, $cashierPassword);
check('POST /api/tokens -> 201, a token for a cashier',
    $s === 201 && is_string($j['token'] ?? null)
    && ($j['user']['role'] ?? '') === 'cashier');
$token = $cashierToken = $j['token'] ?? '';

// ---- pemeriksaan Bab 12, sebagai kasir yang sudah login ----------------
$time = '/^\d{4}-\d\d-\d\dT\d\d:\d\d:\d\d$/';

[$s, $j] = call('GET', '/api/services');
$first = $j['data'][0] ?? [];
check('GET /api/services -> 200, S0001 first, 8000 per kg',
    $s === 200 && ($first['id'] ?? '') === 'S0001'
    && ($first['unit'] ?? '') === 'kg' && ($first['price'] ?? 0) === 8000);

$phone = '0812' . random_int(10000000, 99999999);   // baru pada setiap run
[$s, $j] = call('POST', '/api/customers', ['name' => 'Sri Rahayu',
    'phone' => substr($phone, 0, 4) . '-' . substr($phone, 4)]);
$customer = $j['data'] ?? [];
check('POST /api/customers -> 201, phone kept as digits only',
    $s === 201 && ($customer['phone'] ?? '') === $phone);
[$s, $j] = call('POST', '/api/customers', ['name' => 'Sri Rahayu',
    'phone' => $phone]);
check('the same phone again -> 422, errors.phone',
    $s === 422 && isset($j['errors']['phone']));
```

Kode 16.1 (Lanjutan) *check_api.php: 25 pemeriksaan kontrak, untuk server mana pun dari ketiganya.*

```
[$s, $j] = call('POST', '/api/orders', [
    'customer_id' => $customer['id'] ?? 0, 'outlet_id' => 'O01',
    'service_id' => 'S0001', 'quantity' => 2.5, 'pickup' => true,
    'delivery_address' => 'Jl. Kenanga 5, Surabaya']);
$o = $j['data'] ?? [];
check('POST /api/orders -> 201, total 34000 (3 kg + pickup), queued',
    $s === 201 && ($o['total'] ?? 0) === 34000
    && ($o['balance'] ?? 0) === 34000 && ($o['status'] ?? '') === 'queued');
check('times are YYYY-MM-DDTHH:MM:SS, finished_at is null',
    preg_match($time, $o['received_at'] ?? '') === 1
    && array_key_exists('finished_at', $o) && $o['finished_at'] === null);
$url = '/api/orders/' . ($o['id'] ?? 0);

[$s, $j] = call('POST', '/api/orders', ['quantity' => 0, 'pickup' => true]);
$fields = array_keys($j['errors'] ?? []);
check('a bad order -> 422, all of its errors at once',
    $s === 422 && is_string($j['message'] ?? null)
    && ! array_diff(['customer_id', 'service_id', 'quantity',
        'delivery_address'], $fields));

$move = fn (string $to) => call('PATCH', "$url/status", ['status' => $to]);
[$s, $j] = $move('departed');
check('queued -> departed skips a step: 422, errors.status',
    $s === 422 && isset($j['errors']['status']));
[$s, $j] = $move('finished');
check('queued -> finished: 200, finished_at set',
    $s === 200 && preg_match($time, $j['data']['finished_at'] ?? '') === 1);
[$s, $j] = $move('departed');
check('finished -> departed before payment: 422, errors.status',
    $s === 422 && isset($j['errors']['status']));

[$s, $j] = call('POST', "$url/payments", ['amount' => 40000,
    'method' => 'cash']);
check('paying more than the balance -> 422, errors.amount',
    $s === 422 && isset($j['errors']['amount']));
[$s, $j] = call('POST', "$url/payments", ['amount' => 34000,
    'method' => 'qris']);
check('paying the balance -> 201, balance 0, payments[0].method qris',
    $s === 201 && ($j['data']['balance'] ?? -1) === 0
    && ($j['data']['payments'][0]['method'] ?? '') === 'qris');
```

Kode 16.1 (Lanjutan) *check_api.php: 25 pemeriksaan kontrak, untuk server mana pun dari ketiganya.*

```
[$s, $j] = $move('departed');
check('finished -> departed after payment: 200',
    $s === 200 && ($j['data']['status'] ?? '') === 'departed');
[$s, $j] = call('POST', "$url/payments", ['amount' => 1000,
    'method' => 'cash']);
check('a departed order is closed: 409 with a message',
    $s === 409 && is_string($j['message'] ?? null));

[$s, $j] = call('GET', '/api/orders?status=departed');
check('GET /api/orders?status=departed -> 200, data[] and meta',
    $s === 200 && array_is_list($j['data'] ?? null)
    && ($j['meta']['total'] ?? 0) >= 1
    && ($j['meta']['current_page'] ?? 0) === 1);
[$s, $j] = call('GET', '/api/orders/999999');
check('GET /api/orders/999999 -> 404 with a message',
    $s === 404 && is_string($j['message'] ?? null));

[$s, $j] = call('GET', '/api/reports/daily?date=' . date('Y-m-d'));
$r = $j['data'] ?? [];
check('GET /api/reports/daily -> 200, the counts add up',
    $s === 200 && ($r['orders_received'] ?? 0) >= 1
    && ($r['collected'] ?? 0) >= 34000
    && array_sum($r['by_status'] ?? []) === $r['orders_received']);

// ---- peran: hanya pemilik yang menghapus pelanggan -----------------------
$gone = '/api/customers/' . ($customer['id'] ?? 0);
[$s, $j] = call('DELETE', $gone);
check('a cashier deletes a customer -> 403 with a message',
    $s === 403 && is_string($j['message'] ?? null));
[$s, $j] = login($ownerEmail, $ownerPassword);
$token = $j['token'] ?? '';
check('the owner logs in -> 201, role owner',
    $s === 201 && ($j['user']['role'] ?? '') === 'owner');
[$s, $j] = call('DELETE', $gone);
check('the owner deletes a customer with orders -> 409',
    $s === 409 && is_string($j['message'] ?? null));
[$s, $j] = call('POST', '/api/customers', ['name' => 'Temporary',
    'phone' => '0813' . random_int(10000000, 99999999)]);
[$s] = call('DELETE', '/api/customers/' . ($j['data']['id'] ?? 0));
check('the owner deletes a customer without orders -> 204', $s === 204);
```

Kode 16.1 (Lanjutan) *check_api.php: 25 pemeriksaan kontrak, untuk server mana pun dari ketiganya.*

```
// ---- keluar (sign out) ---------------------------------------------------
$token = $cashierToken;
[$s] = call('DELETE', '/api/tokens/current');
check('DELETE /api/tokens/current -> 204', $s === 204);
[$s] = call('GET', '/api/services');
check('the old token no longer works -> 401', $s === 401);

printf("\n%d passed, %d failed\n", $passed, $failed);
exit($failed === 0 ? 0 : 1);
```

`call` mengirim satu permintaan dengan fungsi *stream* bawaan PHP dan opsi konteks HTTP [1], sehingga program tidak memerlukan paket Composer dan berjalan dengan PHP 8.1 atau yang lebih baru. Opsi `ignore_errors` membuat PHP mengembalikan badan jawaban 4xx alih-alih gagal, dan kode statusnya diambil dari baris pertama `$http_response_header`. Setiap `check` mencetak `PASS` atau `FAIL` dengan kalimat yang menyatakan apa yang diharapkan, sehingga kegagalan terbaca seperti laporan *bug*. Kode keluarnya 0 hanya bila semuanya lulus, sehingga pemeriksa dapat berjalan otomatis setelah setiap *deployment*.

Tiga rincian membuat program dapat diulang. Nomor telepon pelanggan baru dibuat acak, sehingga run kedua tidak tersandung "The phone has already been taken." dari run pertama. Pelanggan yang berhasil dihapus pemilik dibuat untuk keperluan itu dalam run yang sama. Dan permintaan pesanan juga mengirim `outlet_id`, yang diwajibkan proyek Laravel Bab 10 dan diabaikan dua lainnya; field tambahan tidak berbahaya, field yang hilang berbahaya.

16.3.5 Membaca run yang gagal

Pemeriksa membuktikan nilainya pada hari ia gagal, sehingga cara membaca kegagalan layak diketahui. Setiap baris menyebut satu harapan, dan baris-baris itu berjalan dalam urutan sesi di Subbab 16.3.2, sehingga `FAIL` yang *pertama* biasanya yang perlu dipelajari; kegagalan berikutnya sering hanya akibatnya. Bila pemeriksaan login gagal, setiap pemeriksaan sesudahnya berjalan tanpa token dan sebagian besar gagal dengan 401: satu penyebab, dua puluh gejala. Bila `POST /api/orders` gagal, pemeriksaan langkah status dan pembayaran tidak punya pesanan untuk dikerjakan. Perbaiki kegagalan pertama, jalankan pemeriksa lagi, dan ulangi.

Dua jenis kegagalan pertama sering terjadi. Pemeriksaan yang mengharapkan sebuah field gagal karena field itu punya nama atau bentuk lain: `customerId` alih-alih `customer_id`, `"QUEUED"` alih-alih `"queued"`, tanggal dengan pecahan detik. Perbaikannya tempatnya di pengaturan JSON atau tampilan server, tidak pernah di pemeriksa. Pemeriksaan yang mengharapkan sebuah status gagal karena server menjawab dengan status lain: 400 alih-alih 422, 500 alih-alih 404. Maka badan jawabannya biasanya menyebut alasannya, dan cara tercepat melihatnya adalah mengirim permintaan yang sama dengan tangan memakai `curl -i`, yang mencetak baris status, *header*, dan badannya. Run pertama *capstone* ASP.NET Core, yang dijelaskan di Subbab 16.5.8, menempuh persis jalan ini.

16.3.6 Penyelesaian keliru yang menggoda

Cara yang menggoda untuk membandingkan server adalah menyimpan jawaban JSON lengkap dari satu server dan membandingkan yang lain dengannya, karakter demi karakter. Cara ini langsung gagal, dan karena alasan yang salah: pengenal, waktu, dan telepon acak berbeda pada setiap run; satu kerangka kerja menulis `4.5` sedangkan yang lain `4.50`; urutan field dalam objek tidak bermakna dalam JSON. Godaan sebaliknya adalah hanya memeriksa kode status. Maka server yang menjawab 422 dengan badan kosong, atau dengan `errors` berkunci `customerId`, akan lulus, dan aplikasi Raka tidak menampilkan pesan apa pun. Pemeriksa mengambil jalan tengah: ia menguji setiap field yang benar-benar dibaca aplikasi, menurut nama dan nilai, dan tidak menguji yang tidak dibaca.

16.4 Soal 16.2: Aturan dalam Tiga Bahasa

16.4.1 Soalnya

Harga dan alur kerja harus keluar sama persis dalam C#, PHP, dan Java, sampai ke rupiahnya, untuk setiap layanan dalam daftar harga, termasuk kasus-kasus yang janggal: jumlah dengan dua angka desimal, minimum kilogram, satuan yang dibulatkan ke atas, dan biaya antar-jemput di batasnya.

16.4.2 Memahaminya dengan tangan

Tabel 16.2 mengerjakan sembilan kasus dengan tangan memakai Persamaan (16.1). Jumlahnya ditulis dalam perseratus h; layanan kilogram menagih $\max(h, 300)$, layanan satuan membulatkan h ke atas menjadi ratusan utuh.

Tabel 16.2 Sembilan harga yang dikerjakan dengan tangan.

Layanan, harga	Jumlah	Ditagih	Subtotal	Antar-jemput	Total
Cuci-kering-lipat, 8.000	4,50 kg	450	36.000	tidak	36.000
Cuci-kering-lipat, 8.000	2,00 kg	300	24.000	tidak	24.000
Cuci-kering-lipat, 8.000	4,02 kg	402	32.160	tidak	32.160
Setrika saja, 6.000	3,35 kg	335	20.100	tidak	20.100
Gorden, 15.000	4,40 kg	440	66.000	ya	76.000
Bed cover, 35.000	2 buah	200	70.000	ya	80.000
Jas, 65.000	2 buah	200	130.000	ya	130.000
Kemeja, 20.000	5 buah	500	100.000	ya	100.000
Sepatu, 45.000	1,5 pasang	200	90.000	tidak	90.000

Tiga baris terakhir adalah batas-batasnya. Pesanan jas Rp 130.000 berada di atas Rp 100.000, sehingga antar-jemput gratis. Lima kemeja tepat Rp 100.000, yang *tidak* di bawah Rp 100.000, sehingga antar-jemput juga gratis; program yang menulis `<=` alih-alih `<` akan menagih Rp 10.000 di sini. Sepatu dihitung dalam pasang utuh, sehingga 1,5 pasang ditagih sebagai 2.

16.4.3 Penyelesaiannya

Ketiga implementasi punya bentuk yang sama: ubah jumlah menjadi perseratus utuh, hitung dengan bilangan bulat, dan bulatkan setengah ke atas dengan $+50$ sebelum membagi 100. Tidak ada bilangan *floating-point* di mana pun. Versi C# ada di *capstone* (**Kode 16.3**); versi PHP adalah `PriceCalculator` dari Bab 8, dan versi Java adalah `LaundryRules` dari Bab 11. Program pemeriksa Bab 11, yang disalin ke `Chapter16/spring/check`, mencetak kesembilan kasus **Tabel 16.2** dan alur kerjanya; program itu berjalan dengan JDK saja di mesin tempat buku ini ditulis:

