Profitina Laundry

Edisi Pertama

Irfan Subakti

Profitina

profitina.com

DAFTAR ISI

Kata Pengantar

Buku ini adalah laporan lapangan, bukan brosur. Ia ditulis sementara sistem yang diceritakannya sedang dibangun, dan ia mempertahankan catatan kerjanya alih-alih merapikannya.

Kebanyakan studi kasus buku teks diciptakan semata untuk mendemonstrasikan satu fitur sintaks. Profitina Laundry mengambil jalan berbeda: berpijak pada bisnis kecil nyata yang benar-benar dimiliki dan dijalankan sang penulis, dibangun ulang secara jujur di empat stack teknologi berbeda, bukan sekali saja.

Profitina Laundry dibuka dengan menjelaskan mengapa satu studi kasus yang berpijak pada kenyataan, dibangun empat kali, lebih bernilai daripada empat studi kasus rekaan yang masing-masing dibangun sekali, lalu menceritakan kisah nyata (sengaja singkat, sengaja disesuaikan demi privasi) di balik bisnis ini. Buku ini menjelaskan bagaimana bisnis laundry sesungguhnya beroperasi, terlepas dari perangkat lunak apa pun, lalu mengubah kenyataan operasional itu menjadi model data — skema yang dibentuk oleh peran nyata, aturan nyata, dan kebutuhan pelaporan nyata. Sebuah tur arsitektur menelusuri keempat teknologi tempat bisnis ini dibangun, dengan satu bab penuh yang masuk ke dalam aplikasi ASP.NET secara spesifik, dan satu studi kasus debugging khusus seputar satu bug nyata yang mendidik: bug ValidationGroup. Buku ini ditutup secara praktis — cara menjalankan Profitina Laundry sendiri — dan reflektif, dengan pelajaran yang melampaui kode dan ke mana proyek ini melangkah selanjutnya.

Setibanya Anda di halaman terakhir, Anda semestinya mampu:

- Melihat bagaimana kenyataan operasional bisnis kecil nyata — penerimaan, pemrosesan, siap diambil, pembayaran — diubah menjadi skema basis data.
- Membandingkan logika bisnis yang sama yang diimplementasikan di empat stack teknologi berbeda, berdampingan.
- Masuk ke dalam aplikasi ASP.NET yang dibangun untuk masalah bisnis nyata, bukan rekaan.
- Mengikuti studi kasus debugging sungguhan — bug ValidationGroup — dari gejala sampai akar masalah dan perbaikannya.
- Menjalankan Profitina Laundry sendiri sebagai implementasi referensi yang bisa langsung dipraktikkan.
- Membawa pulang pelajaran tentang memijakkan studi kasus pada kenyataan yang berlaku jauh melampaui proyek ini saja.

Untuk siapa buku ini ditulis: mahasiswa yang mengikuti kuliah Pemrograman Web (WebPro) atau Pemrograman Berbasis Kerangka Kerja (FBP), yang bertemu sistem persis ini sebagai studi kasus berjalan mereka di banyak lecture; pengajar yang mencari studi kasus untuk diadaptasi ke kuliah mereka sendiri; serta pemilik usaha kecil dan masyarakat umum yang penasaran apa sesungguhnya yang dibutuhkan untuk "mendigitalkan" bisnis jasa kecil dalam praktiknya.

Sepatah kata tentang metode. Setiap contoh kode dalam buku ini dikompilasi dan dijalankan sebelum dituliskan, dan lampirannya memuat catatan verifikasi sebagai buktinya. Ketika sebuah angka muncul, angka itu berasal dari pengukuran, bukan dari ingatan. Bila Anda menemukan kekeliruan, atau bagian yang bisa lebih jelas, saya sungguh ingin mendengarnya — begitulah edisi berikutnya menjadi lebih baik.

Surabaya, September 2026

Irfan Subakti
yifana@gmail.com

Profitina
profitina.com

BAB 1

Prakata — Mengapa Buku Ini Ada

Sebuah bisnis laundry kecil, kisah nyata di baliknya, dan alasan mengapa semua ini layak dituliskan dengan baik.

> **Apa buku ini**
>
> Ini adalah kisah sekaligus dokumentasi teknis dari Profitina Laundry: sebuah bisnis laundry fiktif, yang dibangun sebagai sistem perangkat lunak nyata dan berjalan sebanyak empat kali secara terpisah, satu kali untuk setiap teknologi sisi-server yang diajarkan di mata kuliah Pemrograman Web (WebPro), dan kelak untuk kelima kalinya di mata kuliah Framework-based Programming (FBP). Buku ini ditulis untuk tiga pembaca sekaligus — mahasiswa yang belajar membangun sistem nyata, pengajar yang mencari studi kasus yang jujur dan membumi, serta pemilik bisnis yang penasaran apa sebenarnya yang dibutuhkan untuk "mendigitalisasi" sebuah bisnis jasa kecil.

1.1 Satu Studi Kasus, Banyak Tujuan

Kebanyakan studi kasus di buku teks dibuat semata-mata untuk mengilustrasikan satu fitur sintaks: tabel "Book" dan "Author" untuk mendemonstrasikan JOIN, tabel "Product" dan "Order" untuk mendemonstrasikan foreign key. Studi kasus semacam itu sekali pakai, dan langsung dilupakan begitu pelajaran selesai. Profitina Laundry dibangun untuk menjadi kebalikannya: satu bisnis yang koheren, dimodelkan sekali dengan perhatian yang sungguh-sungguh, lalu diimplementasikan secara lengkap dan jujur di setiap teknologi yang perlu diajarkan sebuah mata kuliah — sehingga apa yang dipelajari mahasiswa tentang model data bisnis ini, aturan-aturannya, dan sisi-sisi kasarnya di halaman pertama, tetap benar hingga halaman dua ratus.

Keputusan itu ada harganya. Membangun satu aplikasi yang berjalan saja sudah merupakan pekerjaan. Membangun bisnis yang sama sebanyak empat kali, dalam empat teknologi berbeda, masing-masing cukup nyata untuk memunculkan bug dan trade-off desain yang sungguhan, adalah pekerjaan yang jauh lebih besar. Buku ini ada karena pekerjaan itu layak dituliskan dengan baik, alih-alih dibiarkan hidup hanya sebagai berkas-berkas sumber yang berserakan di sebuah folder proyek — dorongan yang sama yang membuat tim rekayasa sungguhan menulis dokumen desain, bukan sekadar mengirim kodenya.

1.2 Berlandaskan Bisnis Nyata

Profitina Laundry bukan skenario yang murni dikarang. Ia dibangun berdasarkan pengalaman penulis sendiri dalam memiliki dan mengelola sebuah bisnis laundry, bersama seorang partner bisnis dan sejumlah kecil karyawan, pada babak kehidupan yang lebih awal. Sistem dalam buku ini — para pelanggan, karyawan, layanan, tagihan, dan ritme keseharian mulai dari penerimaan, pencucian, pelipatan, hingga pengambilan — terinspirasi langsung dari pengalaman nyata tersebut, dengan penyesuaian di sana-sini untuk kepentingan pembelajaran: nama, angka, dan detail spesifik difiksikan, tetapi bentuk bisnisnya, peran-peran yang sungguh-sungguh dijalankan orang-orang di dalamnya, dan ritme operasional yang sungguh-sungguh mereka ikuti, adalah nyata.

> **Mengapa hal ini penting lebih dari yang terlihat**
>
> Studi kasus yang dikarang murni untuk pelajaran sintaks cenderung memodelkan basis datanya saja, bukan bisnisnya — hanya cukup struktur untuk mendemonstrasikan sebuah JOIN atau foreign key, tidak lebih. Studi kasus yang berlandaskan bisnis nyata cenderung memunculkan kompleksitas nyata di tempat yang tepat: seorang supervisor yang juga seorang karyawan (relasi self-referencing yang sungguhan, bukan yang dibuat-buat), sebuah layanan yang tidak bisa begitu saja dihapus setelah seorang pelanggan ditagih untuknya (persoalan integritas referensial yang sungguhan, bukan aturan artifisial), seorang karyawan bagian depan yang seharusnya diberi pelanggan walk-in baru berdasarkan beban kerja aktualnya saat ini, bukan selalu nama yang sama di urutan teratas. Semua ini muncul dalam skema dan kode buku ini bukan karena membuat contoh buku teks yang baik, melainkan karena memang seperti itulah catatan bisnis laundry yang sungguhan.

1.3 Untuk Siapa Buku Ini Ditulis

Tiga kelompok pembaca yang saling tumpang tindih akan menemukan sesuatu yang berguna di sini, dan buku ini berusaha untuk tidak hanya menulis untuk yang pertama di antaranya.

- Mahasiswa dan pembelajar mandiri yang menempuh mata kuliah WebPro atau FBP, yang akan bertemu dengan sistem yang persis sama ini sebagai studi kasus yang berjalan di banyak pertemuan kuliah, dan yang dapat menggunakan buku ini sebagai satu tempat di mana keseluruhan sistem — bukan hanya potongan satu bab — dijelaskan secara koheren.
- Pengajar yang mencari studi kasus yang dapat mereka adaptasi untuk mata kuliah mereka sendiri, yang mungkin lebih tertarik bukan pada kode spesifik Profitina Laundry, melainkan pada keputusan desain di baliknya: mengapa satu skema kanonis, mengapa empat implementasi teknologi independen alih-alih satu, mengapa pengujian eksekusi nyata dan pengungkapan bug yang jujur, alih-alih contoh clean-room yang diidealkan.
- Pemilik bisnis kecil dan masyarakat umum yang penasaran apa sebenarnya yang dibutuhkan untuk "mendigitalisasi" bisnis jasa kecil seperti laundry — bukan sebagai promosi untuk produk tertentu, melainkan sebagai contoh kerja yang jujur: data apa yang sebenarnya perlu dicatat bisnis seperti ini, aturan apa yang perlu ditegakkan atas data tersebut, dan seperti apa sebenarnya sistem pemesanan-dan-penagihan yang nyata dan berjalan untuk bisnis seperti ini, dari ujung ke ujung.

1.4 Komitmen pada Kejujuran

Setiap klaim teknis dalam buku ini didukung oleh sesuatu yang benar-benar dijalankan, bukan sekadar dideskripsikan. Ketika sebuah halaman mengatakan suatu potongan kode berhasil, klaim itu berasal dari log eksekusi yang nyata: basis data nyata, server web nyata, permintaan dan respons HTTP nyata, diperiksa langsung, bukan diasumsikan. Ketika proses pengembangan proyek ini sendiri menemukan bug yang sungguhan — dan memang terjadi, lebih dari sekali — buku ini mengatakannya secara terus terang, termasuk persisnya bagaimana bug itu ditemukan dan persisnya bagaimana bug itu diperbaiki, karena itu setidaknya merupakan pelajaran yang sama berharganya dengan satu baris kode yang kebetulan berhasil pada percobaan pertama.

