Kuliah 1 — Dari Pemrograman Web ke Kerangka Kerja

Mata kuliah: Pemrograman Berbasis Kerangka Kerja

Buku: Irfan Subakti, *Pemrograman Berbasis Kerangka Kerja: ASP.NET Core, Laravel, dan Spring Boot*, Bab 1

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini kita mencari tahu mengapa kerangka kerja ada. Kita mengikuti sebuah laundry kecil yang situs web buatan tangannya mulai merepotkan, menamai delapan pekerjaan yang dibutuhkan setiap permintaan web, membangun daftar harga yang sama tanpa kerangka kerja dan dengan kerangka kerja, lalu menyiapkan meja kerja untuk satu semester. Buku: Bab 1, Subbab 1.1–1.11.

Cakupan Pertemuan Ini

1. **Masalahnya** — daftar harga Bu Wati, dua puluh berkas PHP, dan satu keinginan: satu perubahan di satu tempat.
2. **Gagasannya** — delapan langkah per permintaan, pustaka lawan kerangka kerja, Model–View–Controller.
3. **Tiga kerangka kerja** — ASP.NET Core 10, Laravel 13, dan Spring Boot 4, serta alasan memilih ketiganya.
4. **Meja kerja** — VS Code, .NET 10 SDK, PHP 8.5 dengan Composer, JDK 25 dengan Maven, SQLite, dan MySQL.
5. **Tanpa dan dengan** — daftar harga yang ditulis tangan dalam C#, PHP, dan Java, lalu dengan *minimal API* ASP.NET Core.

Contoh berjalan sepanjang mata kuliah adalah Profitina Laundry: pelanggan, pesanan, layanan, dan pembayaran. Fitur yang sama dibangun di setiap kerangka kerja, sehingga pada akhirnya kita bisa membandingkannya secara adil. Buku: Bab 1.

Titik Awal Kita: Daftar Harga Bu Wati

Profitina Laundry dibuka di Rungkut, Surabaya, pada 2019. Kini: tiga gerai, empat belas orang, dua belas layanan dari Rp 8.000 per kilogram sampai Rp 45.000 per pasang sepatu.

Seorang mahasiswa membuatkan situs web dengan cara yang diajarkan mata kuliah web pertama: satu berkas PHP per halaman, masing-masing membuka basis data, menjalankan kueri, dan mencetak HTML dengan `echo`.

Lalu usahanya tumbuh. Alamat baru harus diketik di lima berkas dan satu terlupa; harga berubah di satu halaman tetapi tidak di halaman lain; URL aneh dicetak kembali ke halaman tanpa di-*escape*.

Ketika Bu Wati bertanya “bisakah pelanggan melihat apakah cuciannya sudah siap?”, tidak ada yang berani menyentuh kodenya.

Keinginannya sederhana dan sangat umum: satu perubahan dibuat di satu tempat; setiap halaman tampil seragam tanpa menyalin; setiap nilai otomatis diamankan; setiap fitur baru punya rumah yang jelas. Keinginan itulah yang dipenuhi kerangka kerja. Buku: Subbab 1.1.

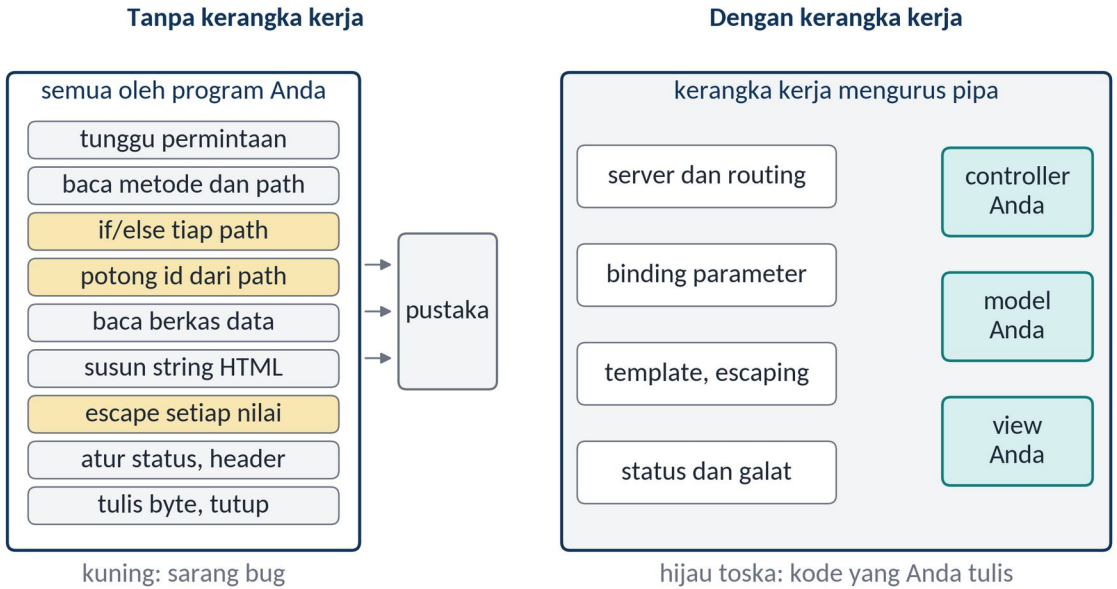
Delapan Langkah untuk Setiap Permintaan

1. **Menunggu** koneksi dan membaca permintaan. **2. Merutekan**: menentukan kode mana yang bertanggung jawab atas jalur dan metode ini.
3. **Mengekstrak** bagian jalur yang berubah-ubah (kode `S0004`) dan mengonversinya. **4. Mengambil** data dari penyimpanan.
5. **Memutuskan** menurut aturan bisnis. **6. Merender** HTML atau JSON, dengan meng-*escape* setiap nilai dari luar.
7. **Menjawab** dengan kode status, *header*, dan *byte*. **8. Bertahan**: mencatat log, dan mengirim halaman galat yang sopan alih-alih mogok.

> **WAWASAN KUNCI:** Hanya langkah 4 dan 5 yang menyangkut *laundry*. Enam lainnya sama untuk klinik, sekolah, atau bank, dan di sanalah sebagian besar *bug* tinggal: `if` yang salah dalam perutean, id yang terpotong satu karakter terlalu awal, nilai yang dicetak tanpa di-*escape*.

Tanyakan ke kelas: langkah mana yang gagal pada tiap insiden Bu Wati? Alamat yang terlupa adalah tata letak bersama yang tidak ada (merender), harga yang basi adalah data di dua tempat (mengambil), URL aneh adalah escaping yang tidak ada (merender). Buku: Subbab 1.2.1.

Pustaka atau Kerangka Kerja: Siapa Memanggil Siapa?

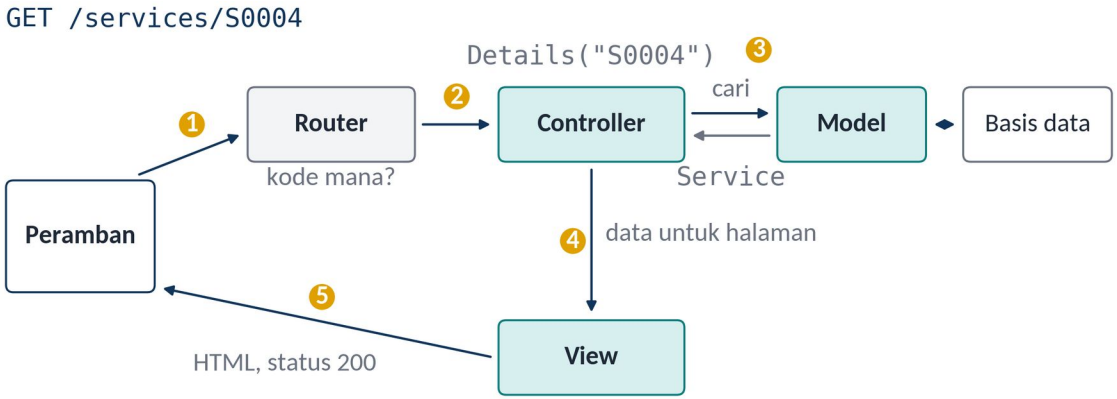


Gambar 1.1 Pustaka dipanggil oleh program Anda; kerangka kerja memanggil program Anda. Dengan kerangka kerja, satu-satunya kode yang Anda tulis adalah bagian yang menyangkut bisnis Anda.

*Gambar 1.1: **pustaka** adalah sekumpulan rutin siap pakai yang dipanggil program Anda kapan pun ia mau. **Kerangka kerja** adalah program yang hampir lengkap dengan lubang-lubang, dan Anda yang mengisi lubangnya. Johnson dan Foote menjelaskannya pada 1988: alur kendali*

milik kerangka kerja. Ini sering disebut prinsip Hollywood, "jangan hubungi kami, kami yang akan menghubungi Anda", atau ***pembalikan kendali*** *(inversion of control). Coba pada program hari ini:* `HttpListener` *dan* `fgetcsv` *adalah pustaka yang dipanggil kode kita; setelah* `app.Run()`*, ASP.NET Core memegang perulangan dan memanggil handler kita. Buku: Subbab 1.2.2.*

Model, View, Controller



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.

*Gambar 1.2: **model** menyimpan data dan aturan bisnis, **view** mengubah data menjadi HTML, **controller** menangani satu permintaan dan menghubungkan keduanya. **Router** memilih aksi controller dari jalur dan metode. Pegangan untuk seluruh mata kuliah: jaga controller tetap ramping. Buku: Subbab 1.2.3.*

Apa yang Diberikan Kerangka Kerja, dan Harganya

Memberi: server web, perutean, pengikatan parameter, mesin templat dengan *escaping* otomatis, ORM untuk basis data, validasi, login, log, dan perkakas pengujian.

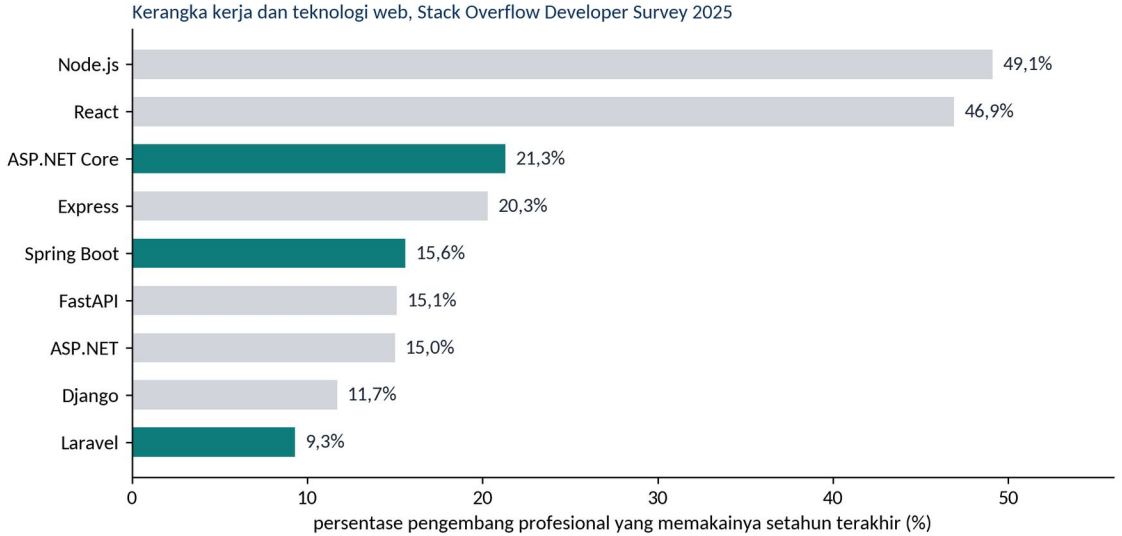
Memberi: tempat yang pasti untuk setiap jenis kode, sehingga anggota tim baru tahu harus mencari di mana.

Harganya: kurva belajar, konvensi yang harus diikuti, dan “sihir” yang menyembunyikan apa yang terjadi sampai Anda memahaminya.

Harganya: ketergantungan pada siklus rilis kerangka kerja; versi mayor setiap satu atau dua tahun berarti pemutakhiran yang direncanakan.

Pertahanan terbaik terhadap “sihir” adalah mengenal delapan langkah tadi: bila ada yang gagal, tanyakan langkah mana yang gagal dan baca apa yang dicatat kerangka kerja untuk langkah itu. Buku: Subbab 1.2.4.

Mengapa ASP.NET Core, Laravel, dan Spring Boot



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.

Gambar 1.3: ASP.NET Core dipakai oleh 21,3% pengembang profesional, Spring Boot oleh 15,6%, dan Laravel oleh 9,3%. Bersama-sama mereka mewakili tiga keluarga besar pengembangan sisi server: dunia Microsoft, hosting PHP, dan perusahaan besar berbasis Java. Buku: Subbab 1.3.