```
Prices (Problem 8.1)
  Wash, dry & fold              4.50 kg            36000
  Wash, dry & fold              2.00 kg            24000
  Wash, dry & fold              4.02 kg            32160
  Ironing only                  3.35 kg            20100
  Curtain                       4.40 kg   pickup   76000
  Bed cover, any size           2.00 item pickup   80000
  Dry cleaning - suit 2 pieces  2.00 item pickup  130000
  Dry cleaning - shirt/blouse   5.00 item pickup  100000
  Shoes deep clean              1.50 pair          90000
Workflow (Problem 8.2)
  Queued   owing  25000 -> Finished
  Finished owing  25000 -> Collect Rp 25.000 before handing over the laundry.
  Finished owing      0 -> Departed
  Departed owing      0 -> This order has already left the shop.
```

Setiap total cocok dengan **Tabel 16.2**. Aturan C# menghasilkan angka yang sama di dalam *capstone* ASP.NET Core yang berjalan (Subbab 16.5.6), tempat pesanan kontrak keluar sebesar Rp 34.000 dan pesanan Sari Wulandari sebesar Rp 36.000.

16.4.4 Penyelesaian keliru yang menggoda

Penyelesaian yang menggoda adalah `Math.round(price * Math.max(quantity, 3))` dengan `double`, atau `round($price * max($qty, 3))` di PHP. Lebih pendek, dan benar untuk sebagian besar masukan. Ia gagal ketika pecahan biner tidak dapat menyatakan sebuah desimal dengan tepat: 4,02 disimpan sebagai 4,0199999999999996..., dan `8000 * 4.02` menghasilkan 32159,999999999996, yang kebetulan diselamatkan pembulatan *float* di sini tetapi tidak oleh *cast* yang memotong, mode pembulatan yang berbeda, atau harga yang sedikit berbeda. Lebih buruk lagi, ketiga bahasa membulatkan `.5` secara berbeda secara bawaan (`Math.Round` di C# membulatkan setengah ke genap), sehingga ketiga server akan berselisih satu rupiah pada kasus yang jarang, jenis *bug* yang paling sulit ditemukan. *Floating point* biner, sebagaimana dibakukan dalam IEEE 754 [2], eksak untuk bilangan bulat tetapi tidak untuk sebagian besar pecahan desimal; perseratus utuh dan aritmetika bilangan bulat menghapus seluruh golongan masalah itu.

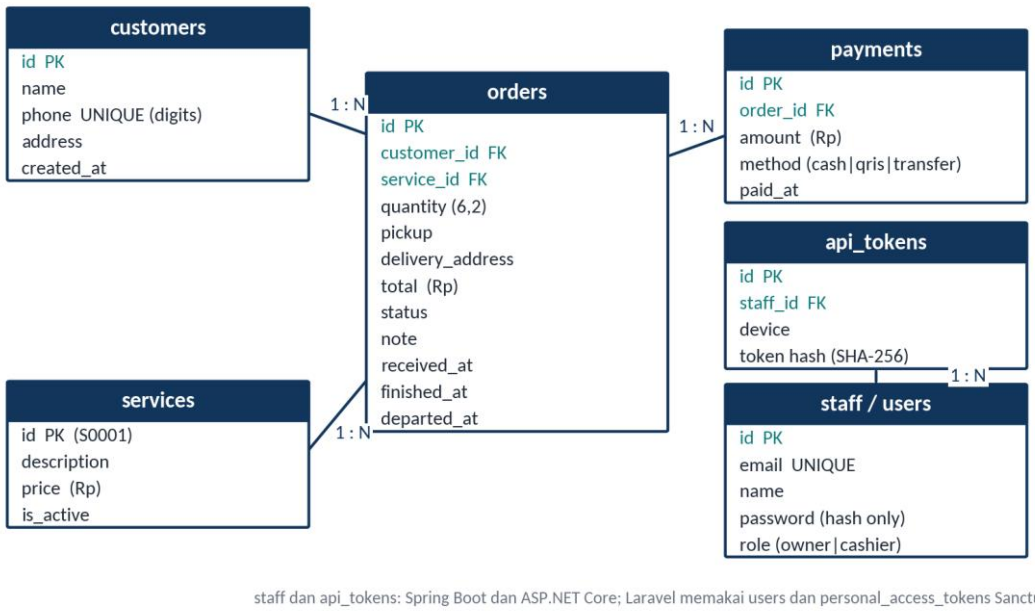
16.5 Soal 16.3: Capstone ASP.NET Core

16.5.1 Soalnya

Bangun API laundry dengan ASP.NET Core 10 sehingga lulus ke-25 pemeriksaan, dengan token *bearer* bergaya Sanctum, hak pemilik untuk menghapus pelanggan, aturan Soal 16.2, dan papan publik. Simpan data di SQLite melalui EF Core. Karena mesin tempat buku ini ditulis tidak dapat mengunduh EF Core, aplikasi juga harus dapat dibangun tanpanya, agar dapat dijalankan dan diperiksa sungguhan.

16.5.2 Merancangnya di atas kertas

Gambar 16.2 adalah model datanya, sama di ketiga kerangka kerja kecuali tempat akun-akun disimpan.



Gambar 16.2 Tabel-tabel *capstone*: layanan dari daftar harga (`S0001` sampai `S0012`), pelanggan, pesanan dengan antar-jemput, pembayaran, staf, dan token API.

Lapisannya adalah lapisan Bab 13, dengan nama-nama C#. Controller di `Api` membaca JSON dan menulis JSON. `LaundryService` di `Application` menerapkan setiap aturan. Penyimpanan berada di balik antarmuka, `ILaundryStore`, dengan dua implementasi: `EfLaundryStore` dengan EF Core dan SQLite, dan `MemoryLaundryStore` dengan *list* di memori. Sebuah sakelar di berkas proyek memilih salah satunya saat *build* (**Kode 16.2**).

Kode 16.2 `LaundryCapstone.csproj`: sakelar `UseEf` memutuskan apakah EF Core ikut dikompilasi.

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

  <PropertyGroup>
    <TargetFramework>net10.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
    <!-- true: SQLite melalui EF Core (build normal).
         false: store di memori yang tidak butuh paket NuGet. -->
    <UseEf Condition="'$(UseEf)' == ''">true</UseEf>
    <DefineConstants Condition="'$(UseEf)' == 'true'"
      >$(DefineConstants);USE_EF</DefineConstants>
  </PropertyGroup>

  <ItemGroup Condition="'$(UseEf)' == 'true'">
    <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite"
                      Version="10.0.*" />
  </ItemGroup>

  <ItemGroup Condition="'$(UseEf)' != 'true'">
    <Compile Remove="Data/Ef/**" />
  </ItemGroup>

  <ItemGroup>
    <Content Include="data/Service.csv" CopyToOutputDirectory="PreserveNewest" />
  </ItemGroup>

</Project>
```

Dengan bawaan `UseEf=true`, proyek merujuk `Microsoft.EntityFrameworkCore.Sqlite` dan mendefinisikan simbol `USE_EF`; dengan `-p:UseEf=false` proyek tidak merujuk paket apa pun, membuang folder `Data/Ef` dari *build*, dan tidak memerlukan apa pun selain .NET SDK. Semua bagian lain aplikasi sama persis pada kedua *build*.

16.5.3 Domain dan penyimpanan

Kode 16.3 memuat aturan Soal 16.2 dan alur kerjanya.

Kode 16.3 `LaundryRules.cs`: harga dan alur kerja dalam aritmetika bilangan bulat, seperti di Bab 8 dan 11.

```
using System.Globalization;

namespace LaundryCapstone.Domain;

/// <summary>Aturan toko: aritmetika yang sama dengan Bab 8 dan 11.</summary>
public static class LaundryRules
{
    public const long MinimumKg = 3;
    public const long PickupFee = 10_000;
    public const long FreePickupFrom = 100_000;

    /// <summary>4.5 -> 450: jumlah dalam perseratus, eksak.</summary>
    public static long Hundredths(decimal quantity) =>
        decimal.ToInt64(quantity * 100m);

    /// <summary>Minimal 3 kg untuk layanan per kg; lainnya satuan utuh.</summary>
    public static long Total(bool perKg, long price, decimal quantity,
                             bool pickup)
    {
        long h = Hundredths(quantity);
        long billed = perKg ? Math.Max(h, MinimumKg * 100)
                            : (h + 99) / 100 * 100;
        long subtotal = (price * billed + 50) / 100;         // setengah ke atas
        long fee = pickup && subtotal < FreePickupFrom ? PickupFee : 0;
        return subtotal + fee;
    }

    /// <summary>Status berikutnya, atau eksepsi yang menyebut alasannya.</summary>
    public static OrderStatus Advance(OrderStatus status, long balance)
    {
        OrderStatus next = status switch
        {
            OrderStatus.Queued => OrderStatus.Finished,
            OrderStatus.Finished => OrderStatus.Departed,
            _ => throw new InvalidOperationException(
                     "This order has already left the shop.")
        };
        if (next == OrderStatus.Departed && balance > 0)
            throw new InvalidOperationException(
                $"Collect {Rupiah(balance)} before handing over the laundry.");
        return next;
    }
```

Kode 16.3 (Lanjutan) *LaundryRules.cs: harga dan alur kerja dalam aritmetika bilangan bulat, seperti di Bab 8 dan 11.*

```
    /// <summary>36000 -> "Rp 36.000".</summary>
    public static string Rupiah(long amount) =>
        "Rp " + amount.ToString("#,0", CultureInfo.InvariantCulture)
                      .Replace(',', '.');

    /// <summary>Queued -> "queued": kata yang disimpan dan dikirim.</summary>
    public static string Value(this OrderStatus s) =>
        s.ToString().ToLowerInvariant();

    public static string Value(this PaymentMethod m) =>
        m.ToString().ToLowerInvariant();
}
```

Entitas-entitasnya adalah kelas biasa di `Domain/Entities.cs`. `Service` menyimpan kolom-kolom daftar harga (`Id` seperti `S0001`, `Description`, `Price`, `Active`) dan menurunkan `Name`, `PerKg`, dan `Unit` dari deskripsi, persis seperti kelas Java di Bab 11, sehingga "Wash, dry & fold (per kg)" menjadi nama "Wash, dry & fold" dan satuan `kg`. `Order` menghitung `Paid` dan `Balance` dari pembayarannya. **Kode 16.4** adalah antarmuka yang dipakai service.

Kode 16.4 `ILaundryStore.cs`: semua yang diperlukan aplikasi dari penyimpanan.

```
using LaundryCapstone.Domain;

namespace LaundryCapstone.Data;

/// <summary>
/// Semua yang diperlukan aplikasi dari penyimpanan. EfLaundryStore menyimpan
/// di SQLite; MemoryLaundryStore dalam list (build tanpa NuGet).
/// </summary>
public interface ILaundryStore
{
    Task<List<Service>> ActiveServicesAsync();
    Task<Service?> FindServiceAsync(string id);
    Task<List<Customer>> CustomersAsync();
    Task<Customer?> FindCustomerAsync(long id);
    Task<Customer?> FindCustomerByPhoneAsync(string phone);
    Task<bool> CustomerHasOrdersAsync(long customerId);
    Task<(List<Order> Items, int Total)> OrderPageAsync(
        OrderStatus? status, int page, int size);
    Task<Order?> FindOrderAsync(long id);        // dengan pelanggan, layanan,
                                                 // dan pembayaran
    Task<Dictionary<OrderStatus, long>> CountByStatusAsync(
        DateTime from, DateTime to);
    Task<long> BilledAsync(DateTime from, DateTime to);
    Task<long> CollectedAsync(DateTime from, DateTime to);
    Task<Staff?> FindStaffAsync(string email);
    Task<bool> AnyServiceAsync();
    Task<ApiToken?> FindTokenAsync(long id);
    void Add(object entity);
    void Remove(object entity);
    Task SaveChangesAsync();
    Task EnsureCreatedAsync();
}
```

Antarmuka ini berbicara dengan istilah aplikasi (`CustomerHasOrdersAsync`, `BilledAsync`), bukan istilah tabel atau kueri. `EfLaundryStore` mengimplementasikan setiap metode dengan kueri LINQ atas `LaundryDbContext`; `MemoryLaundryStore` dengan kueri yang sama atas *list*. `Add`, `Remove`, dan `SaveChangesAsync` mencerminkan *unit of work* EF Core: pada *build* EF ketiganya adalah metode EF Core sendiri, pada *build* memori `SaveChangesAsync` membagikan pengenal kepada baris-baris baru. *Store* memori bukan mainan: dialah yang membuat *capstone* dapat dijalankan di mesin tanpa paket, dan ia pengganti yang praktis untuk tes yang cepat. Tetapi ia juga membawa risiko yang harus disebut bab ini. Dua implementasi dari satu antarmuka dapat berbeda pendapat, misalnya tentang urutan baris yang dikembalikan sebuah kueri atau tentang apakah pesanan milik pelanggan yang dihapus masih menunjuk kepadanya, dan pemeriksa yang lulus terhadap salah satunya tidak menyatakan apa pun yang pasti tentang yang lain. Itulah sebabnya Subbab 16.5.6 menyebut dengan tepat *build* mana yang diperiksa.

16.5.4 Service-nya

Kode 16.5 adalah jantung *capstone*.