Apa yang tidak akan dilakukan buku ini

Buku ini tidak akan mengklaim sebuah teknologi "langsung berhasil" lintas platform padahal sebenarnya tidak — ASP.NET Web Forms klasik, yang dibahas di Bab 6 dan 8, adalah contoh yang jelas: ini teknologi nyata yang masih banyak digunakan, tetapi tidak pernah diporting ke .NET modern yang lintas platform, dan buku ini mengatakannya persis seperti itu, alih-alih diam-diam berharap pembaca tidak menyadarinya di sebuah Mac. Buku ini juga tidak akan menyajikan kisah pengembangan yang bersih dan bebas bug padahal kisah yang sesungguhnya memiliki bug yang sungguhan di dalamnya. Kedua jalan pintas itu umum ditemukan dalam tulisan teknis, dan keduanya akan membuat buku ini kurang berguna justru bagi pembaca yang ditulis untuknya.

1.5 Bagaimana Buku Ini Disusun

Bab 2 menceritakan kisah nyata (yang sengaja dibuat singkat) di balik Profitina Laundry. Bab 3 menjelaskan bagaimana sebuah bisnis laundry sebenarnya beroperasi, terlepas dari perangkat lunak apa pun, untuk pembaca yang belum pernah menjalankannya. Bab 4 mengubah kenyataan operasional itu menjadi model data — skema basis data sesungguhnya yang menjadi pilihan akhir proyek ini, dan alasannya. Bab 5 mengunjungi keempat implementasi teknologi pada tingkat tinjauan umum, dan Bab 6 masuk lebih dalam ke yang paling kaya di antara keempatnya, aplikasi ASP.NET Web Forms, termasuk aturan bisnis persis yang ditegakkannya. Bab 7 adalah satu studi kasus yang berfokus pada satu bug nyata yang ditemukan proyek ini pada kodenya sendiri, bagaimana bug itu didiagnosis, dan bagaimana ia diperbaiki — bab tentang metodologi debugging sama besarnya dengan tentang ASP.NET secara spesifik. Bab 8 adalah panduan praktis memulai bagi pembaca yang ingin menjalankan sendiri salah satu dari keempat implementasi ini, di Windows, macOS, atau Linux. Bab 9 melangkah mundur dan menarik pelajaran yang ditawarkan proyek ini di luar kodenya sendiri, bagi pengajar maupun pembaca dari kalangan bisnis. Bab 10 menutup dengan ke mana arah Profitina Laundry selanjutnya.

Pembaca yang hanya tertarik pada teknologinya dapat mulai dari Bab 4 dan melewati Bab 2-3 tanpa kehilangan apa pun yang esensial bagi kodenya. Pembaca yang lebih tertarik pada landasan bisnis dunia nyata, atau pada apa yang bisa dipelajari pemilik bisnis kecil dari proyek seperti ini, mungkin akan merasa Bab 2, 3, dan 9 paling berharga, dan dapat memperlakukan bab-bab yang sarat kode di antaranya sebagai kedalaman opsional, bukan bacaan wajib.

BAB 2

Kisah Nyata di Balik Profitina Laundry

Catatan singkat dan jujur tentang asal-usul bisnis ini.

> **Singkatnya**
>
> Profitina Laundry dibuat berdasarkan pengalaman penulis dalam memiliki dan mengelola bisnis laundry bersama seorang partner bisnis dan sekelompok karyawan, pada babak kehidupan yang lebih awal. Tentu saja, ada penyesuaian di sana-sini untuk kepentingan buku ini dan mata kuliah-mata kuliah terkait. Itulah keseluruhannya — dan, secara sengaja, bab ini tidak akan membahas jauh lebih dari itu.

2.1 Bisnis Nyata di Balik Studi Kasus Pembelajaran

Akan lebih mudah untuk mengarang Profitina Laundry dari nol — menyketsa sebuah bisnis kecil yang terdengar masuk akal, memberinya nama, dan membangun skema di sekitarnya sesuai apa pun yang tampak nyaman secara pedagogis. Begitulah sebenarnya kebanyakan studi kasus buku teks dibuat, dan Bab 1 telah menjelaskan mengapa jalan pintas itu cenderung menghasilkan studi kasus yang hanya memodelkan apa yang dibutuhkan pelajaran, tidak lebih.

Profitina Laundry mengambil jalan yang berbeda. Struktur pemilik-dan-partner-nya, karyawan bagian depan dan bagian belakangnya, layanan-layanannya, pelanggan-pelanggannya, dan ritme keseharian menerima laundry lalu mengembalikannya dalam keadaan bersih, bukanlah sesuatu yang dikarang dari halaman kosong — semua itu diambil dari sebuah bisnis laundry nyata yang benar-benar dimiliki dan dijalankan penulis, bersama seorang partner dan staf kecil, pada masa yang lebih awal. Sistem yang didokumentasikan buku ini adalah bisnis tersebut, diterjemahkan menjadi perangkat lunak, dengan detail-detail spesifik disesuaikan untuk kepentingan pembelajaran.

2.2 Mengapa Detailnya Disesuaikan, Bukan Persis Sama

Dua hal ini sama-sama benar, dan buku ini ingin jujur mengenai keduanya. Pertama, bisnis yang menjadi landasan Profitina Laundry adalah nyata: pelanggan nyata, karyawan nyata, keputusan sehari-hari yang nyata mengenai harga, penempatan staf, dan beban kerja. Kedua, buku ini tidak mereproduksi nama persis, angka persis, atau riwayat persis dari bisnis tersebut — nama-nama dalam data contoh buku ini, angka-angka harga spesifik, dan urutan kejadian yang tepat, semuanya difiksikan dan disesuaikan, kadang untuk alasan privasi biasa yang layak dimiliki kisah bisnis nyata mana pun, dan kadang semata-mata karena sebuah contoh pembelajaran lebih diuntungkan oleh angka bulat dan ilustrasi yang bersih, dibandingkan buku besar yang sesungguhnya.

Yang bertahan dari penyesuaian itu adalah bentuk dari hal tersebut: peran-peran yang sungguh-sungguh dipegang orang-orang, aturan-aturan yang sungguh-sungguh perlu ditegakkan bisnis, dan ritme operasional yang sungguh-sungguh diikutinya. Itu juga, bukan kebetulan, persis apa yang perlu benar-benar dipahami sebuah model data — sebab itulah skema di Bab 4 dapat bersandar pada landasan nyata ini untuk keputusan-keputusan seperti relasi supervisor yang self-referencing atau layanan yang menolak untuk dihapus setelah seorang pelanggan ditagih untuknya, alih-alih mengarang aturan-aturan tersebut demi efek dramatis.

2.3 Bab yang Singkat, dengan Sengaja

Pembaca yang berharap kisah asal-usul yang lebih panjang — tanggal spesifik, nama bisnis, sebuah kota, kisah rinci bagaimana kemitraan itu terbentuk atau apa yang terjadi di sepanjang jalan — tidak akan menemukannya di sini, dan itu adalah pilihan yang disengaja, bukan sebuah kelalaian. Pelajaran yang ingin disampaikan buku ini tidak bergantung pada detail-detail spesifik semacam itu, dan menawarkan detail spesifik yang dikarang sebagai gantinya justru akan mengaburkan batas antara apa yang benar-benar diambil dari pengalaman dan apa yang dibuat-buat demi warna naratif. Bab ini hanya mengatakan apa yang perlu dikatakan: bisnisnya nyata, perangkat lunaknya baru, dan penyesuaian di antara keduanya adalah penyesuaian yang wajar.

Bab selanjutnya melanjutkan dari sini ke arah yang lebih berguna bagi pembaca yang membangun perangkat lunaknya: bab tersebut menjelaskan, terlepas dari teknologi tertentu mana pun, bagaimana sebuah bisnis laundry seperti ini sebenarnya beroperasi sehari-hari — kenyataan operasional yang kemudian diubah Bab 4 menjadi model data yang konkret.

BAB 3

Bagaimana Bisnis Laundry Sebenarnya Bekerja

Kenyataan operasional di balik perangkat lunak, sebelum satu baris kode pun ditulis.

Sebelum melihat satu baris kode atau satu tabel basis data pun, akan sangat membantu untuk memahami bisnis yang dilayani oleh semua itu. Bisnis laundry cukup sederhana untuk dijelaskan dalam beberapa paragraf, namun cukup kaya untuk membutuhkan perangkat lunak sungguhan begitu ia tumbuh melewati ukuran tertentu — dan itulah persis rentang yang menjadi perhatian buku ini.

3.1 Alur Fisik: Dari Penerimaan hingga Pengambilan

Satu muatan cucian bergerak melalui sejumlah tahap tertentu antara saat pelanggan mengantarkannya dan saat mereka mengambilnya kembali, dan hampir setiap bisnis laundry, bagaimanapun cara pengelolaannya, mengikuti suatu versi dari urutan yang sama.

- Penerimaan — pelanggan membawa sekantong atau sekeranjang barang, atau meminta dijemput. Seorang karyawan menimbangnya (laundry biasanya menetapkan harga per kilogram) atau menghitung jumlah potongnya, mencatat layanan mana yang berlaku (cuci-lipat biasa, cuci kering, setrika saja, dan sebagainya), dan mencatat milik siapa pesanan tersebut.
- Pemrosesan — cucian dicuci, dikeringkan, dan dilipat atau disetrika sesuai layanan yang dipesan. Pada bisnis kecil, ini dilakukan secara manual atau dengan beberapa mesin, oleh karyawan mana pun yang sedang bertugas; pada bisnis yang lebih besar, prosesnya dapat bergerak melalui stasiun-stasiun khusus.
- Siap diambil — begitu pemrosesan selesai, pesanan menunggu pelanggannya, dan bisnis membutuhkan cara untuk mengetahui pesanan mana yang benar-benar sudah siap dibandingkan yang masih dalam proses, terutama begitu lebih dari segelintir pesanan berada di toko secara bersamaan.
- Pengambilan dan pembayaran — pelanggan kembali, pesanan diserahkan, dan pembayaran diterima (atau, untuk pelanggan tetap dengan rekening berjalan, ditambahkan ke tagihan yang diselesaikan secara berkala).

Skema di Bab 4 menamai tahap-tahap ini secara langsung: status sebuah Bill bergerak melalui persis urutan ini, dan urutan itu bukanlah pilihan desain perangkat lunak yang sewenang-wenang — itu hanyalah apa yang terjadi di meja depan, dituliskan.

3.2 Orang-Orangnya: Peran dalam Laundry Kecil

Bisnis laundry yang cukup kecil untuk berjalan tanpa perangkat lunak khusus pun biasanya masih memiliki lebih dari satu jenis peran di dalamnya, dan relasi antar peran itu lebih penting daripada yang mungkin terlihat pada pandangan pertama.