Ketiganya Berdampingan

Tabel 1.1 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 (Sept. 2026)	10.0 LTS	13.x	4.1.1
Perkakas proyek	CLI `dotnet`	Composer, `php artisan`	Maven, Spring Initializr
Templat	Razor (`.cshtml`)	Blade (`.blade.php`)	Thymeleaf (`.html`)
ORM	Entity Framework Core	Eloquent	Spring Data JPA
Bab	3–6	7–10	11–14

Tabel 1.1: setiap baris akan menjadi akrab selama semester. ASP.NET Core buatan Microsoft, Laravel buatan Taylor Otwell, Spring Boot buatan VMware yang kini milik Broadcom; ketiganya sumber terbuka. Paket diambil dari NuGet, Packagist, dan Maven Central. Buku: Subbab 1.3 dan 1.9.2.

Menyiapkan Meja Kerja

VS Code dengan C# Dev Kit, PHP Intelephense dan Laravel, Extension Pack for Java, serta Spring Boot Extension Pack; di komputer yang lambat pasang hanya ekstensi untuk kerangka kerja pekan itu.

.NET 10 SDK (bukan hanya *runtime*): membawa `dotnet new`, `dotnet build`, `dotnet run`.

PHP 8.5 dan Composer melalui `php.new` di macOS, Linux, dan Windows, lalu *installer* `laravel`.

JDK 25 dan Maven (atau Maven *wrapper* di tiap proyek); **SQLite** untuk pengembangan, **MySQL 9.7 LTS** untuk server bersama.

Buka terminal baru setiap selesai memasang, agar perintah baru dikenali. Di Windows, jalankan PowerShell sebagai administrator untuk skrip `php.new`; di macOS dan Linux cukup terminal biasa. Pasang satu perkakas, periksa versinya, baru lanjut ke perkakas berikutnya: kesalahan yang ditemukan segera jauh lebih mudah diperbaiki daripada tiga kesalahan yang menumpuk. Buku: Subbab 1.4.1–1.4.5.

Memeriksa Semuanya

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

Setiap perintah harus menjawab dengan versi. `dotnet --list-sdks` yang tidak mencetak apa pun berarti hanya *runtime* yang terpasang.

`java -version` yang menjawab 17 atau 21 berarti JDK lama berada lebih dulu di `PATH`: atur `JAVA_HOME` dan letakkan folder `bin`-nya di *depan*.

Mesin buku ini menjawab, misalnya, `10.0.112` untuk SDK dengan `Microsoft.AspNetCore.App 10.0.12`. Buku: Subbab 1.4.6.

Daftar Harga Tanpa Kerangka Kerja

Situs kecil yang sama tiga kali, membaca `data/Service.csv`: C# di atas `HttpListener`, PHP di atas server bawaan, Java di atas `com.sun.net.httpserver`.

Setiap program mengerjakan kedelapan langkah sendiri: 110 baris C#, 66 baris PHP, dan 108 baris Java.

Skrip PHP sekaligus menjadi router, controller, model, dan view; komentar (1) sampai (6) di berkas C# menandai pekerjaan yang akan diambil alih kerangka kerja.

Setiap versi membuat keputusan kecilnya sendiri tentang garis miring di akhir jalur, metode yang tidak dikenal, dan *escaping*, dan keputusannya tidak seragam.

Tampilkan ketiga berkas berdampingan di editor, lalu hitung bersama kelas berapa baris yang benar-benar tentang laundry. Hanya segelintir. Perhatikan juga bahwa ketiga program itu tidak salah; semuanya lulus uji yang sama. Masalahnya bukan kebenaran hari ini, melainkan biaya setiap perubahan besok: rute baru berarti cabang `if` baru, halaman baru berarti HTML yang disalin, nilai baru berarti satu lagi tempat untuk lupa meng-escape. Buku: Subbab 1.5.1–1.5.3.

Daftar Harga yang Sama dengan ASP.NET Core

Kode 1.1 Inti `aspnet/Program.cs`, daftar harga dengan *minimal API* ASP.NET Core (lengkapnya 26 baris).

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

// ... services are read from ../data/Service.csv ...

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);
```

Kode 1.1: `MapGet` adalah rutenya, `{id}` diekstrak dan diikat ke parameter `string id`, daftar menjadi JSON dengan sendirinya, dan `Results.NotFound` mengatur kode status. Berkas proyeknya memakai SDK `Microsoft.NET.Sdk.Web` dan menargetkan `net10.0`. Buku: Subbab 1.6.1.

Menjalankannya

```
cd Chapter01/aspnet
dotnet run --urls http://127.0.0.1:4013
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}
```

Kestrel *adalah server web bawaan ASP.NET Core; JSON dan header* `Content-Type` *berasal dari kerangka kerja. Program ini dibangun dan dijalankan dengan .NET 10 untuk buku ini, dan keluaran di atas adalah keluaran sungguhannya. Buku: Subbab 1.6.2.*

Empat Program, Satu Skrip Uji

```
===== 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
```

Lima permintaan yang sama dikirim ke keempat program. Program tulisan tangan menjawab `POST` dengan 411 (C#) atau 405 (PHP, Java); kerangka kerja menjawab 405, secara konsisten.

Di dalam JSON, `<b>` hanyalah data yang tidak berbahaya; di dalam HTML, program tulisan tangan harus meng-*escape*-nya sendiri, setiap kali.

Blok di atas adalah bagian ASP.NET Core dari hasil perbandingan yang sungguh dijalankan; buku mencetak keempatnya. Buku: Subbab 1.7.

Proyek Nyata Pertama di Setiap Kerangka Kerja