Kode 16.5 `LaundryService.cs`: pelanggan, pesanan, langkah status, pembayaran, dan angka harian, dengan setiap aturan di satu tempat.

```
using LaundryCapstone.Data;
using LaundryCapstone.Domain;

namespace LaundryCapstone.Application;

/// <summary>Yang diperlukan pesanan baru, dari form maupun JSON.</summary>
public sealed record NewOrder(long? CustomerId, string? ServiceId,
    decimal? Quantity, bool Pickup, string? DeliveryAddress, string? Note);

/// <summary>Pesanan per status, uang ditagih dan diterima sehari.</summary>
public sealed record DayFigures(DateOnly Day,
    Dictionary<OrderStatus, long> ByStatus, long Billed, long Collected)
{
    public long Received => ByStatus.Values.Sum();
}

/// <summary>Lapisan service: semua aturan toko diterapkan di sini.</summary>
public sealed class LaundryService(ILaundryStore store, TimeProvider clock)
{
    // Indonesia (WIB) adalah UTC+7 sepanjang tahun: tanpa waktu musim panas.
    private static readonly TimeSpan Wib = TimeSpan.FromHours(7);

    private DateTime Now()
    {
        var t = clock.GetUtcNow().ToOffset(Wib).DateTime;
        return t.AddTicks(-(t.Ticks % TimeSpan.TicksPerSecond));
    }

    public Task<List<Service>> ActiveServicesAsync() =>
        store.ActiveServicesAsync();

    public async Task<Service> ServiceAsync(string id) =>
        await store.FindServiceAsync(id)
        ?? throw new NotFoundException("service", id);

    public async Task<List<Customer>> CustomersAsync(string? phone)
    {
        if (phone is null) return await store.CustomersAsync();
        var found = await store.FindCustomerByPhoneAsync(Digits(phone));
        return found is null ? [] : [found];
    }
```

Kode 16.5 (Lanjutan) *LaundryService.cs: pelanggan, pesanan, langkah status, pembayaran, dan angka harian, dengan setiap aturan di satu tempat.*

```
    public async Task<Customer> CreateCustomerAsync(string name, string phone,
                                                    string? address)
    {
        if (await store.FindCustomerByPhoneAsync(phone) is not null)
            throw new RuleException("phone",
                "The phone has already been taken.");
        var customer = new Customer
        {
            Name = name.Trim(), Phone = phone,
            Address = string.IsNullOrWhiteSpace(address) ? null : address,
            CreatedAt = Now()
        };
        store.Add(customer);
        await store.SaveChangesAsync();
        return customer;
    }

    public async Task DeleteCustomerAsync(long id)
    {
        var customer = await store.FindCustomerAsync(id)
                       ?? throw new NotFoundException("customer", id);
        if (await store.CustomerHasOrdersAsync(id))
            throw new ConflictException(
                $"{customer.Name} has orders and cannot be deleted.");
        store.Remove(customer);
        await store.SaveChangesAsync();
    }

    public Task<(List<Order> Items, int Total)> OrderPageAsync(
        OrderStatus? status, int page, int size) =>
        store.OrderPageAsync(status, Math.Max(page, 1), size);

    public async Task<Order> OrderAsync(long id) =>
        await store.FindOrderAsync(id)
        ?? throw new NotFoundException("order", id);
```

Kode 16.5 (Lanjutan) *LaundryService.cs: pelanggan, pesanan, langkah status, pembayaran, dan angka harian, dengan setiap aturan di satu tempat.*

```
    /// <summary>Aturan yang perlu basis data; kosong berarti beres.</summary>
    public async Task<Dictionary<string, List<string>>> CheckAsync(NewOrder o)
    {
        var errors = new Dictionary<string, List<string>>();
        if (o.CustomerId is long c && await store.FindCustomerAsync(c) is null)
            Add(errors, "customer_id", "The selected customer is invalid.");
        if (!string.IsNullOrWhiteSpace(o.ServiceId)
            && (await store.FindServiceAsync(o.ServiceId))?.Active != true)
            Add(errors, "service_id", "The selected service is invalid.");
        if (o.Pickup && string.IsNullOrWhiteSpace(o.DeliveryAddress))
            Add(errors, "delivery_address",
                "Where should the laundry be delivered?");
        return errors;
    }

    public async Task<Order> CreateOrderAsync(NewOrder o)
    {
        var errors = await CheckAsync(o);
        if (errors.Count > 0) throw new RuleException(errors);
        var service = (await store.FindServiceAsync(o.ServiceId!))!;
        var order = new Order
        {
            CustomerId = o.CustomerId!.Value, ServiceId = service.Id,
            Quantity = o.Quantity!.Value, Pickup = o.Pickup,
            DeliveryAddress = o.Pickup ? o.DeliveryAddress : null,
            Note = o.Note, ReceivedAt = Now(),
            Total = LaundryRules.Total(service.PerKg, service.Price,
                                       o.Quantity.Value, o.Pickup)
        };
        store.Add(order);
        await store.SaveChangesAsync();
        return await OrderAsync(order.Id);
    }

    /// <summary>Satu langkah; status yang diminta harus yang berikutnya.</summary>
    public async Task<Order> AdvanceAsync(long id, OrderStatus? wanted)
    {
        var order = await OrderAsync(id);
        EnsureOpen(order);
        OrderStatus next;
        try { next = LaundryRules.Advance(order.Status, order.Balance); }
        catch (InvalidOperationException e)
        {
            throw new RuleException("status", e.Message);
        }
        if (wanted is not null && wanted != next)
```

Kode 16.5 (Lanjutan) *LaundryService.cs: pelanggan, pesanan, langkah status, pembayaran, dan angka harian, dengan setiap aturan di satu tempat.*

```
            throw new RuleException("status", $"An order that is "
                + $"{order.Status.Value()} can only become {next.Value()}.");
        if (next == OrderStatus.Finished) order.FinishedAt = Now();
        else order.DepartedAt = Now();
        order.Status = next;
        await store.SaveChangesAsync();
        return order;
    }

    public async Task<Order> PayAsync(long id, long amount, PaymentMethod method)
    {
        var order = await OrderAsync(id);
        EnsureOpen(order);
        if (amount > order.Balance)
            throw new RuleException("amount",
                $"Only {LaundryRules.Rupiah(order.Balance)} is still owed.");
        var payment = new Payment
        {
            OrderId = id, Amount = amount, Method = method, PaidAt = Now()
        };
        order.Payments.Add(payment);
        store.Add(payment);
        await store.SaveChangesAsync();
        return order;
    }

    public async Task<DayFigures> DailyAsync(DateOnly day)
    {
        var from = day.ToDateTime(TimeOnly.MinValue);
        var to = from.AddDays(1);                  // [from, to): satu hari penuh
        var counts = await store.CountByStatusAsync(from, to);
        var byStatus = Enum.GetValues<OrderStatus>()
            .ToDictionary(s => s, s => counts.GetValueOrDefault(s));
        return new DayFigures(day, byStatus, await store.BilledAsync(from, to),
                              await store.CollectedAsync(from, to));
    }

    /// <summary>Cucian yang sudah keluar tak menerima langkah/uang lagi.</summary>
    private static void EnsureOpen(Order order)
    {
        if (order.Status == OrderStatus.Departed)
            throw new ConflictException(
                $"Order #{order.Id} has left the shop and is closed.");
    }
```

Kode 16.5 (Lanjutan) *LaundryService.cs: pelanggan, pesanan, langkah status, pembayaran, dan angka harian, dengan setiap aturan di satu tempat.*

```
    public static string Digits(string s) => new(s.Where(char.IsAsciiDigit)
                                                  .ToArray());

    private static void Add(Dictionary<string, List<string>> errors,
                            string field, string message)
    {
        if (!errors.TryGetValue(field, out var list)) errors[field] = list = [];
        list.Add(message);
    }
}
```

Taruh di samping `OrderService` Bab 13 dan kesesuaiannya terlihat baris demi baris: `NewOrder`, `CheckAsync` yang mengembalikan setiap aturan basis data sebagai *dictionary*, `CreateOrderAsync` yang memeriksa lagi, `AdvanceAsync` yang menolak pesanan tertutup dengan `ConflictException`

dan langkah yang salah dengan `RuleException`, `PayAsync` yang menolak pembayaran berlebih. Ketiga eksepsinya sama dengan di Spring (**Kode 16.6**).

Kode 16.6 `Exceptions.cs`: aturan yang dilanggar, record yang tidak ada, keadaan yang bertentangan.

```
namespace LaundryCapstone.Application;

/// <summary>Masukan yang melanggar aturan, per field: 422.</summary>
public sealed class RuleException(Dictionary<string, List<string>> errors)
    : Exception("The given data was invalid.")
{
    public Dictionary<string, List<string>> Errors { get; } = errors;

    public RuleException(string field, string message)
        : this(new() { [field] = [message] }) { }
}

/// <summary>Meminta record yang tidak ada: 404.</summary>
public sealed class NotFoundException(string what, object id)
    : Exception($"No {what} {id}.");

/// <summary>Tidak mungkin pada keadaan record kini: 409.</summary>
public sealed class ConflictException(string message) : Exception(message);
```

Dua rincian khas .NET. Waktu berasal dari `TimeProvider`, jam .NET yang dapat diinjeksi, diubah ke Waktu Indonesia Barat (UTC+7, tanpa waktu musim panas) dan dipotong menjadi detik utuh, agar format waktu kontrak terpenuhi. Dan kelas ini memakai *primary constructor*, `LaundryService(ILaundryStore store, TimeProvider clock)`, bentuk pendek C# untuk *constructor injection*.

16.5.5 API, galat, dan pintu-pintunya

Kode 16.7 menggambarkan apa yang masuk dan apa yang keluar.

Kode 16.7 `Dtos.cs`: *record* permintaan dengan atribut validasi, dan tampilan dalam bentuk kontrak.

```
using System.ComponentModel.DataAnnotations;
using System.Text.Json.Serialization;
using LaundryCapstone.Domain;

namespace LaundryCapstone.Api;

// ---- yang masuk (divalidasi sebelum dilihat service) ----

public sealed record TokenRequest(
    [Required, EmailAddress] string? Email,
    [Required] string? Password,
    [Required, MaxLength(60)] string? Device);

public sealed record CustomerRequest(
    [Required, StringLength(80, MinimumLength = 2)] string? Name,
    [Required] string? Phone,
    [MaxLength(200)] string? Address);

public sealed record OrderRequest(
    [Required] long? CustomerId,
    [Required] string? ServiceId,
    [Required, Range(typeof(decimal), "0.1", "100",
        ParseLimitsInInvariantCulture = true), TwoDecimals] decimal? Quantity,
    bool Pickup,
    [MaxLength(200)] string? DeliveryAddress,
    [MaxLength(200)] string? Note);

public sealed record StatusRequest([Required] string? Status);

public sealed record PaymentRequest(
    [Required, Range(1000, long.MaxValue)] long? Amount,
    [Required] string? Method);

/// <summary>Paling banyak dua angka di belakang koma, mis. 4.02 kg.</summary>
public sealed class TwoDecimalsAttribute : ValidationAttribute
{
    public TwoDecimalsAttribute() : base("must have at most two decimals") { }

    public override bool IsValid(object? value) =>
        value is not decimal d || decimal.Round(d, 2) == d;
}

// ---- yang keluar: JSON kontrak Bab 12 ----

public sealed record DataView<T>(T Data);
```

Kode 16.7 (Lanjutan) *Dtos.cs: record permintaan dengan atribut validasi, dan tampilan dalam bentuk kontrak.*

```
public sealed record PageMeta(int CurrentPage, int PerPage, long Total,
                              int LastPage);

public sealed record PageView<T>(List<T> Data, PageMeta Meta);

public sealed record ServiceView(string Id, string Name, string Unit,
                                 long Price)
{
    public static ServiceView Of(Service s) => new(s.Id, s.Name, s.Unit,
                                                   s.Price);
}

public sealed record CustomerView(long Id, string Name, string Phone,
                                  string? Address)
{
    public static CustomerView Of(Customer c) => new(c.Id, c.Name, c.Phone,
                                                     c.Address);
}

public sealed record PaymentView(long Id, long Amount, string Method,
                                 string PaidAt)
{
    public static PaymentView Of(Payment p) =>
        new(p.Id, p.Amount, p.Method.Value(), Json.Time(p.PaidAt)!);
}

public sealed record OrderView(long Id, CustomerView Customer,
    ServiceView Service, decimal Quantity, bool Pickup,
    string? DeliveryAddress, long Total, long Paid, long Balance,
    string Status, string? Note, string ReceivedAt, string? FinishedAt,
    string? DepartedAt,
    [property: JsonIgnore(Condition = JsonIgnoreCondition.WhenWritingNull)]
    List<PaymentView>? Payments)
{
    /// <summary>Tanpa daftar pembayaran, seperti Laravel bila tak dimuat.</summary>
    public static OrderView Of(Order o, bool withPayments = false) => new(
        o.Id, CustomerView.Of(o.Customer), ServiceView.Of(o.Service),
        o.Quantity, o.Pickup, o.DeliveryAddress, o.Total, o.Paid, o.Balance,
        o.Status.Value(), o.Note, Json.Time(o.ReceivedAt)!,
        Json.Time(o.FinishedAt), Json.Time(o.DepartedAt),
        withPayments ? o.Payments.Select(PaymentView.Of).ToList() : null);
}

public sealed record DailyReport(string Date, long OrdersReceived,
    long Billed, long Collected, Dictionary<string, long> ByStatus);
```

Kode 16.7 (Lanjutan) *Dtos.cs: record permintaan dengan atribut validasi, dan tampilan dalam bentuk kontrak.*

```
public sealed record UserView(string Name, string Role);

public sealed record TokenView(string Token, UserView User);

public static class Json
{
    /// <summary>Satu-satunya format waktu kontrak: 2026-09-25T08:15:00.</summary>
    public static string? Time(DateTime? t) => t?.ToString("yyyy-MM-dd'T'HH:mm:ss");
}
```

Record permintaan memakai tipe *nullable* (`long?`, `decimal?`) dengan `[Required]`, agar field yang hilang dilaporkan sebagai hilang alih-alih diam-diam menjadi 0. Tampilan-tampilannya adalah *record* yang propertinya ditulis dalam PascalCase C#; opsi JSON di `Program.cs` mengubahnya

menjadi snake_case (`DeliveryAddress` menjadi `delivery_address`). `DataView<T>` dan `PageView<T>` adalah amplopnya. **Kode 16.8** adalah controller pesanan yang tipis.