- Seorang pemilik atau kemitraan kecil dari beberapa pemilik, yang menetapkan harga, memutuskan layanan apa yang ditawarkan, dan pada akhirnya bertanggung jawab atas bisnis tersebut.

- Staf karyawan yang kecil, yang bertugas di meja depan, menangani penerimaan, dan melakukan pencucian, pengeringan, pelipatan, dan penyetrikaan itu sendiri.
- Seringkali, satu karyawan bertindak sebagai supervisor shift atau staf senior bagi yang lain — seseorang yang tetap merupakan seorang karyawan, tetapi yang dilapori karyawan lain sehari-hari.

> **Mengapa relasi supervisor ini penting bagi model data**
>
> Poin terakhir itu — seorang supervisor yang juga seorang karyawan — mudah terlewatkan dalam studi kasus yang dikarang, karena mengarang sebuah bisnis dari nol jarang menghasilkan struktur organisasi dengan tekstur yang nyata. Bisnis laundry yang sungguhan, bagaimanapun, hampir selalu memiliki bentuk ini: seorang karyawan senior mengawasi yang lain tanpa menjadi jenis orang yang terpisah dalam sistem. Itulah persis mengapa tabel Employee di Bab 4 memiliki relasi supervisor yang self-referencing — satu rekaman karyawan menunjuk ke rekaman karyawan lain — alih-alih konsep "manajer" yang terpisah dan lebih sederhana. Bisnis nyatalah yang membentuk skema, bukan sebaliknya.

3.3 Apa yang Harus Dicatat, dan Mengapa

Bahkan bisnis laundry yang sangat kecil, yang dijalankan sepenuhnya di atas kertas, pada akhirnya mencatat kurang lebih segelintir hal yang sama, karena masing-masing menjawab pertanyaan yang terus-menerus diajukan kepada bisnis tersebut.

Apa yang dicatat	Pertanyaan yang dijawabnya
Pelanggan	Cucian siapa ini, dan bagaimana kita menghubungi mereka saat sudah siap?
Karyawan	Siapa yang menerima pesanan ini, siapa yang mengerjakannya, dan siapa yang bertanggung jawab atasnya?
Layanan	Apa yang dipesan, dan berapa seharusnya biayanya?
Tagihan / pesanan	Apa status pesanan spesifik ini saat ini, dan berapa yang harus dibayar pelanggan?

Setiap satu dari keempat kategori ini muncul, dengan nama satu atau lain, di setiap satu dari empat implementasi perangkat lunak Profitina Laundry — tabel `orders` yang diratakan yang diajarkan jalur JSP di Bab 10 menggabungkan Services dan Bills menjadi satu baris, sementara tabel `Customer` / `Employee` / `Service` / `Bill` jalur ASP.NET (Bab 6) menjaga keempatnya tetap terpisah dan berelasi. Keduanya merupakan jawaban yang sah atas kenyataan operasional yang sama; Bab 5 membandingkan keduanya secara langsung.

3.4 Aturan yang Ditegakkan Laundry Sungguhan, Disadari atau Tidak

Bisnis yang dijalankan di atas kertas menegakkan aturannya sendiri melalui kebiasaan dan akal sehat, bahkan ketika tidak ada seorang pun yang pernah menuliskannya sebagai kebijakan formal. Mendigitalisasi bisnis berarti membuat aturan-aturan tak tertulis itu menjadi eksplisit — dan beberapa aturan Profitina Laundry layak disebutkan di sini karena berasal langsung dari bagaimana laundry sungguhan sebenarnya harus beroperasi, bukan dari selera desain perangkat lunak.

- Sebuah layanan yang sudah ditagihkan ke pelanggan tidak dapat begitu saja dihapus dari daftar harga, karena tagihan tersebut masih merujuk kepadanya — menghapusnya akan membuat tagihan itu menunjuk ke sesuatu yang tidak ada. Laundry sungguhan tidak akan pernah membiarkan perubahan daftar harga secara diam-diam menghapus apa yang sudah dibebankan kepada pelanggan, dan perangkat lunak pun seharusnya tidak.
- Pelanggan walk-in baru sebaiknya ditugaskan ke karyawan bagian depan mana pun yang saat ini memiliki beban kerja paling ringan, bukan selalu ke karyawan senior yang sama secara default — toko sungguhan menyeimbangkan pekerjaan di antara siapa pun yang sedang bertugas, dan sistem pemesanan yang mengabaikan hal ini akan cepat menghasilkan satu karyawan yang kelebihan beban dan beberapa lainnya yang menganggur.
- Status sebuah pesanan hanya dapat bergerak maju melalui urutan penerimaan-hingga-pengambilan dalam satu arah pada operasi normal; sebuah pesanan tidak kembali ke "diterima" setelah ditandai siap, karena itu bukan sesuatu yang terjadi pada cucian sungguhan.

Ini bukanlah persyaratan perangkat lunak abstrak yang dikarang demi membuat contoh buku teks yang baik. Semua ini adalah akal sehat biasa dalam menjalankan meja laundry, dan Bab 4 dan 6 menunjukkan persis bagaimana masing-masing ditegakkan dalam model data dan dalam kode aplikasi ASP.NET.

3.5 Dari Operasi ke Perangkat Lunak

Dengan gambaran operasional ini di tangan, bab berikutnya mengubahnya menjadi sesuatu yang dapat disimpan basis data: sebuah skema konkret, dengan tabel, kolom, dan relasi yang konkret, dibangun untuk mengatakan persis apa yang dijelaskan Bagian 3.3 dan 3.4 di atas — tidak lebih, dan tidak kurang.

BAB 4

Model Data: Mengubah Bisnis Menjadi Skema

Customer, Employee, Service, dan Bill — serta aturan nyata di balik setiap relasinya.

Bab 3 menjelaskan empat kategori yang harus dicatat sebuah bisnis laundry — pelanggan, karyawan, layanan, dan tagihan yang mengikat semuanya bersama — serta sejumlah aturan tak tertulis yang ditegakkan laundry sungguhan tanpa pernah menyebutnya sebagai "aturan bisnis". Bab ini mengubah gambaran operasional itu menjadi skema sesungguhnya yang menjadi pilihan akhir proyek ini: model kanonis `Bill` / `Customer` / `Employee` / `Service` yang diimplementasikan langsung oleh jalur ASP.NET (Bab 6-8), dan yang tabel `orders` yang diratakan pada jalur JSP (Bab 10) merupakan penyederhanaan yang disengaja darinya.

4.1 Dari Mana Skema Ini Berasal

Ini bukan skema yang dikarang untuk buku ini. Skema ini diadaptasi, tabel demi tabel dan kolom demi kolom, dari desain entity-relationship asli proyek ini — sebuah diagram Vertabelo yang digambar berdasarkan bisnis nyata yang dijelaskan Bab 2, kemudian diekspor sebagai DDL MySQL (`All SQLs.txt`). Versi yang ditampilkan dalam bab ini adalah dialek SQLite yang benar-benar dijalankan aplikasi ASP.NET; basis data MariaDB jalur JSP menggunakan bentuk logis yang sama untuk `Customer`, `Employee`, `Service`, dan `Bill`, hanya dengan perbedaan di tingkat dialek (`ENUM` alih-alih `TEXT` dengan batasan `CHECK`, misalnya).

4.2 Customer dan Employee: Dua Tabel, Satu Bentuk

Seorang pelanggan dan seorang karyawan, pada tingkat apa yang perlu diketahui sebuah laundry tentang seseorang, hampir merupakan jenis rekaman yang sama: nama, jenis kelamin, tempat asal, tanggal lahir, alamat, nomor telepon, dan alamat email. Skema ini mencerminkan hal tersebut secara langsung.

```
CREATE TABLE Customer (
    id        TEXT PRIMARY KEY CHECK (length(id) <= 5),
    name      TEXT NOT NULL,
    gender    TEXT NOT NULL DEFAULT 'f' CHECK (gender IN ('f','m')),
    place     TEXT NOT NULL,
    birthdate TEXT NOT NULL,
    address   TEXT NOT NULL,
    phone     TEXT NOT NULL,
    enrolled  TEXT NOT NULL,
    email     TEXT NOT NULL,
    password  TEXT NOT NULL CHECK (length(password) <= 5),
    credit    NUMERIC(10,2) NOT NULL DEFAULT 0
);
```

`Employee` membawa kolom personal yang sama, ditambah dua kolom yang menandainya secara spesifik sebagai rekaman karyawan: `section` (bagian bisnis mana tempat karyawan ini bekerja — `transaction`, meja depan, atau `wash`, sisi pemrosesan) dan `started` (kapan masa kerjanya dimulai, mencerminkan `Customer.enrolled`). Kedua tabel juga membawa kolom `password` yang dibatasi hingga lima karakter, dan kedua jenis ID (`c0001`, `e0001`, dan seterusnya) juga dibatasi hingga lima

karakter — keduanya diwarisi tanpa perubahan dari desain Vertabelo asli tahun 2020, dan Bagian 4.7 membahas yang pertama dari keduanya secara jujur, alih-alih mengabaikannya.

4.3 Supervisor yang Self-Referencing

Bab 3 menyebutkan sebuah fakta operasional yang nyata: dalam laundry kecil, satu karyawan sering mengawasi yang lain sembari tetap menjadi seorang karyawan itu sendiri, alih-alih menempati kategori "manajer" yang terpisah. Skema ini mengekspresikan hal tersebut persis seperti kenyataannya — sebagai tabel `Employee` dengan foreign key yang menunjuk kembali ke dirinya sendiri.

```
    spv_id    TEXT NULL,
    FOREIGN KEY (spv_id) REFERENCES Employee(id)
        ON DELETE SET NULL ON UPDATE CASCADE
```

Mengapa ON DELETE SET NULL secara spesifik di sini

Jika seorang karyawan yang mengawasi suatu saat meninggalkan bisnis dan rekamannya dihapus, karyawan-karyawan yang melapor kepadanya seharusnya tidak ikut terhapus — mereka cukup menjadi tidak terawasi hingga seorang supervisor baru ditugaskan. `ON DELETE SET NULL` mengekspresikan persis itu: menghapus `e0001` (Borat Sagdiyev, yang mengawasi `e0002` dan `e0003` pada data awal di bawah) mengosongkan `spv_id` mereka kembali menjadi `NULL`, alih-alih ikut menghapus atau memblokirnya sama sekali. Laundry sungguhan tidak memecat seluruh timnya karena supervisornya pindah, dan skema ini sependapat.