```
dotnet new mvc -o FirstMvc
laravel new first-laravel
curl -o first-spring.zip "https://start.spring.io/starter.zip?..."
```

`dotnet new mvc` membuat proyek MVC lengkap dengan controller, view Razor, dan tata letak; `dotnet run` menjalankannya.

`laravel new` membuat aplikasi Laravel; `php artisan serve` menjalankannya. Spring Initializr memberi proyek Maven dengan dependensi `web` dan `thymeleaf`.

Proyek Laravel dan Spring membutuhkan Packagist dan Maven Central; jalankan di mesin Anda sendiri dengan mengikuti README bab ini. Jangan khawatir bila proyek-proyek ini tampak penuh berkas; Bab 3, 7, dan 11 membahas setiap folder satu per satu, dan setiap folder punya tugas yang jelas. Buku: Subbab 1.6.3.

Satu Rute Lagi: Kerangka Kerja Lawan Tulisan Tangan

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

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

`{max:int}` adalah **batasan rute** (*route constraint*): rute hanya cocok bila segmennya bilangan bulat, dan nilainya tiba sudah berupa `int`. Untuk `/services/under/cheap` tidak ada rute yang cocok dan kerangka kerja menjawab 404.

Versi C# tulisan tangan membutuhkan cabang `else if` sendiri dengan `StartsWith`, `int.TryParse`, perulangan `StringBuilder`, dan panggilan `Escape` untuk setiap deskripsi, sekitar selusin baris.

Dalam rantai `if`, uji yang lebih spesifik harus mendahului uji yang lebih umum; router kerangka kerja mengurus urutan itu untuk kita.

Keluaran di atas berasal dari eksekusi sungguhan perubahan ini untuk `/services/under/7000`: hanya setrika saja, Rp 6.000, yang di bawah batas. Tanyakan ke kelas langkah mana yang diambil alih batasan `:int`: merutekan dan mengekstrak. Buku: Soal Latihan 1.3 dan 1.4 beserta penyelesaiannya di Subbab 1.13.

Menjadikannya Lebih Baik: Yang Masih Kita Kerjakan Sendiri

Berkas data — CSV yang diurai dengan ekspresi reguler. Bab 5 memindahkannya ke SQLite melalui Entity Framework Core.

Halaman HTML — Bab 3 membangunnya dengan controller, view Razor, tata letak bersama, dan *escaping* otomatis.

Formulir dan validasi — Bab 3 dan 6. **Login dan peran** — Bab 6, 10, dan 14. **Pengujian otomatis** — Bab 10 dan 14.

Bab 2 lebih dulu membangun kerangka kerja mini buatan sendiri, sekitar tiga ratus baris, dengan router, kontainer DI, ORM, dan mesin templat. Tujuannya bukan untuk dipakai sungguhan, melainkan menghapus misterinya. Buku: Subbab 1.8.

Memakainya Sungguhan

Mulailah dari berkas data: penjahit mendaftar “potong kelim celana”, klinik kecil mendaftar tarif konsultasinya, kebun buah dan durian mendaftar buahnya per kilogram menurut musim. Pertahankan tiga kolomnya, dan program tetap berjalan tanpa diubah.

Dalam praktik, pilihan kerangka kerja ditentukan oleh orang dan tempatnya: tim C# memilih ASP.NET Core, agensi dengan *hosting* PHP bersama memilih Laravel, perusahaan besar berbasis Java memilih Spring Boot.

Ketiganya mampu membangun semua isi mata kuliah ini; Bab 15 membandingkannya dengan pengukuran setelah kita membangun fitur yang sama di masing-masing. Untuk proyek sungguhan, pertanyaan pertama bukan “kerangka kerja mana yang terbaik?” melainkan “siapa yang akan merawatnya, dan di mana ia akan berjalan?”. Buku: Subbab 1.9.

Kesalahan yang Sering Terjadi

Memasang *runtime* alih-alih SDK; **terminal lama** yang dibuka sebelum pemasangan; **dua Java** di `PATH`.

Folder yang salah: program membaca `../data/Service.csv` relatif terhadap tempat ia dijalankan. **Port yang masih terpakai:** eksekusi sebelumnya masih hidup; hentikan dengan **Ctrl+C**.

Menganggap kerangka kerja itu sihir: tanyakan langkah mana dari delapan langkah yang gagal, alih-alih mengubah sesuatu secara acak.

> **PERINGATAN:** HTML yang dirakit dengan penggabungan *string* harus meng-*escape* setiap nilai dari luar, setiap kali. Serahkan pada mesin templat, dan anggap `Html.Raw`, `{!! !!}`, atau `th:utext` sebagai tempat yang perlu diperiksa.

Masing-masing kesalahan ini menghabiskan satu sore bagi pemula; menamainya hanya butuh satu menit. Buku: Subbab 1.10.

Ringkasan: Lima Hal untuk Diingat

1. Setiap permintaan melewati delapan langkah: menunggu, merutekan, mengekstrak, mengambil, memutuskan, merender, menjawab, bertahan. Hanya mengambil dan memutuskan yang menyangkut bisnis.
2. Pustaka dipanggil oleh kode Anda; kerangka kerja memanggil kode Anda (Gambar 1.1).
3. Kerangka kerja menata kode sebagai Model–View–Controller dengan router di depannya (Gambar 1.2); jaga controller tetap ramping.
4. Mata kuliah ini memakai ASP.NET Core 10, Laravel 13, dan Spring Boot 4 di atas VS Code, .NET 10, PHP 8.5, JDK 25, SQLite, dan MySQL.
5. Ditulis tangan, daftar harga butuh 110, 66, dan 108 baris; dengan *minimal API* hanya 26, dan kerangka kerja membuat keputusan-keputusan kecil secara konsisten.

Kalau hanya satu kalimat yang Anda bawa pulang, jadikan kalimat Bu Wati: satu perubahan di satu tempat. Setiap bab berikutnya adalah cara mewujudkannya untuk satu jenis perubahan lagi: harga, tata letak halaman, aturan, login, pengujian. Bila sebuah fitur kerangka kerja terasa aneh, tanyakan keinginan Bu Wati yang mana yang dilayaninya. Buku: Subbab 1.11.

Pertemuan Berikutnya

Bab 2 — Fondasi Bersama Kerangka Kerja Modern. MVC, perutean, ORM, injeksi dependensi, dan templat, dengan membangun kerangka kerja mini buatan kita sendiri.

Sebelum itu: pasang meja kerja, jalankan keempat program daftar harga, dan kerjakan soal latihan bab ini; penyelesaiannya ada di buku.

Pemanasan yang baik: tambahkan rute `/services/under/{max:int}` ke minimal API sendiri, lalu coba nilai yang bukan angka. Buku: Subbab 1.12 dan 1.13.

Kuliah 2 — Fondasi Bersama Kerangka Kerja Modern

Mata kuliah: Pemrograman Berbasis Kerangka Kerja

Buku: Irfan Subakti, *Pemrograman Berbasis Kerangka Kerja: ASP.NET Core, Laravel, dan Spring Boot*, Bab 2

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini kita membuka kotaknya. Kita menamai empat gagasan yang menjadi dasar setiap kerangka kerja web modern (perutean, injeksi dependensi, pemetaan objek-relasional, dan templat), membangun masing-masing dengan tangan dalam kerangka kerja mungil bernama MiniFrame, lalu menunjuk empat gagasan yang sama di ASP.NET Core, Laravel, dan Spring Boot. Buku: Bab 2, Subbab 2.1–2.9.

Cakupan Pertemuan Ini

1. **Pertanyaannya** — bagaimana kerangka kerja memanggil fungsi yang belum pernah dilihatnya?
2. **Empat fondasi** — perutean, DI, ORM, dan templat di sepanjang satu permintaan.
3. **MiniFrame** — keempatnya dalam 226 baris C#, dengan demo yang memperlihatkan setiap fondasi bekerja.
4. **Tiga yang sungguhan** — daftar harga yang sama di Laravel dan Spring Boot, dan tabel yang menerjemahkan di antara semuanya.
5. **Yang ditinggalkan MiniFrame** — dan mengapa kerangka kerja sungguhan berukuran besar.

MiniFrame tidak dimaksudkan untuk dipakai sungguhan. Tujuannya menghapus misteri, sehingga ketika ASP.NET Core, Laravel, atau Spring Boot mengerjakan hal yang sama dalam skala besar, Anda tahu apa yang terjadi di dalamnya. Buku: Bab 2.

Posisi Kita Sekarang

Bab 1: daftar harga tulisan tangan butuh 66 sampai 110 baris; dengan *minimal API* ASP.NET Core, 26 baris.

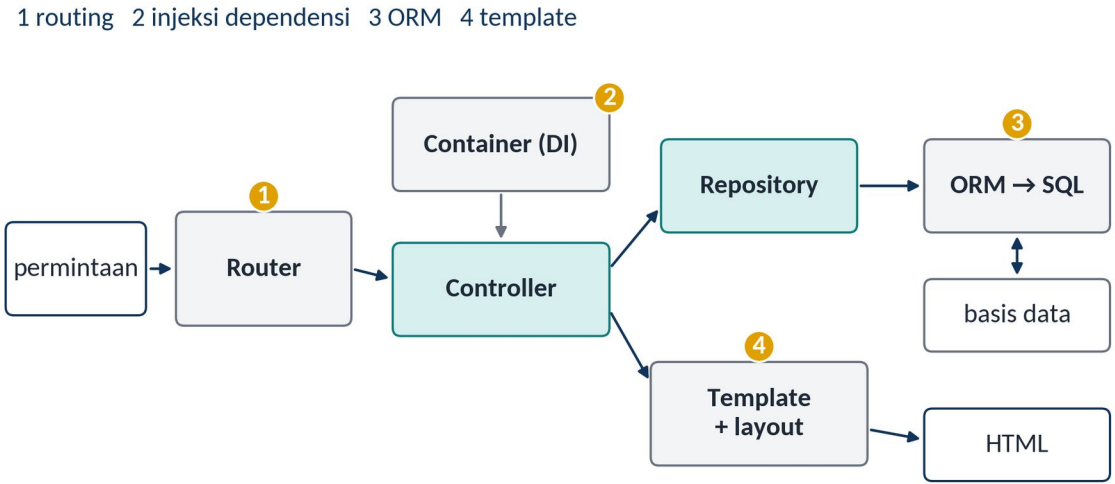
Tetapi *bagaimana*? Ketika `MapGet("/services/{id}", (string id) => …)` menerima `/services/S0004`, sesuatu memecah jalurnya, membandingkannya dengan templat, dan meneruskan `"S0004"` ke fungsi yang belum pernah dilihatnya.

Keponakan Bu Wati, **Dimas**, membuka ketiga kerangka kerja dan menemukan `[HttpGet("{id}")]`, `Route::get('/services/{id}', …)`, dan `@GetMapping("/{id}")`. Tiga sintaks, satu gagasan.

Di ketiganya, controller menerima repositori lewat konstruktornya, dan tidak ada kode di mana pun yang membuatnya. Lagi-lagi tiga sintaks, satu gagasan.

Tanpa jawaban, kerangka kerja terasa seperti sihir, dan sihir mustahil di-debug. Bab ini menamai empat gagasan itu sekali, membangun masing-masing dengan tangan, lalu menemukannya di ketiga kerangka kerja. Buku: Subbab 2.1.

Empat Fondasi di Sepanjang Satu Permintaan



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.