Kode 16.8 `OrdersController.cs`: baca JSON, panggil `LaundryService`, tulis JSON.

```
using LaundryCapstone.Application;
using LaundryCapstone.Domain;
using Microsoft.AspNetCore.Mvc;

namespace LaundryCapstone.Api;

/// <summary>Tipis: baca JSON, panggil LaundryService, tulis JSON.</summary>
[ApiController]
[Route("api/orders")]
public class OrdersController(LaundryService laundry) : ControllerBase
{
    [HttpGet]
    public async Task<IActionResult> Index(string? status, int page = 1)
    {
        OrderStatus? wanted = null;
        if (status is not null)
        {
            if (!Enum.TryParse<OrderStatus>(status, true, out var s))
                return Invalid("status", "The selected status is invalid.");
            wanted = s;
        }
        page = Math.Max(page, 1);
        var (items, total) = await laundry.OrderPageAsync(wanted, page, 20);
        return Ok(new PageView<OrderView>(
            items.Select(o => OrderView.Of(o)).ToList(),
            new PageMeta(page, 20, total, Math.Max(1, (total + 19) / 20))));
    }

    [HttpGet("{id:long}")]
    public async Task<DataView<OrderView>> Show(long id) =>
        new(OrderView.Of(await laundry.OrderAsync(id), withPayments: true));

    /// <summary>Setiap aturan yang dilanggar dilaporkan sekaligus.</summary>
    [HttpPost]
    public async Task<IActionResult> Store(OrderRequest body)
    {
        var order = new NewOrder(body.CustomerId, body.ServiceId,
            body.Quantity, body.Pickup, body.DeliveryAddress, body.Note);
        var rules = await laundry.CheckAsync(order);
        if (!ModelState.IsValid || rules.Count > 0)
            return ApiErrors.Unprocessable(ModelState, rules);
        return StatusCode(201, new DataView<OrderView>(
            OrderView.Of(await laundry.CreateOrderAsync(order))));
    }
```

Kode 16.8 (Lanjutan) *OrdersController.cs: baca JSON, panggil LaundryService, tulis JSON.*

```
    [HttpPatch("{id:long}/status")]
    public async Task<IActionResult> UpdateStatus(long id, StatusRequest body)
    {
        if (!ModelState.IsValid) return ApiErrors.Unprocessable(ModelState);
        if (!Enum.TryParse<OrderStatus>(body.Status, true, out var wanted))
            return Invalid("status", "The selected status is invalid.");
        return Ok(new DataView<OrderView>(
            OrderView.Of(await laundry.AdvanceAsync(id, wanted))));
    }

    [HttpPost("{id:long}/payments")]
    public async Task<IActionResult> Pay(long id, PaymentRequest body)
    {
        if (!ModelState.IsValid) return ApiErrors.Unprocessable(ModelState);
        if (!Enum.TryParse<PaymentMethod>(body.Method, true, out var method))
            return Invalid("method", "The selected method is invalid.");
        var order = await laundry.PayAsync(id, body.Amount!.Value, method);
        return StatusCode(201, new DataView<OrderView>(
            OrderView.Of(order, withPayments: true)));
    }

    private ObjectResult Invalid(string field, string message) =>
        ApiErrors.Unprocessable(ModelState, new() { [field] = [message] });
}
```

`Store` menggabungkan galat atribut validasi (`ModelState`) dengan aturan service dan menjawab sekali, seperti `OrderApiController.store` di Spring. Itu hanya berjalan berkat satu baris di `Program.cs`: `SuppressModelStateInvalidFilter = true`. Secara bawaan, `[ApiController]` menjawab masukan yang salah sendiri, *sebelum* aksinya berjalan, dengan status 400 dan dokumen *problem details*; kontrak menginginkan 422, bentuk Laravel, dan aturan service dalam jawaban yang sama. **Kode 16.9** membangun jawaban itu dan memetakan eksepsi service.

Kode 16.9 `ApiErrors.cs`: setiap galat dalam bentuk kontrak.

```
using System.Text.Json;
using LaundryCapstone.Application;
using Microsoft.AspNetCore.Diagnostics;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.ModelBinding;

namespace LaundryCapstone.Api;

/// <summary>Setiap galat dalam bentuk kontrak: {"message", "errors"}.</summary>
public static class ApiErrors
{
    public const string Invalid = "The given data was invalid.";

    /// <summary>422 dengan galat model ditambah galat aturan.</summary>
    public static ObjectResult Unprocessable(ModelStateDictionary model,
        Dictionary<string, List<string>>? more = null)
    {
        var errors = new Dictionary<string, List<string>>();
        foreach (var (key, entry) in model)
        {
            if (entry.Errors.Count == 0) continue;
            var list = errors.TryGetValue(Snake(key), out var l) ? l
                     : errors[Snake(key)] = [];
            list.AddRange(entry.Errors.Select(e => e.ErrorMessage));
        }
        foreach (var (key, messages) in more ?? [])
        {
            var list = errors.TryGetValue(key, out var l) ? l : errors[key] = [];
            list.AddRange(messages);
        }
        return new ObjectResult(new { message = Invalid, errors })
        { StatusCode = StatusCodes.Status422UnprocessableEntity };
    }

    /// <summary>"CustomerId" or "$.customer_id" -> "customer_id".</summary>
    public static string Snake(string key)
    {
        key = key.StartsWith("$.") ? key[2..] : key;
        return JsonNamingPolicy.SnakeCaseLower.ConvertName(key);
    }
}
```

Kode 16.9 (Lanjutan) *ApiErrors.cs: setiap galat dalam bentuk kontrak.*

```
/// <summary>Eksepsi kita sebagai jawaban 422, 404, dan 409.</summary>
public sealed class ApiExceptionHandler : IExceptionHandler
{
    public async ValueTask<bool> TryHandleAsync(HttpContext http,
        Exception error, CancellationToken cancel)
    {
        (int status, object body) = error switch
        {
            RuleException e => (422, (object)new
            {
                message = ApiErrors.Invalid, errors = e.Errors
            }),
            NotFoundException e => (404, new { message = e.Message }),
            ConflictException e => (409, new { message = e.Message }),
            _ => (0, new { })
        };
        if (status == 0) return false;                  // bug sungguhan: 500
        http.Response.StatusCode = status;
        await http.Response.WriteAsJsonAsync(body, cancel);
        return true;
    }
}
```

`ApiExceptionHandler` mengimplementasikan `IExceptionHandler`, kait ASP.NET Core untuk eksepsi yang lolos dari controller [3]. Ia menjawab `RuleException`, `NotFoundException`, dan `ConflictException` dengan 422, 404, dan 409 serta mengembalikan `false` untuk yang lain, sehingga *bug* sungguhan tetap menjadi 500 dan dicatat di log. Controller pelanggan (**Kode 16.10**) memperlihatkan pemeriksaan peran.

Kode 16.10 `CustomersController.cs`: telepon disimpan sebagai angka, dan `DELETE` hanya untuk pemilik.

```
using System.Text.RegularExpressions;
using LaundryCapstone.Application;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace LaundryCapstone.Api;

[ApiController]
[Route("api/customers")]
public partial class CustomersController(LaundryService laundry)
    : ControllerBase
{
    [HttpGet]
    public async Task<DataView<List<CustomerView>>> Index(string? phone) =>
        new((await laundry.CustomersAsync(phone)).Select(CustomerView.Of)
                                                  .ToList());

    [HttpPost]
    public async Task<IActionResult> Store(CustomerRequest body)
    {
        // "0812-3456-7890" disimpan sebagai angka, lalu diperiksa, seperti Laravel
        string phone = LaundryService.Digits(body.Phone ?? "");
        if (body.Phone is not null && !MobilePhone().IsMatch(phone))
            ModelState.AddModelError("phone",
                "Write the phone as 08..., 10 to 13 digits.");
        if (!ModelState.IsValid) return ApiErrors.Unprocessable(ModelState);
        var customer = await laundry.CreateCustomerAsync(body.Name!, phone,
                                                         body.Address);
        return StatusCode(201, new DataView<CustomerView>(
            CustomerView.Of(customer)));
    }

    [HttpDelete("{id:long}"), Authorize(Roles = "OWNER")]
    public async Task<IActionResult> Destroy(long id)
    {
        await laundry.DeleteCustomerAsync(id);
        return NoContent();
    }

    [GeneratedRegex("^08[0-9]{8,11}$")]
    private static partial Regex MobilePhone();
}
```

`[Authorize(Roles = "OWNER")]` pada `Destroy` adalah bentuk ASP.NET Core dari aturan URL di Spring; service lalu menolak pelanggan yang punya pesanan dengan 409. Telepon dipangkas menjadi angka dan diperiksa dengan ekspresi reguler yang diubah kompilator menjadi kode saat *build* (`[GeneratedRegex]`). Pintu-pintunya sendiri adalah *authentication handler* (**Kode 16.11**) dan service token (**Kode 16.12**).

Kode 16.11 `BearerTokenHandler.cs`: dari token *bearer* ke staf yang sudah login, dengan 401 dan 403 milik kontrak.

```
using System.Security.Claims;
using System.Text.Encodings.Web;
using LaundryCapstone.Application;
using Microsoft.AspNetCore.Authentication;
using Microsoft.Extensions.Options;

namespace LaundryCapstone.Security;

/// <summary>"Authorization: Bearer 12|secret" -> pemilik token 12.</summary>
public sealed class BearerTokenHandler(
    IOptionsMonitor<AuthenticationSchemeOptions> options, ILoggerFactory logger,
    UrlEncoder encoder, TokenService tokens)
    : AuthenticationHandler<AuthenticationSchemeOptions>(options, logger, encoder)
{
    public const string SchemeName = "Bearer";

    protected override async Task<AuthenticateResult> HandleAuthenticateAsync()
    {
        string header = Request.Headers.Authorization.ToString();
        if (!header.StartsWith("Bearer ", StringComparison.Ordinal))
            return AuthenticateResult.NoResult();       // tetap anonim
        var token = await tokens.FindAsync(header[7..].Trim());
        if (token is null) return AuthenticateResult.Fail("Unknown token.");

        Claim[] claims =
        [
            new(ClaimTypes.Name, token.Staff.Email),
            new(ClaimTypes.Role, token.Staff.Role.ToString().ToUpperInvariant()),
            new("token_id", token.Id.ToString())
        ];
        var principal = new ClaimsPrincipal(new ClaimsIdentity(claims,
                                                               SchemeName));
        var ticket = new AuthenticationTicket(principal, SchemeName);
        return AuthenticateResult.Success(ticket);
    }

    protected override Task HandleChallengeAsync(AuthenticationProperties p) =>
        Json(401, "Unauthenticated.");

    protected override Task HandleForbiddenAsync(AuthenticationProperties p) =>
        Json(403, "This action is unauthorized.");
```

Kode 16.11 (Lanjutan) *BearerTokenHandler.cs: dari token bearer ke staf yang sudah login, dengan 401 dan 403 milik kontrak.*

```
    private Task Json(int status, string message)
    {
        Response.StatusCode = status;
        return Response.WriteAsJsonAsync(new { message });
    }
}
```

Kode 16.12 `TokenService.cs`: token bergaya Sanctum, di-*hash* dengan SHA-256 dan dibandingkan dalam waktu konstan.

```
using System.Security.Cryptography;
using System.Text;
using LaundryCapstone.Data;
using LaundryCapstone.Domain;
using Microsoft.AspNetCore.Identity;

namespace LaundryCapstone.Application;

/// <summary>
/// Token API gaya Laravel Sanctum: "id|secret" ditampilkan sekali,
/// tabel hanya menyimpan SHA-256 dari rahasianya.
/// </summary>
public sealed class TokenService(ILaundryStore store,
    IPasswordHasher<Staff> hasher, TimeProvider clock)
{
    public async Task<(string Token, Staff Staff)> IssueAsync(string email,
        string password, string device)
    {
        var staff = await store.FindStaffAsync(email);
        if (staff is null || hasher.VerifyHashedPassword(staff,
                staff.PasswordHash, password) == PasswordVerificationResult.Failed)
            throw new RuleException("email",
                "These credentials do not match our records.");
        string secret = Convert.ToBase64String(RandomNumberGenerator.GetBytes(30))
            .Replace('+', '-').Replace('/', '_');          // 40 karakter
        var token = new ApiToken
        {
            StaffId = staff.Id, Device = device, TokenHash = Sha256(secret),
            CreatedAt = clock.GetLocalNow().DateTime
        };
        store.Add(token);
        await store.SaveChangesAsync();
        return ($"{token.Id}|{secret}", staff);
    }

    /// <summary>Token di balik "id|secret", bila rahasianya cocok.</summary>
    public async Task<ApiToken?> FindAsync(string plain)
    {
        int bar = plain.IndexOf('|');
        if (bar < 1 || !long.TryParse(plain[..bar], out long id)) return null;
        var token = await store.FindTokenAsync(id);
        if (token is null || !CryptographicOperations.FixedTimeEquals(
                Encoding.ASCII.GetBytes(token.TokenHash),
                Encoding.ASCII.GetBytes(Sha256(plain[(bar + 1)..]))))
            return null;
        token.LastUsedAt = clock.GetLocalNow().DateTime;
```

Kode 16.12 (Lanjutan) *TokenService.cs: token bergaya Sanctum, di-hash dengan SHA-256 dan dibandingkan dalam waktu konstan.*

```
        await store.SaveChangesAsync();
        return token;
    }

    public async Task RevokeAsync(long id)
    {
        if (await store.FindTokenAsync(id) is { } token)
        {
            store.Remove(token);
            await store.SaveChangesAsync();
        }
    }

    private static string Sha256(string text) => Convert.ToHexStringLower(
        SHA256.HashData(Encoding.UTF8.GetBytes(text)));
}
```

Inilah `BearerTokenFilter` dan `TokenService` Bab 14 dalam C#. `AuthenticateResult.NoResult()` membiarkan permintaan tanpa token tetap anonim; *fallback policy* di `Program.cs`, "wajib pengguna terautentikasi", lalu menolaknya di mana pun tidak ada `[AllowAnonymous]` yang menyatakan lain, yang setara dengan `anyRequest().authenticated()` di Spring. `HandleChallengeAsync` dan `HandleForbiddenAsync` menulis badan 401 dan 403 milik kontrak. Kata sandi diperiksa dengan `PasswordHasher<Staff>`, *hasher* ASP.NET Core Identity, yang merupakan bagian *shared framework* dan tidak memerlukan paket; ia menyimpan PBKDF2 dengan HMAC-SHA512 dan 100.000 iterasi [4]. **Kode 16.13** merangkai semuanya.