4.4 Service, dan Aturan Melawan Penghapusan Sesuatu yang Sudah Ditagih

`Service` adalah tabel paling sederhana dalam skema ini — sebuah id, deskripsi, dan harga — tetapi di sinilah aturan perlindungan-hapus dari Bab 3 menjadi terlihat sebagai sebuah batasan sungguhan, bukan kebijakan yang harus diingat seseorang untuk ditegakkan secara manual.

```
CREATE TABLE Service (
    id          TEXT PRIMARY KEY CHECK (length(id) <= 5),
    description TEXT NOT NULL,
    price       NUMERIC(7,2) NOT NULL
);
```

`Bill.Service_id` mereferensikan `Service.id` dengan `ON DELETE RESTRICT` — basis data itu sendiri menolak untuk menghapus sebuah layanan yang masih dirujuk oleh tagihan mana pun yang ada. Bab 6 menunjukkan seperti apa hal ini dari sisi aplikasi ASP.NET: upaya menghapus layanan yang sedang digunakan ditangkap dan dilaporkan kembali ke pengguna sebagai pesan kesalahan yang jelas, alih-alih dibiarkan gagal sebagai pengecualian basis data yang tidak tertangani, atau lebih buruk lagi, diam-diam dibiarkan membuat sebuah tagihan menjadi yatim.

4.5 Bill: Tempat Customer, Employee, dan Service Bertemu

`Bill` adalah tabel yang menjadi alasan keberadaan semua tabel lainnya — satu baris per pesanan, mengikat bersama siapa yang mengantarkannya, siapa yang menanganinya, dan apa yang dipesan.

```
CREATE TABLE Bill (
    id          TEXT PRIMARY KEY CHECK (length(id) <= 5),
    arrived     TEXT NOT NULL,
    finished    TEXT NULL,
    departed    TEXT NULL,
    status      TEXT NOT NULL DEFAULT 'queued'
                CHECK (status IN ('queued','finished','departed')),
    rack        TEXT NOT NULL,
    unit        INTEGER NOT NULL,
    payment     NUMERIC(10,2) NOT NULL,
    Customer_id TEXT NOT NULL,
    Employee_id TEXT NOT NULL,
    Service_id  TEXT NOT NULL,
    FOREIGN KEY (Customer_id) REFERENCES Customer(id) ON DELETE RESTRICT,
    FOREIGN KEY (Employee_id) REFERENCES Employee(id) ON DELETE RESTRICT,
    FOREIGN KEY (Service_id)  REFERENCES Service(id)  ON DELETE RESTRICT
);
```

Ketiga foreign key `Bill` menggunakan `RESTRICT`, dengan alasan mendasar yang sama seperti Bagian 4.4: seorang pelanggan, karyawan, atau layanan yang masih dirujuk oleh tagihan mana pun yang ada tidak dapat begitu saja lenyap dari bawah tagihan tersebut. `rack` mencatat di mana cucian secara fisik disimpan selagi menunggu (detail fisik nyata yang harus dicatat laundry sungguhan, yang jarang disebutkan dalam studi kasus yang dikarang), dan `unit` mencatat berapa banyak potong atau satuan kilogram yang dicakup pesanan tersebut, yang langsung memengaruhi `payment`.

4.6 Catatan tentang Status: Satu Kolom Kasar, Tiga Stempel Waktu

`Bill.status` hanya bergerak melalui tiga nilai — `queued`, `finished`, `departed` — yang lebih kasar dibandingkan alur empat-tahap penerimaan-hingga-pengambilan yang dijelaskan Bab 3 dalam bentuk prosa (penerimaan, pemrosesan, siap, diambil). Itu bukan ketidakkonsistenan yang perlu ditutup-tutupi; itu adalah perbedaan yang nyata dan diungkapkan secara terbuka dalam cara kedua jalur memodelkan kenyataan operasional yang sama. Skema ASP.NET melipat "penerimaan" dan "pemrosesan" menjadi satu status `queued`, dan alih-alih nilai status keempat untuk "siap", ia mencatat momen sebuah pesanan menjadi siap secara langsung, sebagai stempel waktu `finished`, berdampingan dengan `arrived` dan `departed`. Tabel `orders` jalur JSP (Bab 10) mengambil pendekatan sebaliknya: empat nilai status eksplisit (`received`, `washing`, `ready`, `picked_up`) dan tanpa kolom stempel waktu terpisah sama sekali.

Keduanya merupakan cara yang sah untuk memodelkan kenyataan empat-tahap yang sama; Bab 5 membandingkan skema kedua jalur secara berdampingan, dan README serta kedua berkas skema proyek ini mengungkapkan perbedaan ini secara eksplisit, alih-alih memperlakukan salah satunya sebagai yang lebih benar.

4.7 Dua Pilihan Desain yang Layak Disebutkan secara Jujur

- ID lima-karakter (`c0001`, `e0001`, `C4W`, `b0001`) adalah warisan langsung dari desain Vertabelo asli tahun 2020, bukan batasan yang akan dipilih buku ini hari ini untuk sebuah bisnis yang sedang bertumbuh — sebuah jaringan laundry sungguhan yang melampaui ID lima-karakter akan perlu memperlebar kolom ini, dan itu adalah jenis migrasi skema yang normal dan diharapkan, bukan cacat desain yang spesifik pada Profitina Laundry.
- Baik `Customer.password` maupun `Employee.password` menyimpan nilai teks-polos yang dibatasi hingga lima karakter. Ini diwariskan dari desain asli demi kesetiaan skema, dan tidak ada satu pun kolom ini yang pernah dibaca dari atau ditulis ke oleh halaman mana pun dalam aplikasi — mata kuliah yang menaungi skema ini tidak pernah mengajarkan fitur login, sehingga kedua kolom ini ada semata demi kelengkapan. Sistem sungguhan yang menyimpan kredensial aktual harus melakukan hashing terhadapnya (dengan bcrypt, PBKDF2, atau Argon2) dan tidak boleh pernah membatasi panjang kata sandi seperti ini; buku ini mengatakannya secara terus terang, alih-alih membiarkan pembaca menyalin pola ini ke dalam sistem produksi.

4.8 Dari Skema ke Perangkat Lunak

Dengan skema yang kini konkret, bab selanjutnya melangkah kembali ke gambaran lengkapnya: bagaimana logika bisnis dan model data yang sama ini diekspresikan empat kali berbeda, dalam empat teknologi berbeda, di seluruh jalur Static, PHP, JSP, dan ASP.NET — apa yang dibagikan bersama oleh masing-masing, dan apa yang, secara jujur, dilakukan berbeda oleh masing-masing.

BAB 5

Empat Teknologi, Satu Bisnis: Tur Arsitektur

Apa yang diajarkan Static, PHP, JSP, dan ASP.NET masing-masing, dan mengapa keempatnya tidak dibangun serupa.

Profitina Laundry bukanlah satu aplikasi — ia adalah bisnis yang sama, diekspresikan secara independen sebanyak empat kali, satu kali untuk setiap teknologi sisi-server yang diajarkan mata kuliah Pemrograman Web (WebPro). Bab ini mengunjungi keempatnya pada tingkat tinjauan umum, sebagai proyek-proyek independen yang mandiri, bukan empat front-end untuk satu backend bersama, sebelum Bab 6 masuk lebih dalam ke yang paling kaya di antara keempatnya.

5.1 Static (Bab 2-3): Front End, Berdiri Sendiri

Implementasi paling awal secara sengaja dibuat tanpa backend: HTML5, CSS3, dan JavaScript murni, tanpa kode sisi-server dan tanpa basis data sama sekali. Tujuannya sempit dan spesifik — mengajarkan HTML semantik dan scripting sisi-klien secara terisolasi, sebelum sebuah server memasuki gambaran — dan ia berhasil mencapai itu justru karena tetap sesederhana ini. Implementasi ini membaca sejumlah kecil layanannya dari berkas `services.json` statis melalui `fetch()`, yang juga merupakan satu-satunya tempat ia membutuhkan server HTTP jenis apa pun, karena peramban memblokir `fetch()` pada halaman `file://` murni demi alasan keamanan.

5.2 PHP (Bab 5-7): Backend Sungguhan yang Pertama

Jalur PHP adalah tempat Profitina Laundry pertama kali menyentuh basis data sungguhan. Pengiriman formulir kontak dari pengunjung divalidasi, lalu ditulis melalui prepared statement PDO ke dalam tabel `messages` di MariaDB/MySQL, dan sebuah halaman terpisah, `view-messages.php`, membacanya kembali. Ruang lingkupnya sengaja dibuat lebih sempit dibandingkan jalur JSP dan ASP.NET — satu formulir, satu tabel — karena Bab 5-7 adalah tempat mata kuliah ini pertama kali memperkenalkan penanganan formulir sisi-server dan akses basis data, bukan tempat diajarkannya CRUD lengkap.

5.3 JSP (Bab 9-11): CRUD Lengkap, Gaya Jakarta EE

Jalur JSP/Servlet adalah implementasi create-read-update-delete lengkap pertama Profitina Laundry: `orders-list.jsp`, `order-create.jsp`, `order-edit.jsp`, dan `order-delete.jsp`, didukung oleh satu tabel `orders` yang diratakan (Bab 4, Bagian 4.6 membandingkan model statusnya dengan jalur ASP.NET). Bab 11 menambahkan data yang sama sebagai ekspor XML dan JSON yang dapat dibaca mesin, `orders-xml.jsp` dan `orders-json.jsp`, dengan demo `fetch()` sisi-klien yang langsung mengonsumsi JSON tersebut. Jalur ini berjalan di Apache Tomcat 10, yang penting secara spesifik karena Tomcat 10 adalah versi pertama yang menggunakan namespace `jakarta.servlet.*` modern, alih-alih namespace `javax.servlet.*` yang lebih lama — kode yang ditulis untuk Tomcat 10 tidak akan berjalan tanpa modifikasi di Tomcat 9 atau versi sebelumnya, dan buku ini mengatakannya secara terus terang, alih-alih membiarkan pembaca menemukannya sendiri dengan cara yang sulit.

5.4 ASP.NET (Bab 13-15): Implementasi Paling Kaya

Jalur ASP.NET Web Forms adalah implementasi Profitina Laundry yang paling lengkap, dan satu-satunya dari keempatnya yang menyasar skema kanonis lengkap `Bill` / `Customer` / `Employee` / `Service` dari Bab 4 secara langsung, dengan halaman CRUD khusus untuk masing-masing dari keempat entitas ditambah satu halaman pemesanan yang dibangun khusus, `NewBill.aspx`, yang menegakkan aturan penugasan karyawan berbasis-keseimbangan-beban yang dijelaskan Bab 3. Jalur ini juga merupakan satu-satunya dari keempat implementasi dengan keterbatasan lintas platform yang nyata dan diungkapkan secara terbuka, yang dibahas Bab 6 secara langsung, alih-alih diabaikan begitu saja.