Gambar 2.1: router (1) memilih aksi; container (2) membangun controller beserta semua yang dibutuhkannya; controller meminta data kepada repositori, dan ORM (3) mengubahnya menjadi SQL dan baris-barisnya menjadi objek; mesin templat (4) memasukkan data ke halaman, halaman ke layout, lalu mengirim HTML kembali. Buku: Subbab 2.2.

Empat Definisi

> **DEFINISI. Rute** memasangkan metode HTTP dan *templat* URL seperti `/services/{id}` dengan sebuah aksi. **Router** menyimpan rute dalam tabel rute, menemukan rute yang cocok dengan jalur, dan mengekstrak *parameter*-nya. **Batasan** (*constraint*) seperti `{max:int}` membatasi apa yang boleh dicocokkan sebuah parameter.

Injeksi dependensi: sebuah kelas tidak membuat apa yang dibutuhkannya; ia menyatakannya di konstruktor dan menerimanya. **Container DI** membangun seluruh graf dari *pendaftaran* dan menentukan **masa hidup** (*lifetime*) setiap objek.

Pemetaan objek-relasional (ORM): membaca baris menjadi objek dengan mencocokkan nama kolom dan properti, dan mengubah syarat yang ditulis dalam bahasa pemrograman menjadi SQL. **Migrasi** adalah satu perubahan skema yang berversi, ditulis sebagai kode.

Mesin templat: mengisi *lubang* di halaman dengan nilai, **di-*escape* secara bawaan**, mengulang blok untuk daftar, dan menaruh setiap halaman di dalam **layout** bersama.

Tulis keempat kata ini di papan dan biarkan di sana sepanjang semester. Setiap fitur kerangka kerja baru yang kita temui akan jatuh di bawah salah satunya. Buku: Subbab 2.2.

MiniFrame: Kerangka Kerja Mungil dalam 226 Baris

`Router.cs` — tabel rute, pencocokan templat, dan ekstraksi parameter (55 baris).

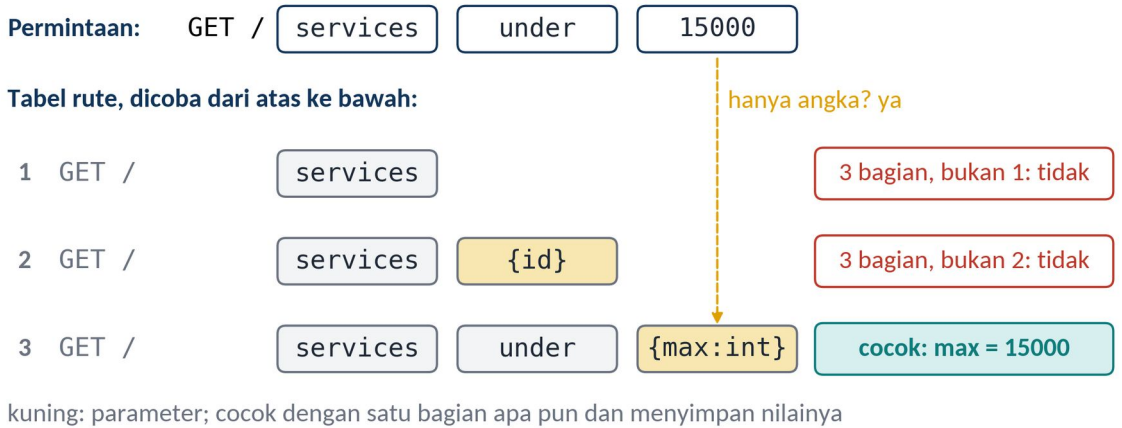
`Container.cs` — pendaftaran, dua masa hidup, dan injeksi lewat konstruktor dengan refleksi (45 baris).

`Mapper.cs` — baris menjadi objek berdasarkan nama, dan syarat C# menjadi teks klausa SQL `WHERE` (90 baris).

`Template.cs` — lubang, perulangan, dan *layout*, dengan *escaping* secara bawaan (36 baris). Di atasnya: `Laundry.cs` (aplikasinya) dan `Program.cs` (pendaftaran, rute, dan server HTTP kecil).

MiniFrame adalah C# murni di .NET 10 tanpa paket apa pun, sehingga dibangun dan dijalankan di sini; setiap keluaran MiniFrame yang ditampilkan hari ini adalah keluaran sungguhannya. Buku: Subbab 2.3.

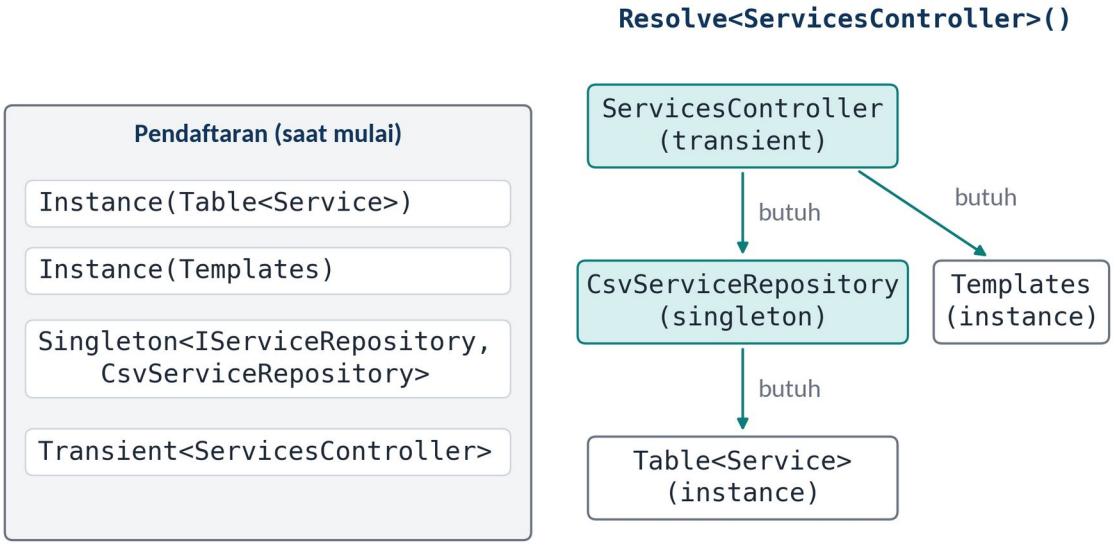
Perutean



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.

Gambar 2.2: router memecah jalur di setiap garis miring, membandingkan jumlah segmen, lalu membandingkan setiap segmen literal dan memeriksa setiap parameter terhadap batasannya. Nilai yang diekstrak masuk ke kamus kecil yang diterima aksi. MiniFrame mencoba rute secara berurutan, sehingga rute umum yang didaftarkan sebelum rute spesifik akan menelan permintaannya. Router sungguhan tidak mencoba rute satu per satu; ia menyusun pohon dan memberi peringkat, sehingga rute literal selalu menang atas rute berparameter. Buku: Subbab 2.3.1.

Injeksi Dependensi



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.

Gambar 2.3: saat awal jalan kita mendaftarkan "untuk `IServiceRepository` pakai `CsvServiceRepository`, sebagai singleton" dan "bangun `ServicesController` sebagai transient". Ketika controller dibutuhkan, container membaca konstruktornya dengan refleksi, membangun setiap parameternya lebih dulu, lalu memanggilnya. Buku: Subbab 2.3.2.

Masa Hidup: Bolehkah Dua Permintaan Berbagi Objek Ini?

Singleton — satu objek untuk seluruh aplikasi: katalog harga, konfigurasi.

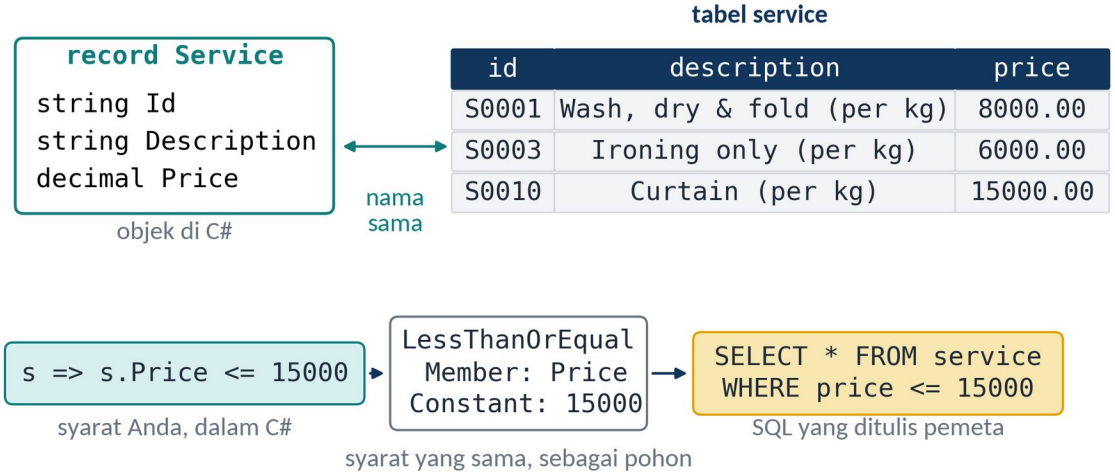
Scoped — satu objek per permintaan: konteks basis data, pengguna yang sedang masuk. MiniFrame tidak memilikinya; ketiga container sungguhan memilikinya.

Transient — objek baru setiap kali diminta: controller yang ringan.

> **PERINGATAN:** *Singleton* yang menyimpan data per permintaan (pengguna saat ini, keranjang belanja) mencampur data satu pengunjung ke halaman pengunjung lain. Tanyakan untuk setiap pendaftaran: *bolehkah dua permintaan berbagi objek ini?*

Bergantung pada antarmuka, bukan pada kelas konkret, itulah yang membuat implementasi bisa diganti: CSV hari ini, SQLite besok, test double di Bab 10, tanpa menyentuh controller. Buku: Subbab 2.3.2.

Pemetaan Objek-Relasional



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.

Gambar 2.4: di C#, lambda yang dioper sebagai `Expression<Func<Service, bool>>` tidak dikompilasi menjadi kode, melainkan disimpan sebagai pohon simpul yang bisa ditelusuri mapper: parameter, properti `Price`, operator `<=`, konstanta 15000. Mencetak pohon itu menghasilkan `WHERE price <= 15000`. EF Core melakukan hal yang persis sama dalam skala jauh lebih besar. Kuncinya ada pada tipe parameternya: bila lambda yang sama dioper sebagai `Func<Service, bool>` biasa, ia menjadi kode yang hanya bisa dijalankan, bukan dibaca, dan tidak ada SQL yang bisa dihasilkan darinya. Eloquent dan Spring Data sampai ke tujuan yang sama dengan jalan lain, seperti yang kita lihat sebentar lagi. Buku: Subbab 2.3.3.

Templat: Lubang, Perulangan, dan Layout

Kode 2.1 `templates/services.html`: perulangan atas baris-baris model, dengan setiap nilai di-*escape*.

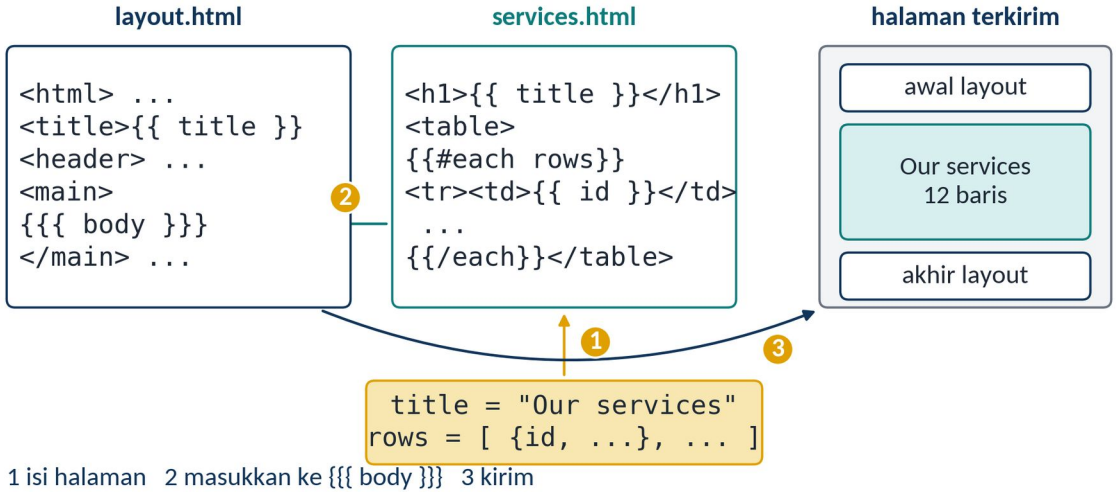
```
<h1>{{ title }}</h1>
<table>
{{#each rows}}  <tr><td><a href="/services/{{ id }}">{{ id }}</a></td>
      <td>{{ description }}</td><td>{{ price }}</td></tr>
{{/each}}</table>
```

`{{ x }}` adalah **lubang ter-*escape***: `<`, `>`, `&`, dan tanda kutip menjadi teks yang tidak berbahaya. `{{{ x }}}` adalah **lubang mentah**, hanya dipakai untuk `{{{ body }}}` di *layout*.

`{{#each rows}}` mengulang blok untuk setiap baris; `layout.html` bersama menyimpan *head* dan *header* satu kali untuk semua halaman.

Kode 2.1 adalah seluruh halaman daftar. Escaping adalah bawaan, dan keluaran mentah adalah pengecualian yang harus bisa dipertanggungjawabkan: satu aturan itu menutup satu kelas serangan sekaligus. Buku: Subbab 2.3.4.

Dari Model ke Halaman



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

Gambar 2.5: dua tahap. Pertama templat halaman diisi dengan model; lalu HTML-nya masuk ke layout di tempat `{{{ body }}}`. `@RenderBody()` *di Razor,* `@yield` *di Blade, dan fragment di Thymeleaf adalah gagasan yang sama. Buku: Subbab 2.3.4.*

Aplikasi di Atasnya

`Laundry.cs` berisi model `Service`, sebuah repositori, dan `ServicesController`: tanpa perutean, tanpa penguraian, tanpa teks SQL, tanpa *escaping* HTML.

`Program.cs` mendaftarkan kelas-kelas, memetakan tiga rute, lalu menjalankan demo luring (`--demo`) atau menyalakan server HTTP kecil.

Pemisahan itulah intinya: aplikasi hanyalah bagian yang menyangkut *laundry*, persis gambar Bab 1: kerangka kerja memanggil kode kita, dan kali ini kerangka kerjanya kita bangun sendiri.

Minta kelas mencari satu baris di `Laundry.cs` yang akan berubah bila toko pindah dari berkas CSV ke SQLite. Tidak ada: yang berubah adalah repositori di balik antarmuka, bukan controller. Buku: Subbab 2.3.5.

Menjalankan Demo

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

Setiap fondasi melaporkan apa yang dikerjakannya. Repositori dibangun sekali (singleton), controller pada setiap permintaan (transient). Eksekusi lengkap di buku juga memperlihatkan `/services/under/cheap` ditolak batasan `int` dengan 404, dan `/services/%3Cb%3E` dijawab dengan `<b>`, sudah di-escape. Buku: Subbab 2.4.

Rute yang Sama di Laravel

Kode 2.2 `routes/web.php` di Laravel: tiga rute, masing-masing menunjuk ke sebuah metode controller; `whereNumber` adalah batasannya.

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

Kode 2.2: satu berkas terpusat, satu baris per rute, sebuah kelas controller dan nama metodenya. Laravel, seperti MiniFrame, peka urutan di sini: `under` mendahului `{id}`. Di ASP.NET Core dan Spring, rute yang sama berupa atribut atau anotasi pada metode controller, dengan batasan seperti di Tabel 2.1. Buku: Subbab 2.5.1.

ORM di Spring Data: Kueri dari Sebuah Nama

Kode 2.3 Inti `ServiceRepository.java`: repositori Spring Data. Kuerinya diturunkan dari nama metode.