Kode 16.13 `Program.cs`: penyimpanan, JSON, galat, sekuriti, dan data awal, secara berurutan.

```
using System.Text.Json;
using LaundryCapstone.Api;
using LaundryCapstone.Application;
using LaundryCapstone.Data;
using LaundryCapstone.Domain;
using LaundryCapstone.Security;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
#if USE_EF
using LaundryCapstone.Data.Ef;
using Microsoft.EntityFrameworkCore;
#endif

var builder = WebApplication.CreateBuilder(args);

// ---- penyimpanan: SQLite lewat EF Core, atau memori tanpa NuGet ----
#if USE_EF
builder.Services.AddDbContext<LaundryDbContext>(o =>
    o.UseSqlite(builder.Configuration.GetConnectionString("Laundry")));
builder.Services.AddScoped<ILaundryStore, EfLaundryStore>();
#else
builder.Services.AddSingleton<ILaundryStore, MemoryLaundryStore>();
#endif
builder.Services.AddSingleton(TimeProvider.System);
builder.Services.AddScoped<LaundryService>();
builder.Services.AddScoped<TokenService>();
builder.Services.AddSingleton<IPasswordHasher<Staff>, PasswordHasher<Staff>>();

// ---- MVC + JSON snake_case; masukan salah kita jawab sendiri (422) ----
builder.Services.AddControllersWithViews()
    .AddJsonOptions(o => o.JsonSerializerOptions.PropertyNamingPolicy =
        JsonNamingPolicy.SnakeCaseLower)
    .ConfigureApiBehaviorOptions(o => o.SuppressModelStateInvalidFilter = true);
builder.Services.AddExceptionHandler<ApiExceptionHandler>();

// ---- keamanan: token bearer, wajib login kecuali ditandai lain ----
builder.Services.AddAuthentication(BearerTokenHandler.SchemeName)
    .AddScheme<AuthenticationSchemeOptions, BearerTokenHandler>(
        BearerTokenHandler.SchemeName, null);
builder.Services.AddAuthorizationBuilder()
    .SetFallbackPolicy(new AuthorizationPolicyBuilder()
        .RequireAuthenticatedUser().Build());

var app = builder.Build();
```

Kode 16.13 (Lanjutan) *Program.cs: penyimpanan, JSON, galat, sekuriti, dan data awal, secara berurutan.*

```
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    await Seeder.SeedAsync(services.GetRequiredService<ILaundryStore>(),
        services.GetRequiredService<IPasswordHasher<Staff>>(),
        services.GetRequiredService<LaundryService>());
}

app.UseExceptionHandler(_ => { });
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();
app.MapGet("/", () => Results.Redirect("/board")).AllowAnonymous();

app.Run();
```

Dibaca dari atas ke bawah, `Program.cs`, satu-satunya berkas awal dalam model *minimal hosting* ASP.NET Core [5], adalah daftar setiap keputusan *capstone*: *store* mana, JSON snake_case, jawaban 422 milik kita sendiri, penangan eksepsi kita, skema *bearer*, "wajib login kecuali ditandai lain". *Seeder* memuat daftar harga dari `data/Service.csv`, berkas yang sama dengan milik Spring, membuat dua akun staf dengan kata sandi yang di-*hash*, dan mendaftarkan tiga pelanggan demo melalui `LaundryService`, sehingga data demo pun lulus aturan service.

16.5.6 Menjalankannya

Di Windows, macOS, atau Linux dengan .NET 10 SDK, buka `Chapter16/aspnet/LaundryCapstone` di terminal atau di VS Code dan nyalakan dengan SQLite:

```
dotnet run --urls http://localhost:5000
```

Tanpa akses ke NuGet, bangun versi memori sebagai gantinya:

```
dotnet run -p:UseEf=false --urls http://localhost:5000
```

Perintah kedua itulah yang dapat berjalan di mesin tempat buku ini ditulis (SDK 10.0.112, *runtime* 10.0.12). Dengan aplikasi yang mendengarkan, pemeriksa dijalankan di terminal kedua:

```
php ../../contract/check_api.php http://localhost:5000
```

dan mencetak:

```
PASS  a wrong password -> 422, errors.email
PASS  no token -> 401 with a message
PASS  POST /api/tokens -> 201, a token for a cashier
PASS  GET /api/services -> 200, S0001 first, 8000 per kg
PASS  POST /api/customers -> 201, phone kept as digits only
PASS  the same phone again -> 422, errors.phone
PASS  POST /api/orders -> 201, total 34000 (3 kg + pickup), queued
PASS  times are YYYY-MM-DDTHH:MM:SS, finished_at is null
PASS  a bad order -> 422, all of its errors at once
PASS  queued -> departed skips a step: 422, errors.status
PASS  queued -> finished: 200, finished_at set
PASS  finished -> departed before payment: 422, errors.status
PASS  paying more than the balance -> 422, errors.amount
PASS  paying the balance -> 201, balance 0, payments[0].method qris
PASS  finished -> departed after payment: 200
PASS  a departed order is closed: 409 with a message
PASS  GET /api/orders?status=departed -> 200, data[] and meta
PASS  GET /api/orders/999999 -> 404 with a message
PASS  GET /api/reports/daily -> 200, the counts add up
PASS  a cashier deletes a customer -> 403 with a message
PASS  the owner logs in -> 201, role owner
PASS  the owner deletes a customer with orders -> 409
PASS  the owner deletes a customer without orders -> 204
PASS  DELETE /api/tokens/current -> 204
PASS  the old token no longer works -> 401

25 passed, 0 failed
```

Run kedua terhadap proses yang sama juga lulus 25 dari 25, yang menunjukkan bahwa nomor telepon acak membuat pemeriksa dapat diulang. Pemeriksa milik Bab 12 sendiri, bila diberi token, lulus 16 pemeriksaannya terhadap server yang sama. Beberapa pesanan lalu diterima dengan tangan, dan dua jawaban disimpan apa adanya. Pesanan 4,5 kg Sari Wulandari, dibaca kembali dengan `GET /api/orders/5`:


```
{
    "data": {
        "id": 5,
        "customer": {
            "id": 1,
            "name": "Sari Wulandari",
            "phone": "081200000101",
            "address": "Jl. Kertajaya 12, Surabaya"
        },
        "service": {
            "id": "S0001",
            "name": "Wash, dry & fold",
            "unit": "kg",
            "price": 8000
        },
        "quantity": 4.5,
        "pickup": false,
        "delivery_address": null,
        "total": 36000,
        "paid": 0,
        "balance": 36000,
        "status": "queued",
        "note": null,
        "received_at": "2026-09-25T16:10:08",
        "finished_at": null,
        "departed_at": null,
        "payments": []
    }
}
```

dan penolakan untuk menyerahkan pesanan bed cover Rp 80.000 yang belum dibayar (dua buah ditambah biaya antar-jemput, baris keenam **Tabel 16.2**):

```
HTTP/1.1 422 Unprocessable Entity
Content-Type: application/json; charset=utf-8

{
    "message": "The given data was invalid.",
    "errors": {
        "status": [
            "Collect Rp 80.000 before handing over the laundry."
        ]
    }
}
```

Yang *tidak* dijalankan di sana adalah *build* EF Core, karena paketnya tidak dapat diunduh. Kodenya adalah antarmuka yang sama dengan kueri LINQ, dan README meminta Anda menjalankan pemeriksa terhadapnya lebih dulu.

16.5.7 Papan konter

Papan ini adalah halaman Razor yang boleh dilihat siapa pun di toko (**Kode 16.14** dan **Kode 16.15**).

Kode 16.14 `BoardController.cs`: antrean dan pesanan yang sudah selesai, tanpa login.

```
using LaundryCapstone.Application;
using LaundryCapstone.Domain;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

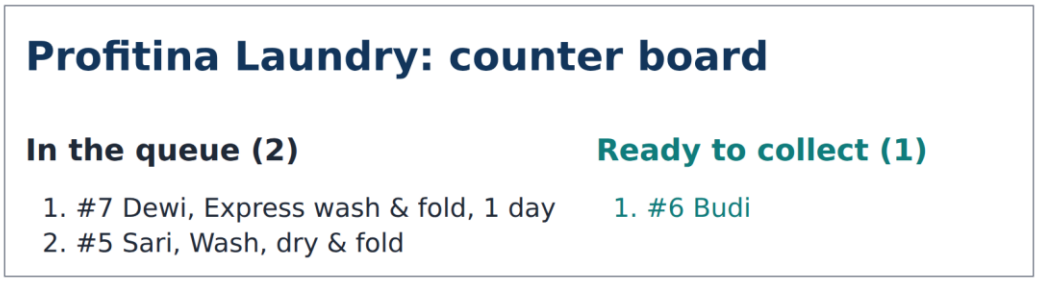
namespace LaundryCapstone.Controllers;

/// <summary>
/// Papan konter yang menghadap pelanggan: hanya nomor pesanan dan nama
/// depan, sehingga boleh ditampilkan tanpa login.
/// </summary>
[Route("board"), AllowAnonymous]
public class BoardController(LaundryService laundry) : Controller
{
    [HttpGet("")]
    public async Task<IActionResult> Index()
    {
        var (queued, _) =
            await laundry.OrderPageAsync(OrderStatus.Queued, 1, 100);
        var (finished, _) =
            await laundry.OrderPageAsync(OrderStatus.Finished, 1, 100);
        return View((queued, finished));
    }
}
```

Kode 16.15 `Views/Board/Index.cshtml`: nomor pesanan dan nama depan, disegarkan setiap 30 detik.

```
@model (List<Order> Queued, List<Order> Finished)
@{
    // "Sari Wulandari" -> "Sari": layar publik hanya menampilkan nama depan
    static string First(string name) => name.Split(' ')[0];
}
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta http-equiv="refresh" content="30">
  <title>Counter board | Profitina Laundry</title>
  <style>
    body { font-family: system-ui, sans-serif; margin: 1rem; color: #1f2937; }
    h1 { color: #12355B; }
    .cols { display: flex; gap: 2rem; flex-wrap: wrap; }
    li { font-size: 1.3rem; margin: .3rem 0; }
    .ready { color: #0F7C7C; }
  </style>
</head>
<body>
  <h1>Profitina Laundry: counter board</h1>
  <div class="cols">
    <section>
      <h2>In the queue (@Model.Queued.Count)</h2>
      <ol>
        @foreach (var o in Model.Queued)
        {
          <li>#@o.Id @First(o.Customer.Name), @o.Service.Name</li>
        }
      </ol>
    </section>
    <section class="ready">
      <h2>Ready to collect (@Model.Finished.Count)</h2>
      <ol>
        @foreach (var o in Model.Finished)
        {
          <li>#@o.Id @First(o.Customer.Name)</li>
        }
      </ol>
    </section>
  </div>
</body>
</html>
```

`[AllowAnonymous]` membuka satu controller ini dalam aplikasi yang tertutup secara bawaan. Karena papan ini tergantung di tempat setiap pelanggan dapat membacanya, papan hanya menampilkan nomor pesanan dan nama *depan*: tanpa nama belakang, tanpa telepon, tanpa alamat. Inilah *minimisasi data*: layar publik menampilkan apa yang diperlukannya dan tidak lebih. Halaman memuat ulang dirinya setiap 30 detik dengan `meta refresh`, yang tidak memerlukan JavaScript. **Gambar 16.3** adalah papan dari *capstone* yang berjalan.



Gambar 16.3 Papan konter *capstone* ASP.NET Core saat berjalan pada 25 September 2026: dua pesanan dalam antrean, satu siap diambil.

16.5.8 Penyelesaian keliru yang menggoda

Versi pertama *record* permintaan tampak seperti ini:

```
public sealed record OrderRequest(
    [property: Required] long? CustomerId, ...
```

Target `property:` tampak benar: ia menaruh atribut pada properti yang dibangkitkan, tempat atribut validasi biasanya tinggal. Proyeknya dibangun tanpa satu peringatan pun. Tetapi permintaan pertama ke `POST /api/orders` gagal dengan `InvalidOperationException`: ASP.NET Core menolak atribut validasi pada properti *record* posisional dan bersikeras agar atribut itu ditaruh pada parameter konstruktor, karena ia membuat *record* melalui konstruktornya [3]. Run pertama pemeriksa gagal di 14 pemeriksaannya karena satu alasan ini. Perbaikannya adalah menghapus `property:`, seperti diperlihatkan **Kode 16.7**.