5.5 Berdampingan

Jalur	Bab	Teknologi	Model data
Static	2-3	HTML5 / CSS3 / JavaScript	Tidak ada (services.json statis)
PHP	5-7	PHP + PDO + MySQL/MariaDB	Satu tabel messages
JSP	9-11	JSP/Servlets + Tomcat 10 + MariaDB	Tabel orders yang diratakan
ASP.NET	13-15	ASP.NET Web Forms + SQLite	Bill / Customer / Employee / Service

5.6 Mengapa Empat Implementasi, Bukan Satu

Sebuah mata kuliah yang mengajarkan empat teknologi sisi-server berbeda, pada prinsipnya, dapat mengajarkan keempatnya terhadap satu studi kasus bersama yang dibangun sekali dan sekadar diganti kulitnya setiap saat. Profitina Laundry secara sengaja tidak melakukan ini, dengan alasan yang layak dinyatakan secara langsung: setiap teknologi diajarkan pada titik yang berbeda dalam mata kuliah, pada tingkat kecanggihan yang berbeda, dan membangun setiap implementasi agar benar-benar sesuai dengan blok babnya masing-masing — alih-alih memaksa setiap jalur ke satu skema sejak hari pertama — memungkinkan setiap jalur mengajarkan apa yang sebenarnya dibahas bab-babnya. Jalur Static dapat tetap menjadi latihan front-end murni tanpa backend yang canggung dan tidak terpakai yang ditempelkan padanya; jalur PHP dapat tetap berfokus pada satu formulir dan satu tabel tanpa berpura-pura menjadi CRUD lengkap sebelum mata kuliah siap mengajarkannya.

> **Biaya dari pilihan desain ini, secara jujur**
>
> Biayanya persis merupakan divergensi skema yang telah diungkapkan Bab 4, Bagian 4.6: tabel orders jalur JSP dan skema Bill/Customer/Employee/Service jalur ASP.NET bukanlah bentuk yang sama, dan seorang mahasiswa yang berpindah dari Bab 10 ke Bab 13 akan menyadari bahwa model datanya berubah di bawah mereka. Itu adalah tindak lanjut nyata yang dilacak untuk proyek ini (menyelaraskan keduanya, bersamaan dengan memperbarui teks buku bab-bab yang terkait, bukan hanya kodenya) alih-alih sebuah kelalaian — lihat bab penutup buku ini untuk mengetahui posisi hal tersebut saat ini.

5.7 Selanjutnya

Bab 6 masuk lebih dalam ke jalur ASP.NET secara spesifik: halaman-halamannya, aturan bisnisnya, dan model state tiga-tingkat ViewState/Session/Application yang menjadi andalan halaman pemesanannya. Bab 7 kemudian menceritakan kisah bug nyata yang ditemukan proyek ini pada kode ASP.NET-nya sendiri, dan persis bagaimana bug itu didiagnosis dan diperbaiki.

BAB 6

Di Dalam Aplikasi ASP.NET

Halaman, aturan bisnis, dan tiga cara untuk menyimpan hitungan.

Dari keempat implementasi Profitina Laundry, aplikasi ASP.NET Web Forms adalah yang paling lengkap: ia menyasar skema kanonis penuh dari Bab 4, dan merupakan satu-satunya jalur dengan alur kerja pemesanan yang dibangun khusus, bukan sekadar CRUD biasa. Bab ini menelusuri apa yang sebenarnya dilakukannya, halaman demi halaman, serta aturan bisnis yang ditegakkannya di sepanjang jalan.

6.1 Satu Master Page Bersama

Keenam halaman — Default (dasbor), Services, Customers, Employees, Bills, dan NewBill — berbagi satu tata letak `Site.master`: navigasi yang konsisten, penataan gaya yang konsisten (`app.css`), dan satu tempat untuk mengubah tampilan situs tanpa menyentuh setiap halaman satu per satu. Ini adalah jawaban ASP.NET Web Forms sendiri terhadap masalah tata letak bersama yang dipecahkan secara berbeda oleh masing-masing dari tiga jalur Profitina Laundry lainnya (sebuah `<header>` bersama pada HTML murni jalur Static dan PHP, navigasi bab-demi-bab `index.jsp` sendiri pada jalur JSP).

6.2 Empat Halaman CRUD, Satu Pola Bersama

Services.aspx, Customers.aspx, Employees.aspx, dan Bills.aspx masing-masing mengikuti tata letak yang sama: sebuah GridView yang mendaftar baris-baris yang ada dengan perintah Edit dan Delete inline, di atas formulir "Tambah X Baru" yang terpisah untuk membuat baris baru. Masing-masing menegakkan aturan yang dibangun Bab 4 langsung ke dalam skema itu sendiri — upaya menghapus Service, Customer, atau Employee yang masih dirujuk oleh sebuah Bill yang ada ditangkap dan dilaporkan kembali ke pengguna sebagai pesan yang jelas, alih-alih dibiarkan muncul sebagai pengecualian basis data mentah.

Employees.aspx juga memperlihatkan relasi supervisor yang self-referencing dari Bab 4, Bagian 4.3 secara langsung: formulir Tambah-dan-Edit-nya menyertakan dropdown supervisor yang diisi dari tabel Employee itu sendiri, dan menghapus seorang karyawan yang mengawasi langsung tercermin di setiap rekaman karyawan yang menunjuk kepadanya, persis seperti aturan `ON DELETE SET NULL` skema tersebut.

6.3 NewBill.aspx: Tempat Aturan Bisnis Hidup

NewBill.aspx adalah halaman pemesanan Profitina Laundry, dan di sinilah aturan-aturan operasional Bab 3 berhenti menjadi abstrak dan menjadi logika aplikasi yang ditegakkan.

- Pembayaran yang dihitung server — jumlah yang harus dibayar pelanggan dihitung di server dari harga layanan yang dipilih dan jumlah unit yang dimasukkan, tidak pernah dipercayakan dari input yang dikirim klien. Seorang pelanggan tidak dapat memengaruhi berapa yang ditagihkan kepadanya dengan mengutak-atik formulir.

- Penugasan karyawan berbasis-keseimbangan-beban — ketika sebuah tagihan baru dibuat, karyawan meja depan ("transaction") dengan jumlah tagihan yang antre paling sedikit ditugaskan secara otomatis, alih-alih selalu default ke karyawan yang sama. Ini adalah pengamatan penempatan staf Bab 3, ditegakkan dalam kode alih-alih diserahkan kepada karyawan mana pun yang kebetulan sedang login.
- Pengenalan pelanggan tetap — seorang pelanggan dicari berdasarkan nomor telepon sebelum sebuah baris Customer baru dibuat; nomor telepon yang sudah tercatat menggunakan kembali rekaman yang ada alih-alih membuat duplikat. Meja laundry sungguhan mengenali pelanggan tetap dari lebih sekadar ejaan nama yang persis, dan halaman ini melakukan hal yang sama.
- Batasan kuantitas khusus-layanan, ditegakkan oleh sebuah CustomValidator — pesanan untuk layanan Setrika Saja ("SS") dibatasi pada jumlah unit maksimum yang masuk akal; pesanan untuk layanan tersebut yang dikirim jauh di atas batas (diuji pada 25 unit) ditolak sebelum mencapai basis data, sementara pesanan berukuran normal untuk layanan yang sama (diuji pada 15 unit) diterima. Satu pesanan setrika-saja untuk puluhan unit sangat tidak mungkin sungguhan, dan menolaknya lebih awal menangkap kesalahan entri data sebelum menjadi sebuah tagihan.

6.4 Tiga Tempat untuk Menyimpan Hitungan: ViewState, Session, dan Application

NewBill.aspx juga melakukan sesuatu yang tidak dibutuhkan kebanyakan halaman CRUD: ia menyimpan tiga penghitung berjalan terpisah tentang berapa banyak tagihan yang telah dipesan, masing-masing disimpan dalam mekanisme manajemen-state ASP.NET Web Forms yang berbeda, secara khusus agar perbedaan di antara ketiganya terlihat, bukan hanya teoretis.

Mekanisme	Menghitung	Bertahan hingga
ViewState	Pemesanan yang dibuat pada instance halaman ini secara spesifik	Postback ke halaman yang sama, direset saat pemuatan halaman baru
Session	Pemesanan yang dibuat oleh pengunjung spesifik ini	Setiap permintaan dari peramban sesi yang sama, direset saat sesi berakhir
Application	Pemesanan yang dibuat oleh semua orang, di seluruh pengunjung	Masa hidup aplikasi yang sedang berjalan itu sendiri

Ini adalah sesuatu yang sungguh-sungguh berguna untuk dilihat dalam halaman nyata, bukan hanya slide: satu pemesanan yang berhasil menambah ketiga penghitung sekaligus, tetapi ketiganya berbeda begitu pengunjung kedua memesan sesuatu — pemesanan kedua tersebut menambah Session dan Application untuk sesi pengunjung tersebut, sementara penghitung Session pengunjung pertama tetap persis di tempatnya semula, dan hanya Application yang bergerak untuk keduanya. Memverifikasi bahwa hal ini sungguh-sungguh terjadi (bukan sekadar mengasumsikan dokumentasi ASP.NET benar) membutuhkan infrastruktur pengujian lintas-permintaan yang nyata, dijelaskan singkat pada catatan pengujian Bab 7.

6.5 Kenyataan Lintas Platform, Dinyatakan secara Terus Terang

> **Ini adalah keterbatasan nyata, bukan kelalaian**
>
> ASP.NET Web Forms klasik — API `System.Web` yang menjadi dasar penulisan aplikasi ini — tidak pernah diporting ke .NET modern yang lintas platform (.NET 5/6/7/8 dan seterusnya). Ia berjalan secara native hanya di Windows dengan IIS atau IIS Express dan .NET Framework. Di Linux atau macOS, satu-satunya cara menjalankannya sama sekali adalah Mono, sebuah implementasi ulang pihak ketiga yang independen dari .NET Framework, dan web host ringan milik Mono sendiri diketahui tidak dapat diandalkan untuk Web Forms secara spesifik pada beberapa kombinasi Mono/OS. Buku ini menyatakan hal itu secara terus terang, alih-alih mengklaim kisah lintas platform yang palsu, dan panduan pengaturan Bab 8 memberikan Windows 11 sebagai jalur kelas satu, dengan Mono dan mesin virtual Windows sebagai dua opsi jujur untuk macOS dan Linux.

Karena itu, menguji perilaku nyata aplikasi ini pada lingkungan pengembangan Linux milik proyek ini sendiri membutuhkan sebuah substitusi kecil yang dibangun khusus — tidak pernah dikirimkan sebagai bagian dari aplikasi sesungguhnya — dijelaskan secara jujur dalam `VALIDATION_LOG.txt` milik jalur ini sendiri dan dirujuk kembali di Bab 7.

6.6 Selanjutnya

Bab 7 adalah satu kisah tunggal yang berfokus: sebuah bug nyata yang dimiliki kode ASP.NET proyek ini sendiri, bersembunyi di depan mata di setiap satu dari keempat halaman CRUD yang dijelaskan Bagian 6.2, dan proses persis yang menemukan, menjelaskan, dan memperbaikinya.

BAB 7

Bug ValidationGroup: Studi Kasus Debugging

Kegagalan GridView yang senyap, dan proses yang benar-benar menemukannya.