```java
// 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);
}
```

Tiga cara menulis satu kueri: pohon ekspresi (EF Core, MiniFrame), pembangun *fluent* `Service::where('price', '<=', $max)` (Eloquent), nama metode (Spring Data).

Model Eloquent hanya menyatakan apa yang berbeda dari konvensi Laravel: tabel `service`, kunci berupa *string*, tanpa *timestamp*.

Kode 2.3: Spring Data memecah nama menjadi `find`, `By`, `Price`, `LessThanEqual` lalu membangun `WHERE price <= ?`. Entitas `Service` adalah kelas dengan `@Entity`, `@Table(name = "service")`, dan `@Id`. Kode Laravel dan Spring membutuhkan Packagist dan Maven Central, jadi ditulis berdasarkan dokumentasi resmi dan diperiksa sintaksnya di sini, tidak dijalankan. Buku: Subbab 2.5.3.

Templat di Blade dan Thymeleaf

Kode 2.4 Bagian dari `resources/views/services/index.blade.php`: halaman daftar, mengisi slot konten milik *layout*.

```
@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
```

Kode 2.4: `{{ }}` adalah lubang ter-escape, `@foreach` perulangannya, `@extends('layouts.app')` bersama `@yield` adalah layout-nya. Di Thymeleaf, halaman yang sama berupa HTML biasa dengan atribut: `th:each="s : ${services}"` mengulang dan `th:text="${s.description}"` adalah lubang ter-escape, sehingga desainer bisa membuka berkasnya langsung di peramban. Buku: Subbab 2.5.4.

Satu Fitur, Tiga Kerangka Kerja

Tabel 2.1 Keempat fondasi di MiniFrame dan di tiga kerangka kerja buku ini.

	MiniFrame	ASP.NET Core	Laravel	Spring Boot
Batasan	`{max:int}`	`{max:int}`	`whereNumber('max')`	`{max:\\d+}`
Injeksi	konstruktor	konstruktor	konstruktor	konstruktor
Baris ke objek	`Table<Service>`	entitas EF Core	model Eloquent	JPA `@Entity`
Lubang ter-*escape*	`{{ x }}`	`@x`	`{{ $x }}`	`th:text`
Lubang mentah	`{{{ x }}}`	`@Html.Raw(x)`	`{!! $x !!}`	`th:utext`
Layout	`{{{ body }}}`	`@RenderBody()`	`@yield`	`th:replace`

Tabel 2.1 adalah tabel terjemahan mata kuliah ini. Pelajari gagasannya sekali, maka setiap sintaks tinggal detail. Spring mendaftarkan kelas dengan memindai anotasi seperti `@Repository` dan `@Service`, bukan dengan mendaftarkannya satu per satu. Buku: Subbab 2.5.5.

Menjadikannya Lebih Baik: Yang Ditinggalkan MiniFrame

Parameter dalam SQL. MiniFrame mencetak nilai ke dalam teks SQL (`WHERE id = '<b>'`); ORM sungguhan mengirim `WHERE id = @p0` dan nilainya terpisah, sehingga tidak ada nilai yang bisa mengubah makna perintah.

Prioritas rute. Router sungguhan mengompilasi tabel menjadi pohon dan memberi peringkat pada templat, sehingga urutan pendaftaran tidak lagi penting.

Masa hidup *scoped* dan pembuangan objek, **konkurensi** untuk ribuan permintaan serentak, **templat terkompilasi**, dan **migrasi** (Bab 5, 9, dan 11).

Setiap kekurangan adalah pelajaran tentang mengapa kerangka kerja sungguhan besar: gagasannya muat dalam 226 baris, keamanan dan kecepatannya tidak. Kekurangan pertama paling penting untuk diingat: MiniFrame hanya menggandakan tanda kutip, cara yang rapuh. Parameter memisahkan perintah dari data sepenuhnya, dan itulah yang dilakukan EF Core, Eloquent, dan Spring Data setiap kali. Buku: Subbab 2.6.

Memakainya Sungguhan: Empat Pertanyaan

Rencanakan setiap fitur baru dengan empat pertanyaan: **Rute yang mana? Apa yang perlu diinjeksikan ke controller? Kueri yang mana? Templat yang mana?**

Untuk “apakah cucian saya sudah siap?” dari Bu Wati: `GET /orders/{code}`; repositori pesanan; satu baris berdasarkan kodenya; halaman status kecil.

Empat jawaban yang sama cocok untuk kerangka kerja apa pun, dan kerangka kerja yang asing bisa dibaca melalui keempatnya.

Coba empat pertanyaan ini pada usaha yang dikenal kelas: nomor antrean klinik, daftar panen kebun buah dan durian. Jawabannya pendek, dan begitu ditulis, kodenya hampir menulis dirinya sendiri di kerangka kerja apa pun yang dipakai tim. Pertanyaan yang sama juga membantu membaca proyek orang lain: temukan rutenya, ikuti ke controller, lihat konstruktornya, lalu kueri dan view-nya. Buku: Subbab 2.7.

Kesalahan yang Sering Terjadi

Rute dalam urutan yang salah di router yang peka urutan: rute umum menelan rute spesifik. `new` **di dalam kelas yang dibangun container**: diam-diam menggagalkan container.

Masa hidup yang salah. SQL dari penggabungan *string*, bahkan lewat pintu darurat SQL mentah milik ORM (`FromSqlRaw`, `DB::raw`, kueri *native*).

Mematikan *escaping* untuk "memperbaiki" `&` **di halaman**: penyebabnya *escaping* ganda. **Menghafal sintaks, bukan gagasan.**

Kesalahan masa hidup paling berbahaya, karena lolos semua uji dengan satu pengguna dan baru gagal ketika dua pelanggan memakai situs pada saat yang sama. Pemula sering mempelajari `@GetMapping`*,* `Route::get`*, dan* `[HttpGet]` *sebagai tiga fakta yang tidak berhubungan; pelajari gagasannya sekali. Buku: Subbab 2.8.*

Ringkasan: Lima Hal untuk Diingat

1. Ketiga kerangka kerja berbagi MVC dan empat fondasi: perutean, DI, ORM, templat (Gambar 2.1).

2. Router mencocokkan metode dan jalur dengan templat, menerapkan batasan, dan meneruskan nilai ke satu aksi.

3. Container membangun objek dari pendaftaran; *singleton*, *scoped*, dan *transient* menyatakan siapa yang berbagi objek itu.

4. ORM memetakan baris ke objek berdasarkan nama dan kueri dalam bahasa pemrograman ke SQL, selalu dengan parameter.

5. Mesin templat mengisi lubang ter-*escape*, mengulang blok, dan memakai *layout*; keluaran mentah harus bisa dipertanggungjawabkan.

MiniFrame mengerjakan keempatnya dalam 226 baris; kekurangannya menjelaskan ukuran kerangka kerja sungguhan. Buku: Subbab 2.9.

Pertemuan Berikutnya

Bab 3 — ASP.NET Core MVC I. Proyek, *pipeline middleware*, perutean, controller, dan view Razor, dengan *layout* bersama dan *tag helper*.

Sebelum itu: jalankan demo MiniFrame, lalu kerjakan soal latihan bab ini, termasuk soal yang mendaftarkan rute umum sebelum rute spesifik.

Soal tentang urutan rute patut dikerjakan di depan kelas: daftarkan `/services/{id}` *sebelum* `/services/under/{max}`*, jalankan demo, dan lihat permintaan* `under` *jatuh ke rute yang salah dengan* `id=under`*. Setelah itu, tukar urutannya atau tambahkan batasan, dan jalankan lagi. Penyelesaian semua soal ada di buku. Buku: Subbab 2.10 dan 2.11.*

Kuliah 3 — ASP.NET Core MVC I: Proyek, Pipeline, Perutean, Controller, dan View Razor

Mata kuliah: Pemrograman Berbasis Kerangka Kerja

Buku: Irfan Subakti, *Pemrograman Berbasis Kerangka Kerja: ASP.NET Core, Laravel, dan Spring Boot*, Bab 3

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini Bu Wati mendapatkan situs web yang benar-benar dilihat pelanggannya: beranda, daftar layanan, penawaran harga, daftar gerai, dan halaman galat yang sopan. Kita membangunnya sebagai LaundryMvc dengan ASP.NET Core 10, hanya dengan kerangka kerja bersama yang ikut dalam SDK, dan berkenalan dengan host, pipeline middleware, kedua jenis perutean, controller, model binding, dan view Razor. Buku: Bab 3, Subbab 3.1–3.12.

Cakupan Pertemuan Ini

1. **Daftar keinginan** — delapan keinginan Bu Wati di balik nota pesanan.
2. ***Host* dan *pipeline*** — `Program.cs`: layanan dulu, lalu *middleware*.
3. **Model** — *record*, aturan harga, *view model*, model formulir, data di balik antarmuka.
4. **Controller dan perutean** — rute konvensional, rute atribut, batasan, *model binding*.
5. **View Razor** — *layout*, *partial view*, *section*, *tag helper*; lalu situs yang berjalan, diperiksa alamat demi alamat.

LaundryMvc dibangun dan dijalankan di sini dengan .NET 10; tangkapan layarnya adalah aplikasi kita sendiri yang sedang berjalan, dan kode statusnya berasal dari eksekusi sungguhan `verify.sh` milik bab ini. Buku: Bab 3.

Situs Web Bu Wati

Beranda dengan tiga layanan yang paling sering ditanyakan; **daftar layanan** dan satu halaman per layanan; layanan **sampai harga tertentu**.

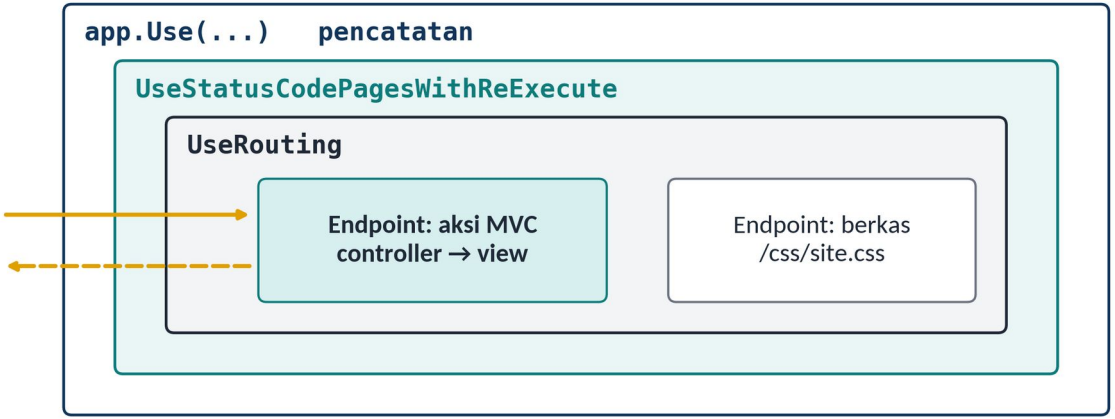
Penawaran harga dengan aturan toko: minimal 3 kg untuk layanan per berat, barang dihitung utuh, antar-jemput Rp 10.000 tetapi gratis mulai Rp 100.000.

Daftar gerai, halaman **tentang** dengan jam buka, **tanpa halaman galat yang jelek**, dan untuk Dimas data layanan dalam **JSON**.

Dua aturan yang dipelajari dengan susah payah: aturan harga tinggal di satu tempat, dan setiap halaman tampil seragam tanpa menyalin *header*.

Bab 1 menyebut aturan pertama "jaga controller tetap ramping"; Bab 2 memperlihatkan bahwa aturan kedua adalah tugas layout. Datanya sama dengan sebelumnya, `data/Service.csv` dan `data/Outlet.csv`; Bab 5 memindahkannya ke basis data. Buku: Subbab 3.1.

Host dan *Pipeline*

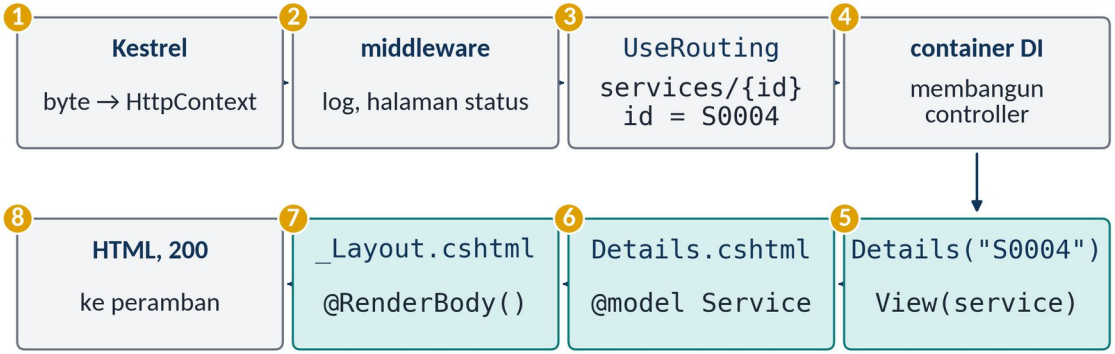


kuning: permintaan masuk melewati setiap lapisan; respons keluar lewat lapisan yang sama (putus-putus)

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

Gambar 3.1: ***host*** *menyalakan Kestrel, membaca konfigurasi, serta menyiapkan pencatatan log dan container DI.* ***Middleware*** *adalah potongan kode yang menerima permintaan, boleh mengerjakan sesuatu, memanggil middleware berikutnya, dan boleh mengerjakan sesuatu lagi dalam perjalanan pulang. Middleware membungkus middleware seperti lapisan bawang, sehingga urutan penambahannya adalah urutan kerjanya. Buku: Subbab 3.2.1.*

Satu Permintaan Melalui MVC



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*.

Gambar 3.2 adalah gambar Bab 2 yang menjadi nyata. Kestrel mengubah byte menjadi `HttpContext`; perutean memilih `ServicesController.Details`; container membangun controller beserta katalognya; aksi mengembalikan view; Razor mengisinya, memasukkannya ke layout, dan menulis HTML. Buku: Subbab 3.2.2.

Membuat Proyeknya

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

Templatnya memberi `Program.cs`, folder `Controllers`, `Models`, `Views`, dan `wwwroot`, `appsettings.json`, serta `Properties/launchSettings.json`.

Kita mengubah sedikit: port menjadi `http://localhost:4030` di lingkungan Development, dan kita menambah folder `Data` serta berkas-berkas CSV.

`appsettings.json` mengatur tingkat log: `Information` secara bawaan, `Warning` untuk pesan kerangka kerja sendiri, agar baris kita menonjol.

Lingkungan *(`Development`, `Production`) mengubah perilaku: di Development ditampilkan halaman pengecualian yang rinci; di Production ditampilkan halaman galat yang ramah. Buku: Subbab 3.3.*

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

Kode 3.1 `Program.cs` yang dipersingkat: layanan didaftarkan pada *builder*, lalu *pipeline* dibangun pada aplikasi.

```
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) => { /* time and log the request */ });
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();
```

Kode 3.1: `AddControllersWithViews()` *mendaftarkan semua yang dibutuhkan MVC. Dua baris berikutnya adalah container MiniFrame dalam wujud sungguhan: satu* `CsvCatalog`*, dibangun sekali oleh sebuah fungsi karena konstruktornya butuh nama folder, dan satu*

`PriceCalculator` yang bisa dibangun container sendiri. Keduanya singleton karena keduanya aman dibagi. Middleware pertama membungkus semua yang lain, sehingga bisa mengukur waktu setiap permintaan. Buku: Subbab 3.4.

Model: Data dan Aturan

Kode 3.2 Inti `Models/Service.cs`: sebuah layanan dengan kode, deskripsi, dan harganya, serta tiga properti yang diturunkan dari deskripsi.

```
// 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";
}
```

Kode 3.2: `record` di C# adalah kelas kecil yang tidak bisa diubah dengan kesamaan berdasarkan nilai. Di sampingnya model berisi `Outlet`, `Money.Rp` (36000 menjadi "Rp 36.000" di satu tempat), dan `PriceCalculator`, aturan toko, yang mengembalikan penawaran beserta penjelasan setiap aturan yang diterapkan. Buku: Subbab 3.5.1 dan 3.5.2.

View Model, Model Formulir, dan Antarmuka

`HomeViewModel(Popular, ServiceCount, OutletCount)` — persis yang ditampilkan beranda, tidak lebih.

`QuoteForm` — `ServiceId`, `Quantity`, `Pickup`, diisi oleh ***model binding***; `Result`-nya memakai `[BindNever]`, sehingga tidak ada URL yang bisa mengisinya.

`ILaundryCatalog` — `Services()`, `FindService(id)`, `Outlets()`, `FindOutlet(id)`: apa yang boleh ditanyakan controller. `CsvCatalog` mengimplementasikannya di atas dua berkas CSV, dibaca sekali.

Komentar di antarmuka itu mengatakan segalanya: Bab 5 mengganti berkas CSV di baliknya dengan basis data sungguhan tanpa menyentuh satu controller pun. Itulah injeksi dependensi yang membayar dirinya sendiri. Buku: Subbab 3.5.3 dan 3.5.4.

Perutean Konvensional

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

`/` berarti `Home/Index`; `/Outlets` berarti `Outlets/Index`; `/Outlets/Details/002` memanggil `Details("002")`. Pencocokannya tidak membedakan huruf besar dan kecil.

`OutletsController(ILaundryCatalog catalog)` menerima katalognya lewat **konstruktor primer**; `Details` mengembalikan `NotFound()` untuk kode yang tidak dikenal dan `View(outlet)` bila ada.

Kelas controller harus publik, tidak abstrak, dan namanya berakhiran `Controller`.

Satu pola melayani setiap controller yang mengikuti konvensi: cocok untuk halaman sederhana yang URL-nya boleh mencerminkan kode. Buku: Subbab 3.6.1.

Perutean Atribut: `ServicesController`

Kode 3.3 Tiga aksi pertama `Controllers/ServicesController.cs`: rute atribut dan sebuah batasan.