Kejutan kedua datang tepat sesudahnya. Dengan *record* yang sudah dibetulkan, pesanan yang salah dijawab dengan `errors` berkunci `CustomerId` dan `Quantity`: `ModelState` menamai field menurut properti C#, atau menurut jalur JSON (`$.customer_id`) ketika JSON-nya sendiri tidak dapat dibaca, dan opsi JSON snake_case tidak berlaku bagi kunci-kunci itu. Pemeriksa, yang mengharapkan `customer_id`, langsung melaporkannya; `ApiErrors.Snake` kini mengubah setiap kunci. Kedua kesalahan itu lolos kompilasi dengan bersih, tampak benar saat ditinjau, dan hanya ditemukan karena sebuah program memeriksa jawaban field demi field. Itulah argumen terkuat yang dapat diajukan bab ini untuk kontrak dalam bentuk kode.

16.6 Soal 16.4: Capstone Laravel

16.6.1 Soalnya

Lengkapi proyek Laravel Bab 10 agar lulus 25 pemeriksaan yang sama dan punya papan yang sama. Bab 10 sudah punya token Sanctum, peran `owner` dan `cashier`, dan Bab 12 sudah membuat API-nya cocok dengan kontrak; yang kurang adalah hapus oleh pemilik dan papannya.

16.6.2 Merancangnya di atas kertas

Konvensi Laravel menentukan sebagian besar desainnya. *Route-model binding* mengubah `{customer}` di URL menjadi model `Customer`, atau 404. Sebuah *gate* bernama `delete-customers` menyatakan aturannya, dan *middleware* `can:` menerapkannya pada rute, menjawab 403 dengan "This action is unauthorized." milik Laravel sendiri. Papannya adalah controller *invokable* dengan tampilan Blade. Berkas-berkas di `Chapter16/laravel` disalin menimpa proyek Bab 10, yang sudah memuat berkas-berkas Bab 12; tidak diperlukan migrasi, karena tidak ada hal baru yang disimpan.

16.6.3 Penyelesaiannya

Kode 16.16 adalah berkas rute API dengan satu rute baru.

Kode 16.16 `routes/api.php`: rute `DELETE` baru, hanya untuk pemilik.

```php
<?php

use App\Http\Controllers\Api\CustomerController;
use App\Http\Controllers\Api\OrderController;
use App\Http\Controllers\Api\PaymentController;
use App\Http\Controllers\Api\ReportController;
use App\Http\Controllers\Api\ServiceController;
use App\Http\Controllers\Api\TokenController;
use Illuminate\Support\Facades\Route;

// Setiap URL di sini diawali /api (Laravel menambahkan prefiksnya).
Route::post('tokens', [TokenController::class, 'store'])
    ->middleware('throttle:5,1');

Route::middleware('auth:sanctum')->group(function () {
    Route::delete('tokens/current', [TokenController::class, 'destroy']);

    Route::get('services', [ServiceController::class, 'index']);
    Route::get('services/{service}', [ServiceController::class, 'show']);

    Route::get('customers', [CustomerController::class, 'index']);
    Route::post('customers', [CustomerController::class, 'store']);
    Route::delete('customers/{customer}',
        [CustomerController::class, 'destroy'])
        ->middleware('can:delete-customers');         // hanya pemilik

    Route::get('orders', [OrderController::class, 'index']);
    Route::post('orders', [OrderController::class, 'store']);
    Route::get('orders/{order}', [OrderController::class, 'show']);

    Route::middleware('order.open')->group(function () {
        Route::patch('orders/{order}/status',
            [OrderController::class, 'updateStatus']);
        Route::post('orders/{order}/payments',
            [PaymentController::class, 'store']);
    });

    Route::get('reports/daily', [ReportController::class, 'daily']);
});
```

Rute-rute itu juga memperlihatkan pintu-pintu lain Bab 10: `throttle:5,1` membatasi percobaan login menjadi lima per menit, `auth:sanctum` melindungi semua yang lain, dan `order.open` menjawab 409 untuk pesanan yang sudah departed sebelum controller berjalan. **Kode 16.17** mendefinisikan *gate*-nya, dan **Kode 16.18** aksinya.

Kode 16.17 `AppServiceProvider.php`: *gate* Bab 10 dan `delete-customers` yang baru.

```php
<?php

namespace App\Providers;

use App\Models\User;
use Illuminate\Support\Facades\Gate;
use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
    public function register(): void
    {
        //
    }

    public function boot(): void
    {
        // Hanya pemilik yang mengubah daftar harga.
        Gate::define('manage-prices', fn (User $user) => $user->isOwner());

        // Hanya pemilik yang menghapus pelanggan (Bab 16).
        Gate::define('delete-customers', fn (User $user) => $user->isOwner());
    }
}
```

Kode 16.18 `CustomerController.php`: `destroy` menolak pelanggan yang punya pesanan dengan 409.

```php
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use App\Http\Requests\StoreCustomerRequest;
use App\Http\Resources\CustomerResource;
use App\Models\Customer;
use Illuminate\Http\Request;

class CustomerController extends Controller
{
    // GET /api/customers?phone=0812...  (konter mencari orang lewat telepon)
    public function index(Request $request)
    {
        $phone = preg_replace('/[^0-9]/', '', (string) $request->query('phone'));
        $customers = Customer::query()
            ->when($phone !== '', fn ($q) => $q->where('phone', $phone))
            ->orderBy('name')
            ->paginate(20);

        return CustomerResource::collection($customers);
    }

    public function store(StoreCustomerRequest $request)
    {
        $customer = Customer::create($request->validated());

        return (new CustomerResource($customer))->response()
            ->setStatusCode(201);
    }

    // DELETE /api/customers/{customer}: hanya pemilik (routes/api.php).
    public function destroy(Customer $customer)
    {
        if ($customer->orders()->exists()) {
            abort(409, "{$customer->name} has orders and cannot be deleted.");
        }
        $customer->delete();

        return response()->noContent();
    }
}
```

`abort(409, ...)` menghasilkan `{"message": "..."}` untuk permintaan JSON tanpa kode tambahan apa pun, persis 409 milik kontrak. **Kode 16.19**, **Kode 16.20**, dan **Kode 16.21** menambahkan papannya.

Kode 16.19 `routes/web.php`: papan adalah rute publik.

```php
<?php

use App\Http\Controllers\Auth\LoginController;
use App\Http\Controllers\BoardController;
use App\Http\Controllers\CounterController;
use App\Http\Controllers\CustomerController;
use App\Http\Controllers\HomeController;
use App\Http\Controllers\OrderController;
use App\Http\Controllers\OutletController;
use App\Http\Controllers\PaymentController;
use App\Http\Controllers\PriceController;
use App\Http\Controllers\ReportController;
use App\Http\Controllers\ServiceController;
use Illuminate\Support\Facades\Route;

// Halaman publik: siapa pun boleh melihatnya.
Route::get('/', HomeController::class)->name('home');
Route::get('services', [ServiceController::class, 'index'])
    ->name('services.index');
Route::get('services/{service}', [ServiceController::class, 'show'])
    ->name('services.show');
Route::get('outlets', [OutletController::class, 'index'])
    ->name('outlets.index');
Route::get('board', BoardController::class)->name('board');   // Bab 16

// Masuk dan keluar.
Route::middleware('guest')->group(function () {
    Route::get('login', [LoginController::class, 'create'])->name('login');
    Route::post('login', [LoginController::class, 'store'])
        ->middleware('throttle:5,1');           // lima percobaan per menit
});
Route::post('logout', [LoginController::class, 'destroy'])
    ->middleware('auth')->name('logout');

// Halaman staf: hanya untuk staf yang sudah login.
Route::middleware('auth')->group(function () {
    Route::get('counter', CounterController::class)->name('counter');
    Route::resource('customers', CustomerController::class);
    Route::resource('orders', OrderController::class)
        ->only(['index', 'create', 'store', 'show']);
    Route::get('reports/daily', [ReportController::class, 'daily'])
        ->name('reports.daily');
```

Kode 16.19 (Lanjutan) *routes/web.php: papan adalah rute publik.*

```php
    Route::middleware('order.open')->group(function () {
        Route::post('orders/{order}/advance',
            [OrderController::class, 'advance'])->name('orders.advance');
        Route::post('orders/{order}/payments',
            [PaymentController::class, 'store'])
            ->name('orders.payments.store');
    });

    // Daftar harga: hanya pemilik.
    Route::resource('prices', PriceController::class)
        ->parameters(['prices' => 'service'])
        ->except(['show', 'destroy'])
        ->middleware('can:manage-prices');
});
```

Kode 16.20 `BoardController.php`: antrean dan pesanan yang sudah selesai.

```php
<?php

namespace App\Http\Controllers;

use App\Enums\OrderStatus;
use App\Models\Order;

// Papan konter yang menghadap pelanggan: hanya nomor pesanan dan nama
// depan, sehingga boleh ditampilkan tanpa login.
class BoardController extends Controller
{
    public function __invoke()
    {
        $load = fn (OrderStatus $s) => Order::with(['customer', 'service'])
            ->where('status', $s)->orderByDesc('received_at')
            ->orderByDesc('id')->limit(100)->get();

        return view('board', [
            'queued' => $load(OrderStatus::Queued),
            'finished' => $load(OrderStatus::Finished),
        ]);
    }
}
```

Kode 16.21 `board.blade.php`: papan yang sama dengan milik Razor, dalam Blade.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta http-equiv="refresh" content="30">
  <title>Counter board | Profitina Laundry</title>
  <style>
    body { font-family: system-ui, sans-serif; margin: 1rem; color: #1f2937; }
    h1 { color: #12355B; }
    .cols { display: flex; gap: 2rem; flex-wrap: wrap; }
    li { font-size: 1.3rem; margin: .3rem 0; }
    .ready { color: #0F7C7C; }
  </style>
</head>
<body>
  <h1>Profitina Laundry: counter board</h1>
  <div class="cols">
    <section>
      <h2>In the queue ({{ $queued->count() }})</h2>
      <ol>
        @foreach ($queued as $o)
          <li>#{{ $o->id }} {{ strtok($o->customer->name, ' ') }},
              {{ $o->service->name }}</li>
        @endforeach
      </ol>
    </section>
    <section class="ready">
      <h2>Ready to collect ({{ $finished->count() }})</h2>
      <ol>
        @foreach ($finished as $o)
          <li>#{{ $o->id }} {{ strtok($o->customer->name, ' ') }}</li>
        @endforeach
      </ol>
    </section>
  </div>
</body>
</html>
```

`strtok($name, ' ')` mengembalikan bagian sebelum spasi pertama, yaitu nama depan, dan `{{ }}` milik Blade meng-*escape*-nya. *Eager loading* `with(['customer', 'service'])` memuat baris-

baris terkait dalam dua kueri tambahan alih-alih dua per pesanan. **Kode 16.22** menguji perilaku yang baru.

Kode 16.22 `CapstoneTest.php`: penghapusan oleh kasir, dua penghapusan oleh pemilik, dan privasi papan.

```php
<?php

namespace Tests\Feature;

use App\Enums\Role;
use App\Models\Customer;
use App\Models\Order;
use App\Models\User;
use Database\Seeders\PriceListSeeder;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Laravel\Sanctum\Sanctum;
use Tests\TestCase;

// Tambahan Bab 16: hapus khusus pemilik dan papan konter publik.
class CapstoneTest extends TestCase
{
    use RefreshDatabase;

    protected function setUp(): void
    {
        parent::setUp();
        $this->seed(PriceListSeeder::class);        // layanan dan outlet
    }

    public function test_a_cashier_may_not_delete_a_customer(): void
    {
        Sanctum::actingAs(User::factory()->create());   // seorang kasir
        $customer = Customer::factory()->create();
        $this->deleteJson("/api/customers/{$customer->id}")
            ->assertForbidden()
            ->assertJson(['message' => 'This action is unauthorized.']);
        $this->assertModelExists($customer);
    }

    public function test_the_owner_deletes_only_customers_without_orders(): void
    {
        Sanctum::actingAs(User::factory()->create(['role' => Role::Owner]));
        $busy = Order::factory()->create()->customer;
        $this->deleteJson("/api/customers/{$busy->id}")->assertConflict();
        $idle = Customer::factory()->create();
        $this->deleteJson("/api/customers/{$idle->id}")->assertNoContent();
        $this->assertModelMissing($idle);
    }
```

Kode 16.22 (Lanjutan) *CapstoneTest.php: penghapusan oleh kasir, dua penghapusan oleh pemilik, dan privasi papan.*

```php
    public function test_the_board_shows_first_names_only(): void
    {
        $order = Order::factory()->create();
        $order->customer->update(['name' => 'Sari Wulandari']);
        $this->get('/board')->assertOk()
            ->assertSee('#' . $order->id . ' Sari')
            ->assertDontSee('Wulandari');
    }
}
```

`Sanctum::actingAs` memasukkan seorang pengguna untuk satu tes tanpa token, padanan Laravel untuk `@WithMockUser`. Tes ketiga layak disalin ke proyek mana pun yang punya layar publik: tes itu menegaskan apa yang harus tampil (`#12 Sari`) sekaligus apa yang tidak boleh (`Wulandari`).

16.6.4 Menjalankannya

Dengan PHP 8.3 atau yang lebih baru, Composer, dan proyek Bab 10 yang disiapkan seperti dijelaskan README-nya, salin berkas-berkas `Chapter16/laravel` menimpanya, lalu

```
php artisan migrate:fresh --seed
php artisan test
php artisan serve --port=8000
php ../contract/check_api.php http://localhost:8000 \
    cashier@laundry.test password owner@laundry.test password
```

(Di Windows PowerShell, tulis perintah terakhir dalam satu baris, tanpa garis miring terbalik.)

Perintah pertama membangun ulang basis data SQLite dengan data demo, yang kedua menjalankan tes Bab 10 dan `CapstoneTest`, yang ketiga menyalakan server, dan yang terakhir menjalankan pemeriksa dengan akun demo Laravel. Di mesin tempat buku ini ditulis, Composer tidak dapat mengunduh Laravel, sehingga tidak satu pun perintah ini dapat berjalan di sana. Setiap berkas PHP bab ini, termasuk pemeriksanya, diperiksa dengan `php -l`, dan pemeriksa dijalankan sungguhan terhadap server ASP.NET Core. Kode Laravel hanya memakai fitur yang didokumentasikan untuk Laravel 13 [6], [7].

16.6.5 Penyelesaian keliru yang menggoda

Menulis pemeriksaan peran di dalam controller memang menggoda: `if (! $request->user()->isOwner()) abort(403);`. Cara itu berjalan, tetapi menyembunyikan aturan di satu metode tempat pengembang berikutnya yang menambah fitur hapus untuk layanan tidak akan melihatnya, dan ia berjalan *setelah route-model binding*, sehingga kasir yang meminta pelanggan 999 mengetahui dari 404 bahwa pelanggan itu tidak ada. *Gate* dan *middleware* `can:` menaruh aturan di tempat rute dibaca, dan berjalan sebelum record dimuat.

16.7 Soal 16.5: Capstone Spring Boot

16.7.1 Soalnya

Lengkapi proyek Spring Boot Bab 14 dengan papan. Hapus oleh pemilik, token, dan galat kontrak sudah tersedia.

16.7.2 Penyelesaiannya

Kode 16.23 dan **Kode 16.24** adalah papannya.

Kode 16.23 `BoardController.java`: papan membaca pesanannya melalui `OrderService`.