Setiap klaim eksekusi-nyata dalam buku ini bersandar pada sesuatu yang benar-benar terjadi, dan bab ini membahas hal paling mendidik yang terjadi selama pembangunan jalur ASP.NET: sebuah bug sungguhan, ditemukan pada kode proyek ini sendiri, yang secara diam-diam merusak tombol Update setiap GridView pada setiap satu dari keempat halaman CRUD yang dijelaskan Bab 6. Bab ini menceritakan kisah tersebut secara lengkap — bukan karena bugnya sendiri eksotis, melainkan karena bagaimana ia ditemukan merupakan pelajaran debugging yang layak dimuat dalam sebuah buku, bukan hanya dalam pesan commit.

7.1 Gejalanya

Mengklik "Update" pada sebuah baris di GridView Services, Customers, Employees, atau Bills tampak tidak melakukan apa-apa. Tidak ada kesalahan yang muncul di layar. Halaman melakukan postback — sebuah round-trip jaringan jelas terjadi — dan grid tersebut sekadar dirender ulang dengan nilai-nilai asli baris yang belum diedit masih di tempatnya. Tidak ada apa pun dari perilaku yang terlihat yang menunjuk pada penyebab tertentu: tidak ada pengecualian, tidak ada pesan validasi, tidak ada teks merah di mana pun.

7.2 Dugaan Pertama yang Keliru

Hal pertama yang layak diperiksa tampak, sejenak, seperti penjelasannya: HTML yang dirender untuk panggilan postback tombol Update berbeda tergantung pada sebuah pengaturan bernama `CausesValidation`. Dengan pengaturan tersebut dibiarkan pada nilai bawaannya (`true`), tombol tersebut melakukan postback melalui ID kontrolnya sendiri yang dihasilkan secara unik dengan argumen kosong; diubah menjadi `false`, ia melakukan postback melalui ID GridView itu sendiri dengan argumen berindeks yang mengidentifikasi baris dan perintah mana yang dipicu. Itu adalah perbedaan nyata dan sah dalam cara ASP.NET merender kedua konfigurasi tersebut — tetapi itu adalah detail rendering, bukan bug, dan mengejarnya sebagai akar penyebab akan menjadi jalan buntu. Hal ini didokumentasikan dalam `VALIDATION_LOG.txt` jalur ini secara spesifik agar tidak keliru dianggap sebagai cacat sesungguhnya oleh pembaca di masa depan yang membuat dugaan pertama yang sama.

7.3 Mengisolasi Masalahnya

Alih-alih terus menebak-nebak di dalam aplikasi lengkap, langkah berikutnya adalah membangun hal sekecil mungkin yang dapat mereproduksi gejala tersebut: sebuah halaman mandiri minimal tanpa apa pun kecuali sebuah GridView dan sebuah perintah Update. Halaman minimal tersebut bekerja dengan sempurna — `RowUpdating` terpicu setiap saat, nilai-nilai tersimpan dengan benar. Halaman minimal yang sama, dibangun ulang terhadap sebuah master page bersama yang menyerupai Site.master, tetap bekerja. Hanya ketika sebuah `RequiredFieldValidator` yang tidak terisi dan tidak terkait ditambahkan di tempat lain pada halaman yang sama — memodelkan formulir "Tambah X

Baru" aplikasi sesungguhnya yang duduk berdampingan dengan grid — gejala tersebut baru tereproduksi persis: perintah Update secara diam-diam berhenti terpicu.

> **Mengapa membangun reproduksi minimal itu penting di sini**
>
> Aplikasi lengkap memiliki master page, banyak GridView, banyak formulir, dan cukup banyak kode di sekitarnya — banyak tempat yang masuk akal untuk menyembunyikan sebuah bug, dan banyak gangguan untuk disaring saat mencarinya. Sebuah reproduksi minimal yang sengaja dipersempit menghilangkan semua gangguan itu sekaligus: begitu halaman kosong dengan tidak lebih dari sebuah GridView dan satu validator tak terkait mereproduksi gejala yang persis sama, ruang pencarian menyempit dari "suatu tempat di empat halaman kode" menjadi "interaksi spesifik ini antara sebuah tombol perintah GridView dan sebuah validator yang tidak ada hubungannya dengannya".

7.4 Akar Penyebab Sesungguhnya

Setiap halaman CRUD memasangkan sebuah GridView (dengan `CommandField` yang dapat diedit) dengan formulir "Tambah X Baru" yang terpisah pada halaman yang sama, dan — dalam kode sebagaimana awalnya ditulis — tidak satu pun yang menetapkan sebuah `ValidationGroup`. Perilaku bawaan ASP.NET Web Forms, ketika tidak ada grup yang ditetapkan di mana pun pada sebuah halaman, adalah memvalidasi setiap validator pada halaman tersebut bersama-sama sebagai satu grup implisit tunggal, terlepas dari formulir mana secara visual sebuah tombol berada.

Tombol Update GridView memiliki `CausesValidation="true"` secara bawaan. Mengkliknya karenanya memicu `Page.Validate()` terhadap grup bawaan tunggal tersebut — yang mencakup kontrol `RequiredFieldValidator` milik formulir Tambah, yang secara alami kosong selama pengeditan grid-saja, karena pengguna sedang mengedit baris yang ada, bukan mengisi formulir Tambah sama sekali. Validasi gagal. Dan perilaku standar ASP.NET Web Forms ketika validasi sebuah tombol `CausesValidation="true"` gagal adalah melewati handler Click/Command tombol tersebut sepenuhnya — yang berarti `RowUpdating` tidak pernah berjalan, tanpa pengecualian dan tanpa kesalahan yang terlihat, karena dari sudut pandang framework tidak ada yang salah: ia dengan benar menolak untuk menjalankan sebuah handler di balik pemeriksaan validasi yang gagal.

```
<%-- Sebelum perbaikan: tidak ada ValidationGroup di mana pun pada halaman --%>
<asp:GridView ID="GridView1" runat="server" >
  <Columns>
    <asp:CommandField ShowEditButton="true" ShowDeleteButton="true" />
  </Columns>
</asp:GridView>

<%-- Formulir Tambah yang tidak terkait, halaman yang sama, grup validasi bawaan
yang sama --%>
<asp:RequiredFieldValidator runat="server" ControlToValidate="txtNewName"  />
<asp:Button ID="btnAdd" runat="server" Text="Add" />
```

7.5 Mengapa Bug Ini Mudah Ditulis dan Sulit Disadari

Tidak ada apa pun dari kode ini yang terlihat salah saat dibaca sekilas. Baik GridView maupun formulir Tambah adalah markup Web Forms yang benar dan idiomatis dengan sendirinya; cacatnya hanya ada

dalam interaksi antara keduanya, dan hanya karena grup validasi bawaan lebar-halaman ASP.NET adalah perilaku implisit yang mudah dilupakan, alih-alih sesuatu yang biasa dituliskan seorang pengembang secara eksplisit. Sebuah halaman dengan hanya GridView, atau hanya formulir Tambah, tidak akan pernah menunjukkan gejala ini — dibutuhkan keduanya, pada halaman yang sama, tanpa pengelompokan eksplisit, yang persis merupakan bentuk yang kebetulan dimiliki setiap satu dari keempat halaman CRUD Profitina Laundry.

7.6 Perbaikannya

Perbaikannya adalah membuat pengelompokan validasi eksplisit alih-alih bersandar pada bawaan implisit: setiap validator, `ValidationSummary`, dan tombol milik formulir Tambah mendapatkan `ValidationGroup` eksplisit, dan `CommandField` GridView (bersama kontrol lain mana pun yang dapat memicu validasi yang tidak diinginkan, seperti dropdown layanan auto-postback milik Bills.aspx) mendapatkan `CausesValidation="False"`, karena pengeditan baris grid sama sekali tidak membutuhkan validasi tingkat-halaman dipicu atas namanya.

```
<%-- Setelah perbaikan --%>
asp:CommandField ShowEditButton="true" ShowDeleteButton="true"
                 CausesValidation="False" />

asp:RequiredFieldValidator runat="server" ControlToValidate="txtNewName"
                           ValidationGroup="AddService" ... />
asp:Button ID="btnAdd" runat="server" Text="Add"
           ValidationGroup="AddService" />
```

Diterapkan di seluruh Services.aspx, Customers.aspx, Employees.aspx, dan Bills.aspx (masing-masing dengan nama grupnya sendiri — `AddService`, `AddCustomer`, `AddEmployee`, `AddBill`), perbaikan ini memulihkan persis perilaku yang selalu dimaksudkan dimiliki halaman-halaman tersebut: mengedit sebuah baris grid tidak lagi menyentuh validator formulir Tambah yang tidak terkait sama sekali, dan `RowUpdating` terpicu setiap saat.

7.7 Memverifikasi Perbaikan Itu Nyata, Bukan Diasumsikan

Sebuah perbaikan yang sekadar terlihat benar bukanlah hal yang sama dengan perbaikan yang dikonfirmasi bekerja. Setiap halaman yang terpengaruh diuji ulang dari ujung ke ujung setelah perubahan tersebut: menambah baris, mengedit baris melalui grid, menghapus baris, dan membaca basis data SQLite yang mendasarinya secara langsung setelah setiap operasi untuk mengonfirmasi bahwa perubahan tersebut benar-benar tersimpan, alih-alih mempercayai respons halaman itu sendiri. Keempat halaman lolos, di setiap operasi, dan satu detail tersisa yang ditandai Bagian 7.2 — pola postback tombol Update yang berubah dari bentuk ID-sendiri menjadi bentuk berindeks-GridView begitu `CausesValidation` menjadi `False` — dikonfirmasi sebagai konsekuensi rendering yang diharapkan dari perbaikan tersebut, bukan masalah baru, dan skrip pengujian proyek ini diperbarui untuk mengenali kedua pola tersebut, alih-alih mengasumsikan hanya satu.

Mengonfirmasi bahwa perbaikan tersebut tidak merusak penghitung ViewState/Session/Application NewBill.aspx (Bab 6, Bagian 6.4) membutuhkan sesuatu yang ekstra: ketiga penghitung tersebut hanya berbeda secara bermakna di seluruh banyak permintaan dalam aplikasi yang sama yang sedang berjalan, sesuatu yang tidak dapat diuji oleh pengujian satu-permintaan-per-panggilan yang

sederhana. Itu membutuhkan pembangunan tambahan kecil pada test harness proyek ini — sebuah mode yang menjaga satu instance aplikasi ASP.NET tetap hidup di seluruh aliran permintaan, sehingga state Session dan Application dapat diamati sungguh-sungguh berubah, di seluruh permintaan berturut-turut yang nyata, alih-alih diasumsikan dari dokumentasi.