```
[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);
    }
```

Kode 3.3: awalan kelas `services` ditambahkan ke templat setiap aksi. `{max:int:min(1000)}` menggabungkan dua batasan. `Under` memakai ulang view `Index`. Controller lengkapnya menambahkan `Quote` di `services/quote` dan aksi JSON di `/api/services/{id}`; templat yang diawali `/` mengabaikan awalan. Begitu controller punya rute atribut, rute konvensional tidak lagi menjangkaunya. Buku: Subbab 3.6.2.

Apa yang Dikembalikan Aksi, dan *Model Binding*

Hasil: `View(model)`, `NotFound()` dengan atau tanpa isi, `Json(...)` atau `Ok(...)`, pengalihan. Aksi menyatakan *apa*; kerangka kerja mengurus *bagaimana*.

Model binding mengisi parameter dan model formulir dari rute, *query string*, dan formulir: `/services/quote?serviceId=S0001&quantity=2.5&pickup=true` menjadi sebuah `QuoteForm`.

Galat konversi (`quantity=abc`) dan pemeriksaan aksi sendiri (`AddModelError`) terkumpul di **`ModelState`**; hanya bila valid, `Quote` memanggil kalkulator.

Membaca `Request.Query["quantity"]` sendiri memang bisa, tetapi melewati konversi, `ModelState`, dan pengembalian nilai ke isian formulir. Nyatakan parameter atau kelas model saja. Buku: Subbab 3.6.3 dan 3.6.4.

View Razor: *Layout, Partial, Section*

Kode 3.4 `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>
```

*Kode 3.4: `@` berpindah dari HTML ke C#, dan setiap nilai yang ditulisnya **di-encode sebagai HTML** secara bawaan. `_ViewImports.cshtml` membawa namespace dan tag helper ke setiap view; `_ViewStart.cshtml` memberi setiap view layout `_Layout`. Beranda merender partial ini sekali per layanan populer; halaman tentang menambahkan jam buka ke footer milik layout melalui sebuah **section**. Buku: Subbab 3.7.1–3.7.3.*

Tag Helper

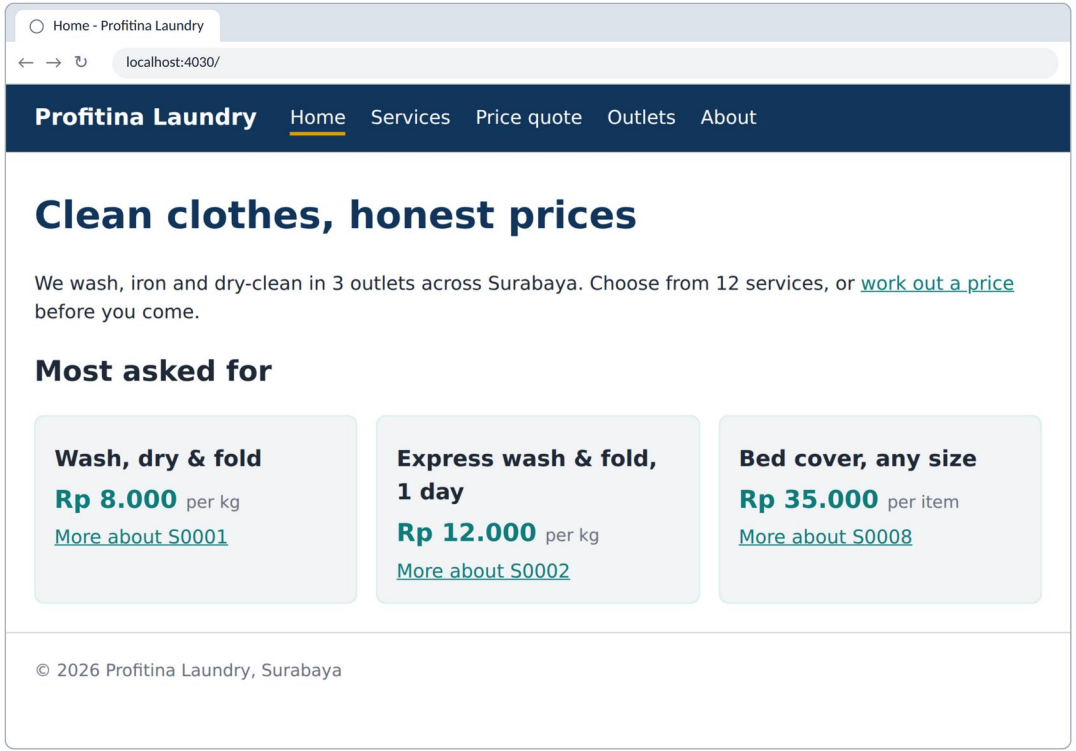
`<a asp-action="Details" asp-route-id="@s.Id">` meminta URL kepada *sistem perutean*: `/services/S0004` untuk rute atribut, `/Outlets/Details/002` untuk rute konvensional. Bila rute berubah, semua tautan ikut.

`<input asp-for="Quantity" />` menulis `name` dan `id` yang dicari *binder*, mengembalikan nilainya, dan menandai isian bila `ModelState` berisi galat.

`bool` menjadi kotak centang plus `Pickup=false` tersembunyi, sehingga kotak yang tidak dicentang tetap mengirim nilai. `<div asp-validation-summary="All">` mendaftar semua galat.

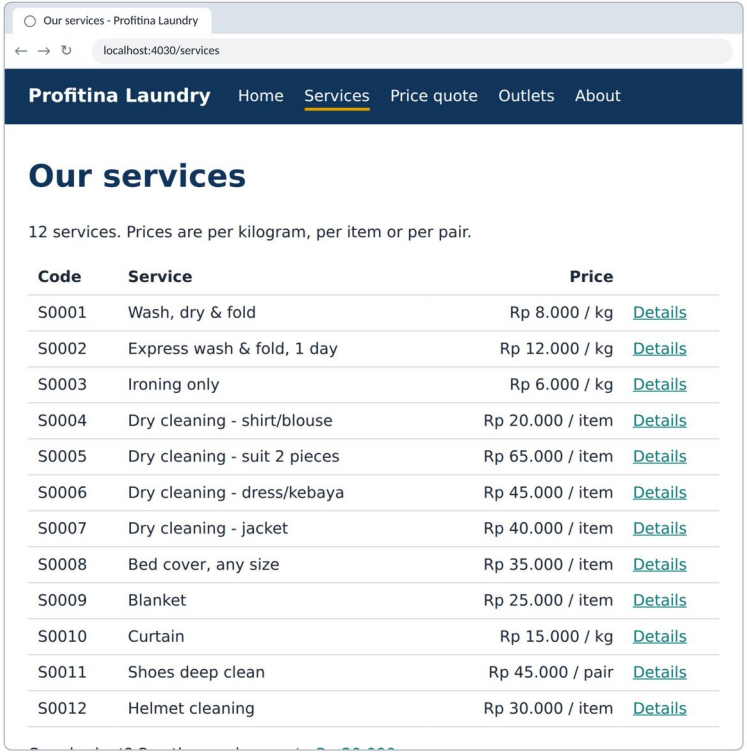
Tag helper adalah alasan view di bab ini hampir tidak berisi URL yang ditulis tangan. Buku: Subbab 3.7.4.

Menjalankannya: Beranda



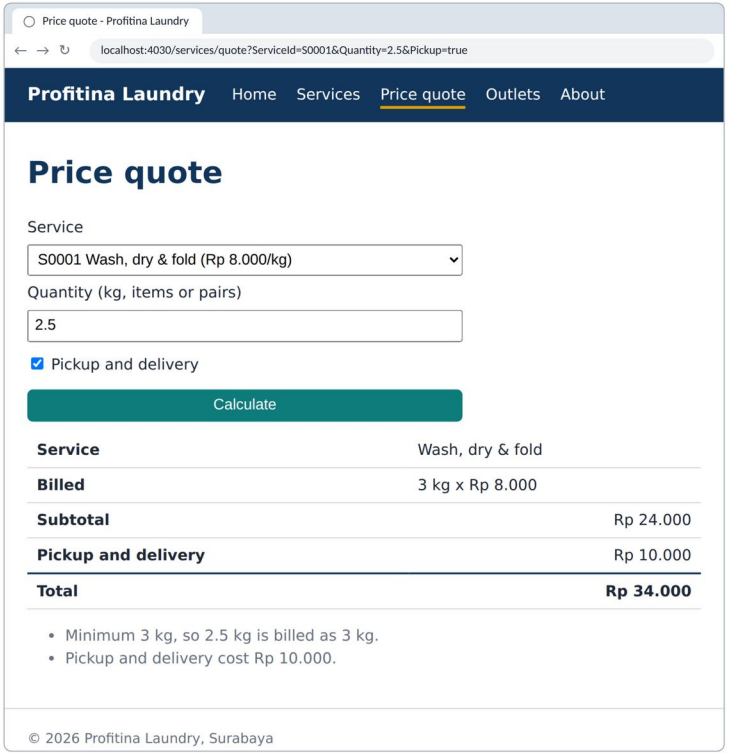
Gambar 3.3 Halaman beranda LaundryMvc: kepala halaman dan menu dari *layout*, teks view dengan hitungan dari *view model*, dan tiga kartu yang digambar *partial view* `_ServiceCard`.

*Gambar 3.3: jalankan dengan `dotnet run` di folder proyek, atau dengan **Run** di VS Code, lalu buka `http://localhost:4030`. Tunjukkan bagian mana yang berasal dari layout, dari view, dan dari partial. Daftar layanan di `/services` adalah layout yang sama dengan sebuah tabel. Buku: Subbab 3.8.1 dan 3.8.2.*



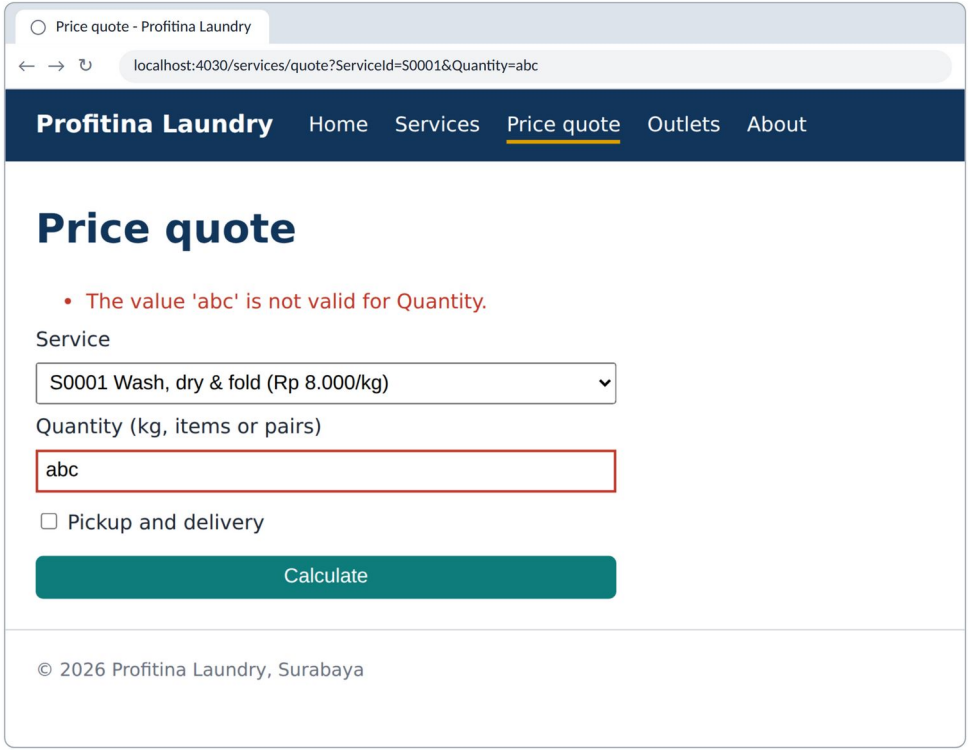
Gambar 3.4 Daftar layanan di `/services`, dirender oleh `Views/Services/Index.cshtml`.

Sebuah Penawaran, dan Penawaran yang Keliru



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 3.5 dan Gambar 3.4: penawaran untuk 2,5 kg cuci, kering, dan lipat dengan antar-jemput, yang lebih dulu dihitung dengan tangan: ditagih 3 kg, subtotal Rp 24.000, antar-jemput Rp 10.000, total Rp 34.000. Situs sepakat dengan hitungan tangan, dan formulir tetap memuat pilihan karena tag helper menuliskan kembali nilai yang terikat. Setiap baris penjelasan datang dari kalkulator, bukan dari view. Buku: Subbab 3.8.2.