```java
package com.profitina.laundry.web;

import com.profitina.laundry.domain.OrderStatus;
import com.profitina.laundry.service.OrderService;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

/**
 * Papan konter yang menghadap pelanggan: hanya nomor pesanan dan nama
 * depan, sehingga boleh ditampilkan tanpa login (SecurityConfig).
 */
@Controller
public class BoardController {

    private final OrderService orders;

    public BoardController(OrderService orders) {
        this.orders = orders;
    }

    @GetMapping("/board")
    public String board(Model model) {
        model.addAttribute("queued",
                orders.page(OrderStatus.QUEUED, 1, 100).getContent());
        model.addAttribute("finished",
                orders.page(OrderStatus.FINISHED, 1, 100).getContent());
        return "board";
    }
}
```

Kode 16.24 `board.html`: papan yang sama dalam Thymeleaf.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="utf-8">
  <meta http-equiv="refresh" content="30">
  <title>Counter board | Profitina Laundry</title>
  <style>
    body { font-family: system-ui, sans-serif; margin: 1rem; color: #1f2937; }
    h1 { color: #12355B; }
    .cols { display: flex; gap: 2rem; flex-wrap: wrap; }
    li { font-size: 1.3rem; margin: .3rem 0; }
    .ready { color: #0F7C7C; }
  </style>
</head>
<body>
  <h1>Profitina Laundry: counter board</h1>
  <div class="cols">
    <section>
      <h2 th:text="'In the queue (' + ${queued.size()} + ')'">In the queue</h2>
      <ol>
        <li th:each="o : ${queued}"
            th:text="'#' + ${o.id} + ' ' + ${o.customer.name.split(' ')[0]}
                     + ', ' + ${o.service.name()}">#1 Sari, Wash</li>
      </ol>
    </section>
    <section class="ready">
      <h2 th:text="'Ready to collect (' + ${finished.size()} + ')'">Ready</h2>
      <ol>
        <li th:each="o : ${finished}"
            th:text="'#' + ${o.id} + ' ' + ${o.customer.name.split(' ')[0]}">
          #2 Budi</li>
      </ol>
    </section>
  </div>
</body>
</html>
```

Rantai web Bab 14 harus meloloskan papan tanpa login, sehingga `SecurityConfig` mendapat `"/board"` dalam daftar halaman terbuka: `.requestMatchers("/login", "/board", "/css/**", "/error").permitAll()` [8]. `${o.customer.name.split(' ')[0]}` memanggil `String.split` milik Java dari templat, dan `th:text` meng-*escape* hasilnya. Controller memanggil metode `page` service yang sudah ada, sehingga papan memakai kueri yang sama, dengan *entity graph* yang sama, seperti halaman-halaman konter.

16.7.3 Menjalankannya

Dengan JDK 25 dan Maven di Windows, macOS, atau Linux, dari `Chapter16/spring`:

```
mvn test
mvn spring-boot:run
php ../contract/check_api.php http://localhost:8080
```

`mvn test` menjalankan 24 run tes Bab 14; aplikasi mendengarkan di port 8080 dengan SQLite dalam berkas `laundry.db`; pemeriksa memakai akun bawaan. Di mesin tempat buku ini ditulis, Maven tidak dapat mengunduh Spring, sehingga perintah-perintah ini tidak dapat berjalan di sana. Sebagai gantinya, setiap berkas sumber Java proyek ini, termasuk tesnya, dikompilasi dengan `javac` terhadap pengganti-pengganti kecil untuk tipe Spring, Jakarta, Hibernate, dan JUnit yang dipakainya, tanpa galat, dan aturannya dijalankan sungguhan (Soal 16.2). Pemeriksaan kompilasi terhadap pengganti menangkap nama, tipe, dan *import* yang salah; ia tidak dapat menangkap atribut anotasi yang salah atau kueri yang ditolak Hibernate saat menyala, itulah sebabnya `mvn test` adalah perintah pertama yang harus dijalankan.

16.8 Membandingkan Ketiga Penyelesaian

Tabel 16.3 menjajarkan ketiga *capstone*, fitur demi fitur.

Tabel 16.3 Ketiga *capstone* dibandingkan.

Fitur	ASP.NET Core 10	Laravel 13	Spring Boot 4.1
Login token	`TokenService` dan `AuthenticationHandler` sendiri	Sanctum, bawaan	`TokenService` dan filter sendiri
Badan 401 dan 403	ditulis di *handler*	bawaan Laravel	ditulis di rantai filter
"Wajib login kecuali ditandai"	*fallback policy*, `[AllowAnonymous]`	grup `auth:sanctum`	`anyRequest().authenticated()`, `permitAll()`
Hanya pemilik	`[Authorize(Roles = "OWNER")]`	*gate* dan *middleware* `can:`	aturan URL dan `@PreAuthorize`
422 dengan semua galat	matikan 400 otomatis, gabungkan `ModelState` dan aturan	Form Request, bawaan	`Validator` dan aturan digabung di controller
JSON snake_case	*naming policy*, ditambah konversi kunci galat	menurut konvensi	pengaturan Jackson
404 untuk record yang tidak ada	`NotFoundException` dan *handler*	*route-model binding*	`NotFoundException` dan *advice*
Papan, nama depan saja	controller `[AllowAnonymous]`, Razor	rute publik, Blade	`permitAll`, Thymeleaf
Penyimpanan	EF Core dan SQLite, atau memori	Eloquent dan SQLite	JPA, Hibernate, dan SQLite
Yang dijalankan di sini	dibangun dan dijalankan; 25/25 dan 16/16	`php -l`	dikompilasi terhadap pengganti; aturan dijalankan

Polanya jelas. Laravel memenuhi kontrak dengan kode paling sedikit, karena kontrak itu *adalah* bentuk bawaan Laravel: 422 dengan `errors`, pesan 401 dan 403, snake_case, amplop `data`. Dua lainnya mencapai jawaban yang sama dengan mengubah bawaan, dan setiap perubahan adalah tempat kesalahan dapat bersembunyi, seperti diperlihatkan Subbab 16.5.8. Seandainya kontrak mengikuti bawaan ASP.NET Core (400 dengan *problem details*, camelCase), timbangannya akan condong ke arah sebaliknya. Pelajaran bagi proyek nyata adalah memilih kontrak demi kliennya lebih dulu, lalu mengharapkan kerangka kerja yang bawaannya tidak cocok memerlukan lapisan penyesuaian yang kecil dan teruji. Perbandingan ukuran fitur di Bab 15 (Subbab 15.5) menghitung baris yang dihabiskan kedua fitur bab ini di setiap kerangka kerja; perbedaannya kecil.

Yang tidak diperlihatkan tabel adalah betapa miripnya ketiga service. `LaundryService.CheckAsync`, `OrderService.check`, dan aturan di dalam Form Request serta *action* Laravel mengajukan empat pertanyaan yang sama dalam urutan yang sama. Aturan, kontrak, dan pemeriksa adalah bagian sistem yang awet; kerangka kerja adalah bagian yang dapat diganti.

Seberapa dapat diganti, dalam praktik? Memindahkan sebuah outlet dari satu versi ke versi lain berarti memindahkan datanya, bukan hanya kodenya. Keempat entitas punya kolom yang sama di ketiga versi, dengan satu perbedaan yang ditunjukkan **Gambar 16.2**: Laravel menyimpan akunnya di tabel `users` dan tokennya di `personal_access_tokens` milik Sanctum, sedangkan dua lainnya memakai `staff` dan `api_tokens`. Karena itu perpindahan menyalin layanan, pelanggan,

pesanan, dan pembayaran tabel demi tabel, dengan mempertahankan pengenalnya agar nomor pesanan yang tercetak di nota tetap berlaku, dan meminta staf menetapkan kata sandinya lagi, karena ketiga kerangka kerja menyimpan *hash* kata sandi dalam format yang berbeda. Token sama sekali tidak disalin: setiap ponsel cukup login sekali lagi. Pemeriksa, yang dijalankan terhadap server baru sebelum outlet dibuka, adalah langkah terakhir perpindahan semacam itu.

16.9 Keputusan Bu Wati

Dengan tiga versi yang menaati kontrak yang sama, pilihan versi mana yang menjalankan outlet-outlet tidak lagi memengaruhi aplikasi Raka, dan pilihan itu dapat dibuat dengan prosedur Subbab 15.9 alih-alih karena kebiasaan.

Batasan yang tidak dapat bergeser. Toko sudah menyewa server virtual kecil untuk situs webnya, dengan memori 2 GB. Ketiganya dapat berjalan di sana; untuk Laravel, *hosting* PHP saja pun cukup. Tidak ada apa pun di toko yang harus tersambung ke platform Java atau Windows yang sudah ada.

Apa yang diketahui orang-orangnya. Dimas, yang merawat sistem selama liburannya, paling menguasai PHP dan telah memakai Laravel sejak Bab 7. Raka menulis aplikasi ponsel dan tidak menyentuh server. Tidak seorang pun di keluarga itu bekerja dengan C# atau Java setiap hari.

Bebannya. Tiga outlet menerima sekitar 150 pesanan sehari seluruhnya. Setiap pesanan menimbulkan barangkali selusin permintaan sepanjang hidupnya (pembuatan, pencarian, dua langkah status, satu atau dua pembayaran, belum termasuk muat ulang papan), sehingga jam tersibuk jauh di bawah satu permintaan per detik. Menurut hukum Little, dengan 30 ms per permintaan, rata-rata kurang dari 0,03 permintaan sedang berjalan. Kecepatan gugur dari pertimbangan.

Kalendernya. Laravel 13 menerima perbaikan sekuriti sampai 17 Maret 2028, .NET 10 sampai 14 November 2028, dan Spring Boot 4.1 perbaikan *open source* sampai 31 Juli 2027 (Subbab 15.7). Masing-masing cukup panjang untuk tahun pertama.

Bukti dari bab ini. Versi ASP.NET Core adalah satu-satunya yang lulus pemeriksa di mesin tempat buku ini ditulis; versi Laravel adalah yang bawaannya cocok dengan kontrak dan yang kodenya untuk fitur-fitur baru paling pendek; versi Spring Boot punya halaman konter paling lengkap (Bab 13) dan tes paling kaya (Bab 14).

Prosedurnya menunjuk Laravel: orang yang akan merawatnya menguasainya, cocok dengan *hosting*-nya, dan memerlukan penyesuaian paling sedikit terhadap kontrak. Keputusan itu dituliskan sebagai *architecture decision record* di proyek, dengan syarat kapan keputusan itu ditinjau ulang: bila toko suatu hari tumbuh menjadi banyak outlet dengan server pusat, atau merekrut pengembang yang bekerja dengan C# atau Java, dua versi lainnya sudah siap, dan pemeriksa akan mengatakan dalam hitungan detik apakah perpindahan merusak sesuatu. Kalimat terakhir itulah hasil sesungguhnya dari *capstone* ini. Keputusan dapat diambil atas dasar yang penting bagi toko, karena kontrak telah membuat ketiga versi dapat saling menggantikan bagi siapa pun di luar server.

16.10 Memakainya Sungguhan

Sebelum versi mana pun dari ketiganya menjalankan sebuah outlet, masih ada daftar pekerjaan pendek, dan daftarnya sama apa pun versi yang dipilih Bu Wati.

- **Jalankan *build* yang belum teruji.** Jalankan `mvn test`, tes Laravel, dan *build* EF Core ASP.NET Core, lalu pemeriksa terhadap masing-masing. Sejauh ini baru *build* memori ASP.NET Core yang lulus.
- **Ganti kata sandi demo**, dan buat satu akun per kasir agar catatan memperlihatkan siapa mengerjakan apa.
- **Layani lewat HTTPS** di belakang *reverse proxy*, dan cadangkan berkas SQLite setiap malam ke mesin lain; uji pemulihannya sekali.
- **Migrasi alih-alih pembaruan skema otomatis** sebelum data sungguhan pertama: migrasi EF Core, migrasi Laravel (sudah tersedia), atau Flyway.
- **Jalankan pemeriksa setelah setiap *deployment*.** Hanya dua detik, dan membuktikan bahwa aplikasi Raka akan tetap berjalan.