7.8 Pelajaran yang Lebih Luas

Cacat spesifik di sini — sebuah `ValidationGroup` yang tidak dilingkupi yang bertabrakan di antara dua formulir tak terkait pada halaman yang sama — adalah jebakan ASP.NET Web Forms yang nyata dan sudah dikenal luas, layak diketahui dengan sendirinya. Tetapi pelajaran yang lebih tahan lama adalah prosesnya: gejala yang samar dan tidak membantu ("tidak terjadi apa-apa, tidak ada kesalahan") diselesaikan bukan dengan menatap lebih keras pada aplikasi lengkap, melainkan dengan membangun reproduksi yang semakin kecil hingga pemicu minimal yang tepat terlihat, dan perbaikan tersebut kemudian diverifikasi terhadap basis data nyata dan perilaku HTTP nyata, alih-alih dipercaya hanya dari pemeriksaan sekilas. Proses itu berlaku jauh melampaui ASP.NET, dan pembaca yang men-debug tumpukan teknologi yang sama sekali berbeda akan mengenali bentuk investigasi yang sama.

7.9 Selanjutnya

Bab 8 adalah panduan praktis untuk menjalankan sendiri salah satu dari keempat implementasi Profitina Laundry, di Windows 11, macOS, atau Linux.

BAB 8

Memulai: Menjalankan Profitina Laundry Sendiri

Pengaturan Windows 11, macOS, dan Linux untuk keempat implementasi.

Bab ini adalah pendamping praktis, bukan manual referensi — bab ini menjalankan masing-masing dari keempat implementasi Profitina Laundry di Windows 11, macOS, atau Linux, dengan catatan platform yang sama jujurnya yang telah disampaikan sekilas oleh Bab 5 dan 6. Versi lengkap dan selalu terkini dari instruksi ini dikirimkan bersama kodenya sendiri, dalam `README.md` tingkat-atas paket aplikasi tersebut; bab ini meringkasnya bagi pembaca buku.

8.1 Static (Bab 2-3)

Tidak ada backend, tidak ada langkah build, dan tidak ada server yang dibutuhkan dalam arti ketat — membuka `Static/index.html` secara langsung di peramban modern mana pun bekerja secara identik di ketiga platform. Satu pengecualian adalah panggilan `fetch()` pada halaman tersebut, yang diblokir peramban pada halaman `file://` murni demi alasan keamanan; menyajikan foldernya sebagai gantinya sepenuhnya menghindari hal ini.

```
cd Static
python3 -m http.server 8000
# lalu buka http://localhost:8000/index.html
```

Satu perintah ini identik di Windows 11, macOS, dan Linux, karena Python 3 menyertakan server bawaan yang sama di mana pun. Ekstensi "Live Server" VS Code, `npx serve`, atau `php -S` milik PHP sendiri semuanya bekerja dengan cara yang sama, jika salah satunya kebetulan sudah terpasang.

8.2 PHP (Bab 5-7)

1. Pasang PHP 8.x dan server yang kompatibel dengan MySQL: XAMPP di Windows 11 (menggabungkan PHP, MariaDB/MySQL, dan Apache dalam satu installer); `brew install php mariadb` di macOS, atau MAMP sebagai alternatif paket; `sudo apt install php php-mysql mariadb-server` di Ubuntu/Debian.
2. Buat basis data dan muat skemanya (identik di setiap platform setelah langkah 1 selesai):

```
mysql -u root -p -e "CREATE DATABASE profitina_laundry CHARACTER SET utf8mb4;
  CREATE USER 'webpro'@'127.0.0.1' IDENTIFIED BY 'webpro_pass123';
  GRANT ALL PRIVILEGES ON profitina_laundry.* TO 'webpro'@'127.0.0.1'; FLUSH
PRIVILEGES;"
mysql -u webpro -pwebpro_pass123 -h 127.0.0.1 profitina_laundry < PHP/schema.sql
```

3. Sajikan foldernya dengan server bawaan PHP sendiri dan buka formulir kontak:

```
cd PHP
php -S localhost:8091
# lalu buka http://localhost:8091/contact-form.html
```

8.3 JSP (Bab 9-11)

4. Pasang JDK (versi 11 atau lebih baru) dan Apache Tomcat 10.x secara spesifik — Tomcat 10 menggunakan namespace `jakarta.servlet.*` yang menjadi sasaran kode ini; Tomcat 9 dan versi sebelumnya menggunakan namespace `javax.servlet.*` yang lebih lama dan tidak kompatibel, dan tidak akan menjalankan kode ini tanpa modifikasi. Di Windows 11: sebuah JDK seperti Eclipse Temurin, ditambah distribusi ZIP Tomcat 10 (tanpa installer yang dibutuhkan). Di macOS: `brew install openjdk@21 tomcat@10`. Di Ubuntu/Debian: `sudo apt install openjdk-21-jdk tomcat10`.
5. Letakkan MariaDB Connector/J pada classpath bersama Tomcat (folder `lib/`-nya di setiap platform), baik diunduh langsung dari rilis Connector/J MariaDB, atau melalui `sudo apt install libmariadb-java` di Ubuntu/Debian, yang memasangnya langsung ke `lib/` bersama milik Tomcat sistem.
6. Buat basis data dan muat skemanya:

```
mysql -u root -p -e "CREATE DATABASE profitina_laundry CHARACTER SET utf8mb4;
  CREATE USER 'webpro'@'127.0.0.1' IDENTIFIED BY 'webpro_pass123';
  GRANT ALL PRIVILEGES ON profitina_laundry.* TO 'webpro'@'127.0.0.1'; FLUSH
PRIVILEGES;"
mysql -u webpro -pwebpro_pass123 -h 127.0.0.1 profitina_laundry < JSP/schema.sql
```

7. Salin seluruh folder JSP/ ke dalam direktori webapps/ milik Tomcat dengan nama "profitina", jalankan Tomcat, lalu buka http://localhost:8080/profitina/index.jsp.

8.4 ASP.NET (Bab 13-15)

> **Baca ini sebelum memilih jalur**
>
> ASP.NET Web Forms klasik tidak pernah diporting ke .NET modern yang lintas platform — ia hanya pernah berjalan secara native di Windows dengan IIS atau IIS Express dan .NET Framework, atau di bawah Mono (sebuah implementasi ulang pihak ketiga yang independen) di Linux atau macOS. Melebih-lebihkan "lintas platform" di sini akan tidak jujur, sehingga buku ini tidak melakukannya.

Windows 11 — jalur kelas satu yang didukung penuh

8. Pasang Visual Studio 2022 dengan beban kerja "ASP.NET and web development" — ini memasang IIS Express dan targeting pack .NET Framework secara bersamaan.
9. Pasang paket NuGet System.Data.SQLite.Core ke dalam proyek.
10. Buat basis data SQLite sekali, dari Schema.sqlite.sql, menggunakan alat SQLite apa pun (CLI sqlite3, atau DB Browser for SQLite).
11. Buka proyeknya di Visual Studio dan tekan F5 — IIS Express menyajikannya secara langsung, tanpa pemasangan server web terpisah yang dibutuhkan.

macOS — dua opsi jujur, tidak satu pun yang "native"

- Mono (`brew install mono`): opsi yang sungguhan, tetapi web host ringan milik Mono diketahui tidak dapat diandalkan untuk Web Forms pada beberapa kombinasi Mono/OS — pengujian proyek ini sendiri menemukan persis hal itu, dan membangun test harness khusus-pengujian alih-alih mengandalkannya (lihat Bab 7). Anggarkan waktu troubleshooting yang sungguhan.
- Mesin virtual Windows (Parallels, UTM, atau VMware, di Apple Silicon atau Intel): pasang Windows 11 di dalam VM dan ikuti langkah-langkah Windows di atas. Ini adalah jalur yang paling mungkin berhasil pada percobaan pertama, dan yang secara historis dilakukan kebanyakan pengembang Web Forms pengguna Mac di dunia nyata.

Linux

Jalur Mono yang sama seperti macOS di atas, karena Mono sendiri lintas platform — inilah persis bagaimana pengujian eksekusi-nyata proyek ini sendiri dilakukan, termasuk substitusi khusus-pengujian dari `Mono.Data.Sqlite` untuk `System.Data.SQLite` yang hanya dibutuhkan build pengujian; kode yang dikirimkan menyasar penyedia Windows/.NET Framework sesungguhnya tanpa modifikasi.

8.5 Di Mana Menemukan Semua Ini di Disk

Keempat implementasi, ditambah versi lengkap dari setiap instruksi dalam bab ini, dikirimkan bersama sebagai satu paket mandiri — folder Static, PHP, JSP, dan ASPNET berdampingan, masing-masing dengan `VALIDATION_LOG.txt`-nya sendiri yang mengungkapkan persis bagaimana ia diuji. Pembaca yang ingin menjalankan kodenya berdampingan dengan buku ini sebaiknya memulai dari `README.md` milik paket tersebut sendiri, alih-alih mengetik ulang apa pun dari bab ini secara manual.

BAB 9

Pelajaran di Luar Kode

Apa yang ditawarkan studi kasus ini bagi pengajar, pemilik bisnis, dan mahasiswa sekaligus.

Bab 1 menyebutkan tiga pembaca sasaran buku ini — mahasiswa, pengajar, dan pemilik bisnis atau masyarakat umum — dan bab-bab sejak itu sebagian besar berbicara kepada yang pertama dari ketiganya. Bab ini melangkah mundur dan berbicara lebih langsung kepada dua lainnya, menarik keluar apa yang diajarkan Profitina Laundry di luar kode sumbernya sendiri.

9.1 Untuk Pengajar: Studi Kasus yang Dibangun untuk Diadaptasi, Bukan Sekadar Dibaca

Pilihan-pilihan spesifik di balik Profitina Laundry, dalam artian yang sesungguhnya, lebih mudah dipindahkan ke mata kuliah lain dibandingkan kodenya sendiri. Seorang pengajar yang membangun studi kasus berjalan miliknya sendiri untuk sekumpulan teknologi yang berbeda dapat meminjam keputusan mendasar yang sama secara langsung: landaskan studi kasus pada sesuatu yang nyata alih-alih mengarangnya dari nol, sehingga secara alami memunculkan kompleksitas sungguhan (relasi supervisor Bab 3, aturan perlindungan-hapus Bab 4) alih-alih hanya kompleksitas yang diantisipasi rencana pelajaran; biarkan implementasi setiap teknologi benar-benar sesuai dengan apa yang diajarkan blok mata kuliah tersebut, alih-alih memaksa setiap implementasi ke satu skema sejak hari pertama, dan ungkapkan divergensi yang dihasilkan secara terbuka alih-alih menyembunyikannya (Bab 5); serta pegang setiap klaim dalam materi pendamping pada standar eksekusi-nyata yang sama yang diterapkan buku ini pada dirinya sendiri — klaim bahwa suatu kode bekerja seharusnya didukung oleh log yang menunjukkannya benar-benar berjalan, dan sebuah bug yang ditemukan di sepanjang jalan sebaiknya dituliskan sebagai pelajaran, bukan diam-diam diperbaiki dan dilupakan.