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

Gambar 3.6: tidak ada yang menulis kode untuk pesan ini; model binding menaruhnya di `ModelState` dan tag helper menampilkannya. Buku: Subbab 3.8.2.

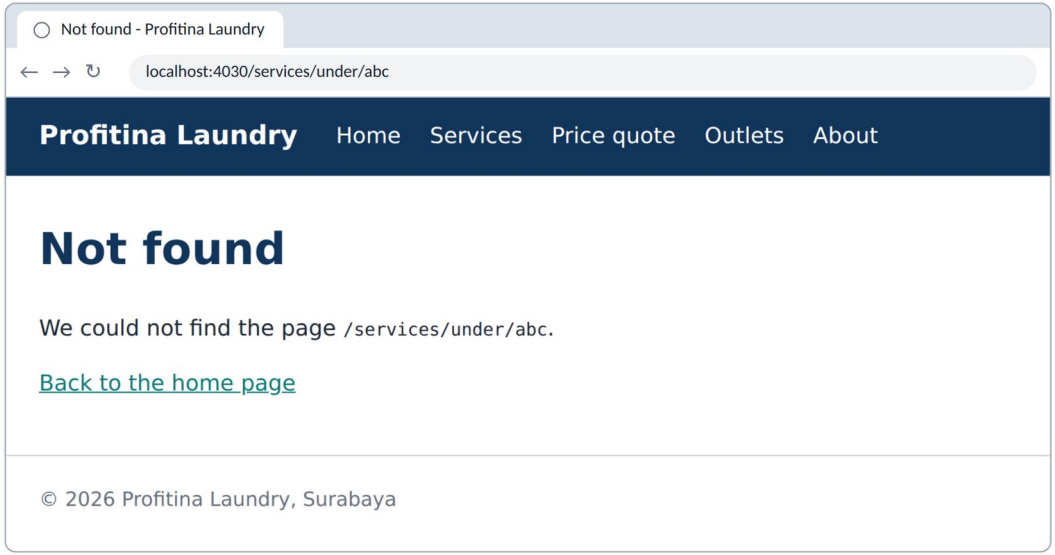
Setiap Alamat, Diperiksa

Tabel 3.1 Kode status yang dikembalikan LaundryMvc yang berjalan untuk alamat-alamat di bab ini, dari `verify.sh`.

Status	Alamat	Alasan
200	`/`, `/Home`, `/home/index`	rute konvensional dengan nilai bawaan
200	`/services/quote`	literal `quote` menang atas `{id}`
200	`/services/under/20000`	`int` dan `min(1000)` terpenuhi
404	`/services/under/abc`, `/services/under/500`	batasan gagal, tidak ada rute
404	`/Outlets/Details/009`	aksi mengembalikan `NotFound()`
404	`/nothing/here`	tidak ada controller bernama `Nothing`
200	`/api/services/S0004`	rute atribut absolut, JSON

Tabel 3.1: `verify.sh` membangun aplikasi, menyalakannya, dan menanyakan setiap alamat dengan `curl`. Perhatikan `/services/quote`: segmen literal lebih diutamakan daripada parameter `{id}`, apa pun urutannya di berkas. Buku: Subbab 3.8.3.

404 yang Ramah dan Tetap Jujur



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

Gambar 3.7: `UseStatusCodePagesWithReExecute("/error/{0}")` *menjalankan pipeline sekali lagi untuk* `/error/404`*, dan* `HomeController.Error` *mengatur* `Response.StatusCode = code`*. Orang melihat halaman bergaya toko; peramban, mesin*

pencari, dan alat pemantau tetap melihat 404. Endpoint JSON menjawab galat dengan JSON, sehingga program lain tidak pernah menerima HTML. Buku: Subbab 3.8.2 dan 3.9.2.

Log

```
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 /nothing/here -> 404 in 1 ms
```

Middleware pertama kita menulis baris-baris ini selama eksekusi sungguhan. Ia berada di luar middleware kode status, sehingga mencatat jalur yang diketik pengunjung dan status hasilnya, tidak pernah `/error/404` *yang dijalankan ulang. Permintaan pertama setelah aplikasi menyala butuh sekitar 200 ms karena view dan katalog dimuat saat pertama dipakai. Buku: Subbab 3.8.4.*

Menjadikannya Lebih Baik: Tanpa Controller Gemuk

```
// 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;
```

Kode ini berjalan dan tampak lebih pendek, tetapi 3, 100000, dan 10000 kini tinggal di tempat yang tak terjangkau *endpoint* JSON maupun pengujian, dan penjelasannya harus dibangun ulang di view.

Hari ketika layar kasir di Bab 5 butuh aturan yang sama, aturannya akan disalin, dan salinan-salinannya akan saling menjauh.

Uji untuk setiap aturan: bisakah endpoint JSON dan unit test memakainya tanpa permintaan HTTP? Bila tidak, aturannya berada di tempat yang salah. Begitulah dulu situs lama Bu Wati menampilkan dua harga setrika. Yang masih kurang: basis data (Bab 5), atribut validasi dan login (Bab 6). Buku: Subbab 3.9.

Memakainya Sungguhan, dan Kesalahan yang Sering Terjadi

Usaha Anda sendiri di atas LaundryMvc: ganti dua berkas CSV dan aturan di `PriceCalculator`; controller dan view hanya sedikit berubah.

Kesalahan: controller yang tidak ditemukan kerangka kerja; view yang tidak ditemukan (`Detail` bukan `Details`); mencampur rute konvensional dan rute atribut pada satu controller.

Kesalahan: *middleware* dalam urutan yang salah; membaca *query string* sendiri; aturan bisnis di controller atau view.

> **PERINGATAN:** `@Html.Raw` mematikan *encoding* Razor. Pakai hanya untuk HTML yang dihasilkan dan dipercaya program Anda sendiri, jangan untuk apa pun yang diketik pengguna.

Bila view tidak ditemukan, pesan galatnya mendaftar setiap tempat yang dicari; bandingkan nama aksi dengan nama berkas huruf demi huruf. Tulis pipeline dari luar ke dalam: pencatatan dan penanganan galat dulu, lalu halaman status, lalu perutean. Buku: Subbab 3.10 dan 3.11.

Soal Terselesaikan: Memindahkan Pencatat ke Dalam

```
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
```

Letakkan `app.Use(...)` untuk pencatatan *sesudah* `UseStatusCodePagesWithReExecute`: kini ia berada di dalam, dan melihat permintaan asli sekaligus `/error/404` yang dijalankan ulang.

Setiap alamat yang salah kini muncul dua kali di log. Tidak ada yang rusak bagi pengunjung; hanya log yang lebih sulit dibaca dan setiap galat terhitung ganda.

Ini adalah baris sungguhan dari eksekusi di buku. Pelajarannya adalah bawang pada Gambar 3.1: apa yang dilihat sebuah middleware bergantung pada lapisan mana yang berada di luarnya. Karena itu, tempatkan pencatatan di lapisan terluar bila Anda ingin satu baris per kunjungan, dan di dalam bila Anda justru ingin melihat setiap eksekusi ulang. Buku: Soal Latihan 3.4 dan penyelesaiannya di Subbab 3.14.

Ringkasan: Lima Hal untuk Diingat

1. `Program.cs` membangun *host*, mendaftarkan layanan, membangun *pipeline*, dan menjalankan Kestrel, dalam urutan itu (Gambar 3.1).
2. *Middleware* membungkus *middleware*; masing-masing bekerja sebelum dan sesudah sisanya, sehingga urutan penting.
3. Model menyimpan data dan aturan; akses data berada di balik `ILaundryCatalog`, terdaftar sebagai *singleton*.
4. Controller dijangkau lewat rute konvensional atau rute atribut dengan awalan, batasan, dan prioritas; *model binding* mengisi parameter dan mencatat galat di `ModelState`.
5. Razor meng-*encode* secara bawaan, berbagi *layout*, memakai ulang *partial*, dan *tag helper* membangun URL dari rute.

Situs yang berjalan sepakat dengan hitungan tangan dan setiap rute menjawab seperti yang diperkirakan; satu-satunya kejutan, “2 item”, menjadi soal latihan. Buku: Subbab 3.12.

Pertemuan Berikutnya

Bab 4 — Evaluasi dan Pembahasan I. Proyek ASP.NET Core kecil yang nyata, diselesaikan secara lengkap dengan semua yang dipelajari di Bab 1 sampai 3.

Sebelum itu: jalankan LaundryMvc dan `verify.sh`, lalu kerjakan soal latihan bab ini, termasuk memindahkan *middleware* pencatatan ke dalam *middleware* kode status.

Soal-soal lainnya, termasuk kejutan “2 item”, beserta penyelesaian lengkapnya ada di buku. Buku: Subbab 3.13 dan 3.14.

Kuliah 4 — Evaluasi dan Pembahasan I: Situs Etalase Profitina Laundry

Mata kuliah: Pemrograman Berbasis Kerangka Kerja

Buku: Irfan Subakti, *Pemrograman Berbasis Kerangka Kerja: ASP.NET Core, Laravel, dan Spring Boot*, Bab 4

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini kita menyelesaikan lima soal nyata dengan semua yang dipelajari di Bab 1 sampai 3: situs etalase, pelacakan pesanan, tabel rute yang ditebak dengan tangan, header waktu respons, dan mode pemeliharaan. Untuk setiap soal kita mengikuti rutinitas yang sama, melihat penyelesaiannya, menjalankannya, lalu mempelajari penyelesaian keliru yang menggoda, yang masih mungkin ditulis pengembang yang cermat sekalipun. Buku: Bab 4, Subbab 4.1–4.10.

Cakupan Pertemuan Ini

1. **Perkakas dan rutinitas** — enam langkah untuk soal kerangka kerja apa pun.
2. **Soal 4.1** — situs etalase: lima halaman di bawah `/laundry`, kata-kata di JSON, angka dari data.
3. **Soal 4.2** — “Di mana cucian saya?”: batasan `regex`, formulir `GET`, privasi.
4. **Soal 4.3** — tabel rute secara manual, dicocokkan dengan daftar milik router sendiri.
5. **Soal 4.4 dan 4.5** — *header* waktu respons dan mode pemeliharaan: *middleware* yang benar.

Seluruh aplikasi, LaundryShowcase, dibangun dan dijalankan dengan .NET 10; setiap kode status, header, dan pesan galat yang ditampilkan hari ini adalah keluaran sungguhan, termasuk kegagalan dari penyelesaian yang keliru. Buku: Bab 4.

Perkakasnya

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

Soal	Topik	Perkakas utama dari Bab 1–3
4.1	Situs etalase	rute atribut dengan awalan, *layout*, DI untuk konten dan data
4.2	Di mana cucian saya?	batasan `regex`, formulir `GET`, `RedirectToAction`, halaman 404
4.3	Tabel rute secara manual	pencocokan rute, prioritas, huruf besar-kecil
4.4	*Header* waktu respons	*middleware*, `Response.OnStarting`
4.5	Mode pemeliharaan	konfigurasi, urutan *middleware*, status 503

Tabel 4.1 adalah peta: bila middleware masih terasa samar, mulailah dari Soal 4.4 dan 4.5; bila perutean yang samar, mulailah dari 4.3. Menamai perkakasnya sudah separuh pekerjaan. Buku: Subbab 4.1.