Satu hal lagi sering terlupa: mengetahui bahwa sistem sedang bermasalah sebelum pelanggan memberi tahu. Ketiga kerangka kerja menulis log setiap permintaan dan setiap eksepsi yang tidak tertangani; arahkan log itu ke berkas yang diputar setiap hari, dan periksa sekali sehari apakah ada jawaban 500. Satu jawaban 500 adalah *bug* yang belum dikenal, bukan nasib buruk. Pemeriksa yang sama dapat dijadwalkan berjalan setiap pagi sebelum toko buka, terhadap server yang hidup, dengan pelanggan uji yang kemudian dihapus pemilik; bila ada yang gagal, pesan singkat ke ponsel Dimas cukup untuk memperbaikinya sebelum antrean pertama datang.

Pendekatan ini berlaku untuk usaha lain tanpa perubahan: kontrak yang ditulis sebagai pemeriksa, aturan dalam kode biasa dengan contoh yang dikerjakan tangan, controller yang tipis, dan satu program yang memutuskan apakah sebuah implementasi dapat diterima.

16.11 Profitina dalam Praktik

Profitina menjalankan disiplin yang sama dengan API-nya sendiri. *Endpoint* publiknya dijelaskan oleh dokumen OpenAPI, dan rangkaian tes memeriksa kode status dan field setiap *endpoint* terhadap dokumen itu, sehingga server MCP, dasbor web, dan skrip pelanggan dapat mengandalkan jawaban yang sama. Sebelum setiap rilis seluruh rangkaian tes, jauh lebih dari seribu tes, harus lulus, dan sekumpulan pemeriksaan yang lebih kecil berjalan terhadap sistem yang hidup setelah *deployment*, dengan semangat yang sama seperti `check_api.php`. Ketika versi baru sebuah pustaka mengubah bawaan, seperti format tanggal atau nama field galat, pemeriksaan itulah yang menyadarinya, jauh sebelum pengguna menyadarinya.

16.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Kontrak hanya dalam kalimat.** Pembaca yang berbeda mengimplementasikan kontrak yang berbeda. Tulis sebagai program yang memeriksa nama dan nilai field.
- **Pemeriksa yang tidak dapat berjalan dua kali.** Nomor telepon atau pengenal yang tetap gagal pada run kedua. Bangkitkan apa yang harus unik.
- **Uang dalam *floating point*.** Hitung dalam satuan utuh (perseratus kilogram, rupiah utuh) dengan bilangan bulat, seperti di Soal 16.2.
- **`<=` padahal aturannya berkata "di bawah".** Kerjakan batasnya dengan tangan: lima kemeja seharga Rp 20.000 tidak boleh membayar biaya antar-jemput.
- **`[property:]` pada atribut validasi *record* posisional** di ASP.NET Core. Taruh pada parameternya.

- **Memercayai *naming policy* untuk kunci galat.** Kunci `ModelState` tidak diubah ke snake_case; ubah sendiri.
- **Data pribadi di layar publik.** Tampilkan nomor pesanan dan nama depan saja, dan uji bahwa sisanya tidak muncul.
- **Pemeriksaan peran di dalam metode controller.** Nyatakan di rute (`can:`, `[Authorize]`, aturan URL) dan, untuk operasi penting, sekali lagi di service.
- **Mengklaim lebih dari yang dijalankan.** Katakan apa yang dibangun, apa yang dijalankan, dan apa yang hanya dikompilasi.

16.13 Ringkasan Bab

- Satu kontrak (**Tabel 16.1**) dan satu pemeriksa (**Gambar 16.1**, 25 pemeriksaan) mendefinisikan API Profitina Laundry untuk ketiga kerangka kerja; pemeriksanya PHP biasa, dapat diulang, dan membaca field yang dibaca aplikasi.
- Aturannya dikerjakan dengan tangan (Persamaan (16.1), Persamaan (16.2), **Tabel 16.2**) dan menghasilkan total yang sama di Java dan C#, dengan aritmetika bilangan bulat dalam perseratus.
- *Capstone* ASP.NET Core dibangun dari nol dengan lapisan service, antarmuka penyimpanan dengan implementasi EF Core dan memori, login token, dan galat kontrak; *build* memorinya lulus 25 dari 25 pemeriksaan dan 16 pemeriksaan Bab 12.
- Dua kesalahan yang lolos kompilasi dengan bersih, `[property:]` pada parameter *record* dan kunci galat PascalCase, hanya ditemukan oleh pemeriksa.
- *Capstone* Laravel memerlukan satu rute, satu *gate*, satu aksi controller, dan papan, karena kontraknya adalah bawaan Laravel; *capstone* Spring Boot memerlukan papan dan satu aturan sekuriti.
- Setiap hasil ditempatkan pada salah satu dari empat tingkat bukti (diperiksa sintaksnya, dikompilasi, dijalankan, diperiksa terhadap kontrak), dan hanya tingkat terakhir yang membuktikan bahwa aplikasi ponsel akan bekerja; perpindahan antarversi memindahkan data, bukan hanya kode, dan diakhiri dengan pemeriksa.
- Papan publik hanya menampilkan nomor pesanan dan nama depan (**Gambar 16.3**), dalam Razor, Blade, dan Thymeleaf.
- Perbandingannya (**Tabel 16.3**) memperlihatkan bahwa kerangka kerja yang bawaannya cocok dengan kontrak memerlukan penyesuaian paling sedikit, dan bahwa aturan, kontrak, dan pemeriksa adalah bagian sistem yang awet.

16.14 Soal Latihan

1. Memakai Persamaan (16.1), hitung dengan tangan total dari: (a) 2,99 kg setrika saja seharga Rp 6.000 dengan antar-jemput; (b) 12,5 kg cuci-kering-lipat seharga Rp 8.000 dengan antar-jemput; (c) 3 selimut seharga Rp 25.000 dengan antar-jemput; (d) 0,5 pasang sepatu seharga Rp 45.000.
2. Tambahkan dua pemeriksaan ke `check_api.php`: `GET /api/customers?phone=` dengan telepon pelanggan baru, ditulis dengan tanda hubung, menemukan tepat pelanggan itu; dan `GET /api/services/S9999` menjawab 404 dengan pesan.
3. Pemeriksa mengirim `outlet_id`, yang hanya diperlukan Laravel. Apa yang terjadi bila versi *capstone* ASP.NET Core kelak menolak field yang tidak dikenal? Di mana Anda akan mengubah desainnya, dan mengapa?

4. Bu Wati ingin papan menampilkan, di samping setiap pesanan yang sudah selesai, sudah berapa lama pesanan itu menunggu, misalnya “2 h”. Jelaskan perubahannya di ketiga kerangka kerja, dan sebutkan di mana perhitungannya semestinya berada.
5. Seorang rekan mengusulkan menampilkan empat angka terakhir telepon di papan, “agar pelanggan dengan nama depan yang sama dapat membedakan pesanannya”. Beri argumen mendukung atau menentang, dan usulkan alternatif.
6. Tulis padanan Spring Boot dari `CapstoneTest.test_the_board_shows_first_names_only` sebagai `@SpringBootTest` dengan `MockMvc`.
7. *Build* memori *capstone* ASP.NET Core lulus pemeriksa, tetapi *build* EF Core tidak dijalankan. Sebutkan tiga jenis *bug* yang hanya dapat ada di *build* EF Core, dan bagaimana pemeriksa akan mengungkap masing-masing.

16.15 Penyelesaian Soal Latihan

1. (a) $h = 299$, per kg, ditagih $\max(299,300) = 300$; subtotal $\lfloor (6000 \cdot 300 + 50)/100 \rfloor = 18000$; di bawah Rp 100.000 dengan antar-jemput: $T = 28000$. (b) $h = 1250$, ditagih 1250, subtotal 100000; tidak di bawah Rp 100.000, sehingga $T = 100000$. (c) Satuan: $h = 300$, ditagih 300, subtotal 75000, dengan antar-jemput $T = 85000$. (d) Pasang: $h = 50$ dibulatkan ke atas menjadi 100, subtotal 45000, $T = 45000$.

2. Setelah pelanggan dibuat:

```
[$s, $j] = call('GET', '/api/customers?phone='
    . substr($phone, 0, 4) . '-' . substr($phone, 4));
check('GET /api/customers?phone= finds the new customer',
    $s === 200 && count($j['data'] ?? []) === 1
    && ($j['data'][0]['id'] ?? 0) === ($customer['id'] ?? -1));
[$s, $j] = call('GET', '/api/services/S9999');
check('GET /api/services/S9999 -> 404 with a message',
    $s === 404 && is_string($j['message'] ?? null));
```

Ketiga server memangkas telepon dalam *query* menjadi angka sebelum mencari, sehingga tanda hubungnya tidak berpengaruh.

3. Pemeriksaan pesanan akan gagal dengan 422 (atau 400) pada `outlet_id`, dan aplikasi Raka, yang mungkin juga mengirimnya, akan rusak dengan cara yang sama. Dua desain masuk akal: jadikan `outlet_id` bagian kontrak untuk ketiga server (toko punya tiga outlet, jadi pesanan semestinya mencatat di mana ia diterima), atau buang dari pemeriksa dan dari *request* Laravel. Yang pertama lebih baik, karena mencerminkan usahanya; bagaimanapun, keputusan itu tempatnya di kontrak, bukan di satu server.

4. Tambahkan waktu tunggu pada apa yang ditampilkan papan, dihitung dari `finished_at` dan waktu kini menurut jam toko: di controller, atau lebih baik di metode kecil service (`waitingSince`) agar dapat diuji dengan jam yang tetap. Tampilan-tampilannya mencetaknya: `@(...)` di Razor, `{{ }}` di Blade (`diffForHumans()` milik Laravel mengerjakan pemformatannya), `th:text` di Thymeleaf. Perhitungannya tempatnya di service atau di pembantu kecil, bukan di templat, karena bergantung pada jam.

5. Menentang: empat angka telepon, di samping nama depan dan waktu seorang pelanggan datang ke toko, membantu siapa pun yang ingin mengenali atau menghubungi orang asing, dan manfaatnya kecil. Nomor pesanan sudah membedakan pesanan; mencetaknya besar-besar di nota, dan memanggil nomor, memecahkan masalah itu tanpa menampilkan data pribadi apa pun.

6. Di `src/test/java/com/profitina/laundry/web/BoardTest.java`:

```
@SpringBootTest
@AutoConfigureMockMvc
class BoardTest {

    @Autowired
    MockMvc mvc;

    @Test
    void theBoardShowsFirstNamesOnly() throws Exception {
        mvc.perform(get("/board"))                      // no login needed
           .andExpect(status().isOk())
           .andExpect(content().string(containsString("Sari")))
           .andExpect(content().string(
                   not(containsString("Wulandari"))));
    }
}
```

`DemoData` memberi Sari Wulandari pesanan yang sudah selesai, sehingga nama depannya muncul dalam daftar pesanan yang siap diambil. `containsString` dan `not` berasal dari Hamcrest, yang disertakan *starter* tes.

7. (a) Kueri yang tidak dapat diterjemahkan EF Core ke SQL, seperti pemanggilan metode di dalam `Where`: permintaan pemeriksa gagal dengan 500, dan pemeriksaan untuk *endpoint* itu gagal. (b) `Include` pembayaran atau layanan yang terlupa: pesanan kembali dengan daftar `payments` kosong atau tanpa nama layanan, dan pemeriksaan pembayaran (`payments[0].method qris`) gagal. (c) Waktu yang disimpan dan dibaca kembali dengan jenis atau presisi berbeda di SQLite: pemeriksaan format waktu gagal. Setiap kegagalan menunjuk satu *endpoint* dan satu field, dan di situlah tempat mencarinya.

Daftar Pustaka

[1] The PHP Group, “HTTP context options” and “$http_response_header,” PHP Manual. Accessed: Sep. 25, 2026. [Online]. Available: https://www.php.net/manual/en/context.http.php

[2] IEEE, “IEEE Standard for Floating-Point Arithmetic,” IEEE Std 754-2019, 2019.

[3] Microsoft, “Model validation in ASP.NET Core MVC and Razor Pages” and “Handle errors in ASP.NET Core,” ASP.NET Core 10 documentation. Accessed: Sep. 25, 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/mvc/models/validation

[4] Microsoft, “PasswordHasher Class,” .NET API reference, and dotnet/aspnetcore source, `PasswordHasher.cs`. Accessed: Sep. 25, 2026. [Online]. Available: https://learn.microsoft.com/dotnet/api/microsoft.aspnetcore.identity.passwordhasher-1

[5] Microsoft, “ASP.NET Core fundamentals: Minimal APIs, controllers, and configuration,” ASP.NET Core 10 documentation. Accessed: Sep. 25, 2026. [Online]. Available: https://learn.microsoft.com/aspnet/core/fundamentals/

[6] Laravel, “Routing: Route Model Binding and Middleware,” Laravel 13.x documentation. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/routing

[7] Laravel, “Laravel Sanctum: API Token Authentication and Testing,” Laravel 13.x documentation. Accessed: Sep. 25, 2026. [Online]. Available: https://laravel.com/docs/13.x/sanctum

[8] Broadcom, “Spring Security Reference: Servlet Applications,” version 7.0. Accessed: Sep. 25, 2026. [Online]. Available: https://docs.spring.io/spring-security/reference/servlet/