> **Gagasan paling dapat dipindahkan dalam buku ini**
>
> Jika ada satu gagasan dari buku ini yang layak dibawa ke mata kuliah yang sama sekali berbeda, inilah itu: sebuah studi kasus yang berlandaskan bisnis nyata dan benar-benar dijalani menghasilkan materi pembelajaran yang lebih baik dibandingkan yang dikarang murni untuk menyesuaikan pelajaran sintaks, karena bisnis nyata itu membawa kasus-kasus tepi sungguhannya sendiri — secara cuma-cuma, tanpa perlu seorang pun mengarangnya demi efek dramatis.

9.2 Untuk Pemilik Bisnis: Apa yang Sebenarnya Terlibat dalam "Digitalisasi"

Seorang pemilik bisnis kecil yang membaca buku ini tanpa tertarik sama sekali pada kodenya tetap dapat membawa pulang sesuatu yang nyata darinya: contoh kerja yang jujur tentang apa yang sebenarnya dibutuhkan untuk mengubah operasi yang berjalan di atas kertas dan ingatan menjadi perangkat lunak, tanpa bingkai promosi penjualan yang biasanya menyelimuti percakapan tersebut.

- Hal ini membutuhkan penamaan yang presisi atas apa yang sudah dicatat bisnis tersebut secara informal — empat kategori Bab 3 (pelanggan, karyawan, layanan, dan pesanan)

sudah ada di setiap bisnis laundry, baik ada yang menuliskannya sebagai daftar formal atau tidak.

- Hal ini membutuhkan pembuatan aturan tak tertulis menjadi eksplisit — staf sebuah bisnis sudah tahu, dari kebiasaan, bahwa layanan yang sudah dibebankan ke seseorang seharusnya tidak begitu saja hilang dari daftar harga, atau bahwa pelanggan baru sebaiknya diarahkan ke siapa pun yang belum sibuk. Perangkat lunak harus diberi tahu hal-hal ini secara langsung, dalam bentuk yang cukup presisi bagi sebuah mesin untuk menegakkannya (aturan foreign-key Bab 4, logika penyeimbangan-beban Bab 6).
- Hal ini membutuhkan penerimaan bahwa versi pertama akan memiliki bug yang sungguhan, dan bahwa menemukan serta memperbaikinya secara jujur (Bab 7) adalah bagian normal dan sehat dari membangun sesuatu yang nyata — bukan tanda bahwa proyeknya salah arah.
- Hal ini membutuhkan pemilihan platform dengan mata terbuka terhadap trade-off-nya, sebagaimana dilakukan Bab 6 dan Bab 8 untuk keterbatasan lintas platform ASP.NET — sebuah keputusan teknologi yang sungguhan selalu memiliki biaya sungguhan di suatu tempat, dan berpura-pura sebaliknya pada akhirnya akan mengejar sebuah bisnis.

Tidak satu pun dari ini membutuhkan pemahaman satu baris pun kode dalam buku ini. Hal ini hanya membutuhkan kesediaan untuk melihat dengan saksama bagaimana bisnis tersebut sebenarnya berjalan hari ini, yang, bukan kebetulan, persis apa yang dibutuhkan untuk membangun Profitina Laundry itu sendiri.

9.3 Untuk Mahasiswa: Membaca Kode Nyata, Termasuk Cacatnya

Kebanyakan kode yang ditemui mahasiswa dalam sebuah mata kuliah disajikan sudah benar, karena menunjukkan sebuah bug dan perbaikannya membutuhkan lebih banyak ruang dibandingkan hanya menunjukkan perbaikannya saja. Profitina Laundry secara sengaja melakukan sebaliknya di Bab 7, dan alasannya layak dinyatakan secara langsung di sini: pengembangan perangkat lunak profesional yang sesungguhnya bukanlah proses linear yang rapi yang diimplikasikan sebuah contoh buku teks yang sudah jadi. Bug ditemukan pada kode yang tampak benar saat dibaca sekilas, kadang berminggu-minggu setelah ditulis, dan keterampilan sesungguhnya yang diajarkan Bab 7 bukanlah "bagaimana ValidationGroup ASP.NET bekerja" melainkan lebih kepada "bagaimana mengisolasi gejala yang tidak jelas menjadi reproduksi minimal, dan bagaimana memverifikasi sebuah perbaikan itu nyata, bukan sekadar masuk akal." Keterampilan itu berlaku pada setiap teknologi yang akan pernah dikerjakan seorang mahasiswa, jauh setelah detail spesifik ASP.NET terlupakan.

9.4 Satu Pelajaran Umum, Dinyatakan Sekali Lagi

Bab 1 menjanjikan komitmen pada kejujuran — log eksekusi nyata, keterbatasan yang diungkapkan, bug yang disebutkan alih-alih disembunyikan — dan setiap bab sejak itu telah berusaha menjaga janji itu secara konkret, bukan hanya sebagai nilai yang dinyatakan. Itu, lebih dari keputusan teknologi atau skema tertentu mana pun, adalah apa yang paling ingin dibawa pulang buku ini oleh pembaca dari kelompok mana pun di antara ketiganya: bahwa dokumentasi yang jujur tentang apa yang sebenarnya terjadi, termasuk bagian-bagian yang tidak berjalan mulus pada percobaan pertama, lebih berharga daripada kisah yang dipoles dari sebuah proses yang tidak pernah sungguh-sungguh terjadi seperti itu.

9.5 Yang Tersisa

Bab terakhir menutup buku ini: di mana posisi Profitina Laundry hari ini, dan ke mana arahnya selanjutnya.

BAB 10

Penutup: Ke Mana Profitina Laundry Selanjutnya

Apa yang masih terbuka, dinyatakan secara jujur, dan apa yang datang setelah buku ini.

Setiap bab sebelum ini menjelaskan Profitina Laundry sebagaimana adanya hari ini: empat implementasi yang berjalan, sebuah model data kanonis dengan trade-off desain yang nyata dan diungkapkan secara terbuka, dan setidaknya satu bug sungguhan yang ditemukan dan diperbaiki secara terbuka. Bab penutup ini justru melihat ke depan — pada apa yang masih menjadi tindak lanjut yang disengaja dan dilacak, dan ke mana arah Profitina Laundry selanjutnya.

10.1 Apa yang Sungguh-Sungguh Selesai

Implementasi Static, PHP, JSP, dan ASP.NET yang dijelaskan Bab 5 dan 6 lengkap dan telah diuji-eksekusi-nyata, masing-masing terhadap basis data dan servernya sendiri yang sungguhan, dengan hasil yang diungkapkan dalam `VALIDATION_LOG.txt` masing-masing jalur, bukan hanya diringkas di sini. Perbaikan ValidationGroup Bab 7 telah diverifikasi dengan cara yang sama — terhadap basis data SQLite yang sungguhan, bukan sekadar dinalar.

10.2 Apa yang Masih Menjadi Tindak Lanjut yang Dilacak, Dinyatakan secara Jujur

Bab 4, Bagian 4.6 dan Bab 5, Bagian 5.6 telah mengungkapkan item terbuka yang paling signifikan: tabel `orders` yang diratakan pada jalur JSP dan skema kanonis `Bill`/`Customer`/`Employee`/`Service` pada jalur ASP.NET memodelkan bisnis yang sama dengan dua cara yang berbeda. Menyelaraskan keduanya dengan benar — memperbarui teks buku Bab 9-11 bersamaan dengan kode JSP, bukan kodenya saja secara terpisah — direncanakan sebagai langkah berikutnya yang disengaja atas materi mata kuliah WebPro, bersamaan dengan dua item yang lebih kecil dan sama jujurnya: menyelaraskan sepenuhnya teks buku Bab 14 dan 15 dengan aplikasi ASP.NET sebagaimana keberadaannya setelah perbaikan Bab 7, dan mengoreksi penyebutan janji pengaturan XAMPP pada sebuah bab sebelumnya untuk jalur PHP (Bab 5-6) agar sesuai dengan apa yang sebenarnya direkomendasikan Bab 8 buku ini.

> **Mengapa hal ini disebutkan di sini, bukan diam-diam diperbaiki lebih dulu**
>
> Sebuah buku yang hanya menjelaskan keadaan yang sudah selesai dan diselaraskan akan lebih sederhana untuk ditulis, tetapi hal itu juga akan salah menggambarkan posisi proyek ini yang sesungguhnya pada saat penerbitan. Menyebutkan item terbuka secara jujur — apa itu, mengapa ia ada, dan bahwa ia direncanakan alih-alih dilupakan — adalah komitmen pada kejujuran yang sama yang dibuka Bab 1, diterapkan pada peta jalan proyek ini sendiri, bukan hanya pada kodenya.

10.3 Ke Mana Profitina Laundry Selanjutnya

Kisah Profitina Laundry tidak berakhir dengan mata kuliah Pemrograman Web. Bisnis yang sama direncanakan untuk muncul untuk kelima kalinya, sebagai studi kasus berjalan bagi mata kuliah Framework-based Programming (FBP) — sebuah implementasi penuh dan independen yang dibangun

dengan cara yang sama seperti setiap jalur sebelumnya dibangun: eksekusi nyata, pengujian nyata, dan trade-off yang didokumentasikan secara jujur miliknya sendiri, alih-alih sekadar salinan-dan-ganti-label dari salah satu dari keempat implementasi dalam buku ini.

Bersamaan dengan itu, upaya dokumentasi yang lebih luas dari proyek ini berlanjut melampaui satu buku ini: FBP sendiri direncanakan sebagai mata kuliah lengkap, dibangun dengan kecermatan yang sama seperti WebPro, dan materi mata kuliah lebih lanjut di luar itu tetap berada dalam peta jalan proyek ini juga.

10.4 Kata Penutup

Profitina Laundry dimulai, sebagaimana yang dikatakan Bab 2 secara singkat dan dengan sengaja tidak diperdalam lebih jauh, sebagai sebuah bisnis nyata — dimiliki, distafi, dan dijalankan oleh orang-orang sungguhan yang membuat keputusan sehari-hari yang sungguhan. Apa yang berusaha dilakukan buku ini di sepanjang sepuluh bab adalah menghormati asal-usul tersebut dengan memperlakukan perangkat lunak yang dibangun di sekelilingnya sama seriusnya: dimodelkan dengan cermat, diuji secara nyata alih-alih diasumsikan, dan didokumentasikan dengan sisi-sisi kasarnya disebutkan, bukan dihaluskan. Jika seorang mahasiswa, pengajar, atau pemilik bisnis menutup buku ini dengan mempelajari satu hal darinya, semoga itu adalah ini: bahwa sebuah bisnis kecil yang nyata, dilihat dari dekat dan dengan jujur, memiliki lebih banyak hal untuk diajarkan dibandingkan kebanyakan contoh buku teks yang pernah bersusah payah dikarang.