Cara Menghadapi Soal Kerangka Kerja



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

*Gambar 4.1: (1) baca dan nyatakan ulang soal dengan kata-kata sendiri; (2) tulis **tabel alamat**, termasuk alamat yang harus gagal; (3) sketsakan halaman dan data di kertas; (4) tulis controller yang ramping dengan aturan di dalam layanan; (5) jalankan dan uji dengan peramban dan `curl`; (6) tengok ke belakang untuk mencari penyelesaian keliru yang menggoda. Langkah 2 paling sering dilewati pemula, padahal langkah itulah yang menangkap sebagian besar kesalahan. Buku: Subbab 4.2.*

Soal 4.1: Situs Etalase

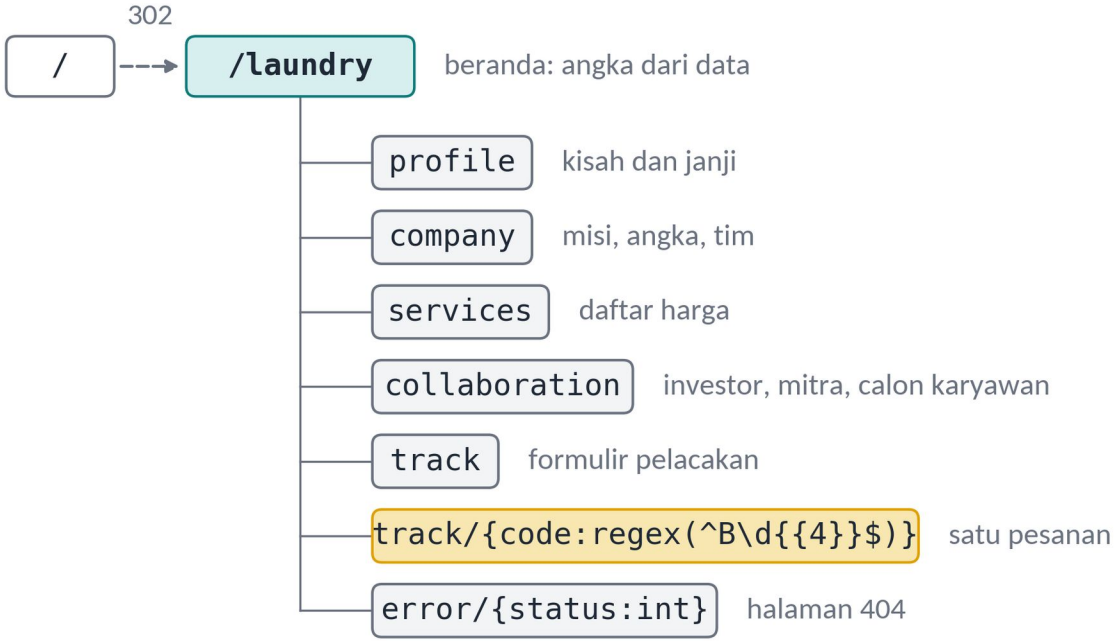
Profitina Laundry ingin situs yang memperkenalkan usahanya kepada calon investor, mitra, dan calon karyawan: **beranda**, **profil**, **perusahaan**, **layanan**, **kerja sama**.

Setiap halaman berada di bawah `/laundry`; semua kata yang tidak dihitung tinggal di satu berkas, `content/site.json`; setiap angka di halaman dihitung dari data.

Tabel alamat menambahkan tepi yang jarang disebut soal: `/` dialihkan dengan 302 ke `/laundry`; `/profile` dan `/Showcase/Profile` harus menjawab 404; `/laundry/Profile` menjawab 200 karena pencocokan tidak membedakan huruf besar-kecil.

Mengapa berkas JSON untuk kata-kata? Karena Bu Wati lebih sering mengubah slogan daripada kode, dan berkas teks bisa diubah tanpa menyentuh C#. Mengapa angka dihitung? Karena “hampir 3.000 pesanan setahun” yang diketik ke halaman sudah salah sehari setelah ditulis. Buku: Subbab 4.3.1 dan 4.3.2.

Merancangnya di Kertas



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.

Gambar 4.2: satu `ShowcaseController` dengan `[Route("laundry")]`. `SiteContent` (bentuk `site.json` sebagai record C#) dan `LaundryData` (layanan, gerai, dan pesanan, dimuat sekali, dengan metode `Facts()`) didaftarkan sebagai singleton. `_ViewImports.cshtml` memakai `@inject` agar setiap view, termasuk layout, bisa membaca konten situs. Buku: Subbab 4.3.3 dan 4.3.4.

Penyelesaiannya: Satu Controller yang Ramping

Kode 4.1 Lima aksi pertama `Controllers/ShowcaseController.cs`: satu controller dengan rute atribut untuk seluruh etalase.

```
[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());

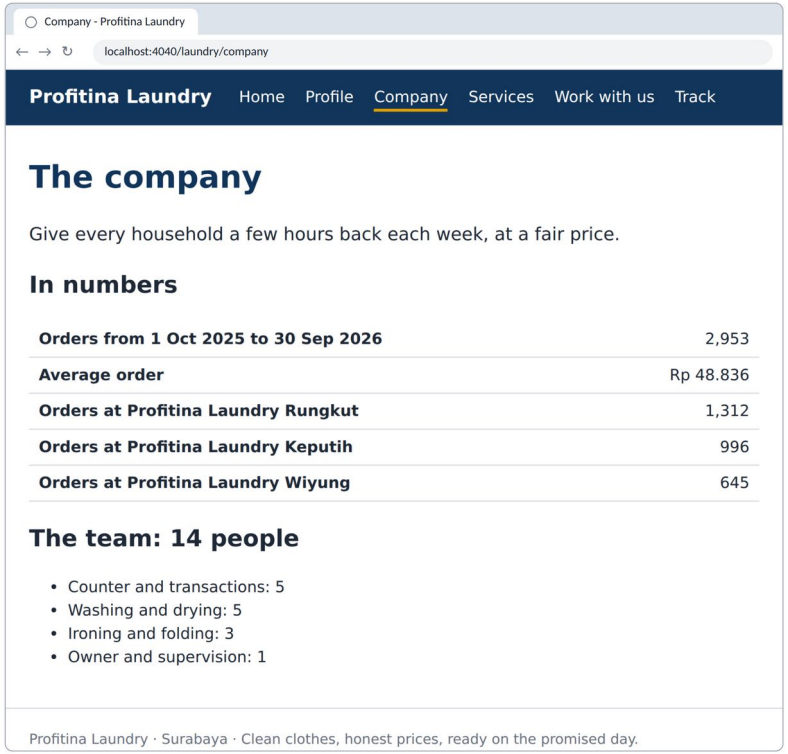
    [HttpGet("company")]                            // /laundry/company
    public IActionResult Company() => View(data.Facts());

    [HttpGet("services")]                           // /laundry/services
    public IActionResult Services() => View(data.Services);

    [HttpGet("collaboration")]                      // /laundry/collaboration
    public IActionResult Collaboration() => View(site.Collaboration);
```

Kode 4.1: setiap aksi hanya satu baris: pilih datanya, pilih view-nya. Konten dan data tiba lewat konstruktor primer, didaftarkan sekali di `Program.cs`, yang juga memetakan `/` ke sebuah pengalihan dan daftar rute khusus pengembangan. View membaca kata-kata dari `SiteContent` dan angka dari fakta; tidak ada view yang berisi angka ketikan tangan. Buku: Subbab 4.3.4.

Menjalankannya: Angka dari Data



Gambar 4.3 Halaman perusahaan: setiap angka dihitung `LaundryData.Facts()` dari data pesanan.

Gambar 4.3: misi dan tim berasal dari `site.json`; angka-angkanya dihitung saat aplikasi menyala. Beranda mengambil slogannya dari berkas JSON dan hitungannya dari data, dan halaman kerja sama mendaftar apa yang ditawarkan dan diminta usaha ini dari setiap kalangan. Buku: Subbab 4.3.5.

Penyelesaian Keliru yang Menggoda untuk 4.1

Satu garis miring di depan: `[HttpGet("/profile")]` alih-alih `[HttpGet("profile")]` pada aksi.

Templat yang diawali `/` mengabaikan awalan controller: halaman kini menjawab di `/profile`, dan `/laundry/profile` menjawab 404.

Tidak ada peringatan sama sekali. Hanya tabel alamat, yang dicocokkan dengan situs yang berjalan, yang menangkapnya.

Inilah alasan langkah 2 ada. Baris `/profile` dan `/laundry/profile` di tabel alamat langsung membongkarnya. Buku: Subbab 4.3.6 dan 4.3.7.

Soal 4.2: Di Mana Cucian Saya?

Pelanggan mengetik kode di notanya, misalnya `B0072`, lalu melihat statusnya: antre, siap di rak, atau sudah diambil.

Halaman publik **tidak menampilkan data pribadi**: layanan, gerai, tanggal, dan jumlah, tetapi tidak pernah nama atau nomor telepon pelanggan.

Dihitung dengan tangan lebih dulu: tiga pesanan dari `Bill.csv`, satu per status. Jumlah untuk B0072 cocok: 7 kg kali Rp 8.000 adalah Rp 56.000.

Siapa pun yang menebak sebuah kode bisa membuka halamannya, sehingga halaman itu harus aman dilihat orang asing. Aturan privasi ini bagian dari rancangan, bukan tambahan belakangan. Buku: Subbab 4.4.1 dan 4.4.2.

Rancangannya: Batasan dan Pengalihan

Rute `track/{code:regex(^B\d{{4}}$)}`: di dalam templat rute, kurung kurawal ekspresi reguler **digandakan**, karena kurung kurawal tunggal berarti “parameter”.

Formulirnya memakai `method="get"`; aksinya mengalihkan ke `/laundry/track/B0072`, alamat yang bisa disimpan sebagai *bookmark* dan dibagikan.

Kode yang tidak dikenal dan kode yang bentuknya salah sama-sama mendapat halaman 404 yang ramah, dengan status 404 tetap terjaga.

View galat menampilkan alamat yang diminta, di-encode oleh Razor, sehingga kode yang aneh pun tidak berbahaya di halaman. Buku: Subbab 4.4.3 dan 4.4.4.

Penyelesaiannya: Dua Aksi untuk Pelacakan

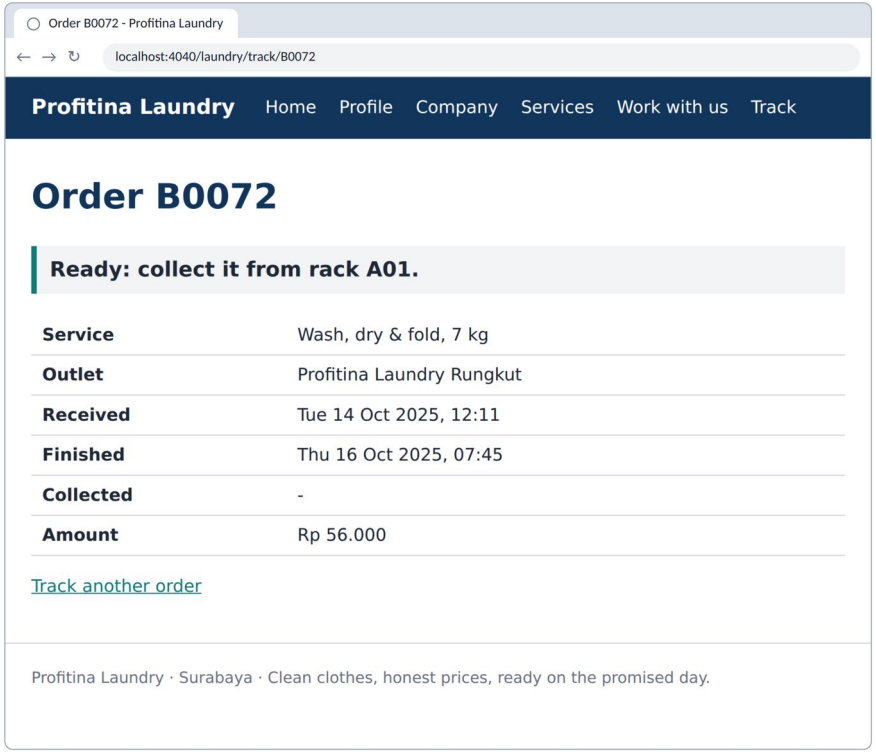
Kode 4.2 Aksi-aksi pelacakan di `ShowcaseController` (sisa `Track` mencari nama layanan dan gerai untuk view).

```
    [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() });
    }

    [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();
        }
```

Kode 4.2: `TrackForm` *menampilkan formulir kosong, atau mengalihkan ke alamat yang bersih dengan* `RedirectToAction`*, sehingga URL yang dibangun dari rute selalu benar.* `Track` *mengembalikan* `NotFound()` *untuk kode yang tidak dikenal, dan middleware kode status mengubahnya menjadi halaman yang ramah. View* `Track.cshtml` *menampilkan status dalam kata-kata, "Ready: collect it from rack A01.", dan tabel fakta pesanan. Buku: Subbab 4.4.4.*

Menjalankannya



Gambar 4.4 Halaman pelacakan pesanan B0072: siap di rak A01, dengan layanan, gerai, tanggal, dan jumlahnya, tanpa apa pun tentang pelanggan.

```
200  /laundry/track
302  /laundry/track?code=b0072
200  /laundry/track/B0072
200  /laundry/track/b0001
404  /laundry/track/B9999
404  /laundry/track/X12
404  /laundry/track/B00721
```

Gambar 4.4: formulir mengalihkan dengan 302; kode huruf kecil juga cocok, karena batasan rute tidak membedakan huruf besar-kecil; kode berbentuk benar yang tidak ada, bentuk yang salah, dan kode yang satu digit terlalu panjang semuanya mendapat 404. Buku: Subbab 4.4.5.

Penyelesaian Keliru yang Menggoda untuk 4.2

```
[HttpGet(@"track/{code:regex(^B\d{4}$)}")]     // WRONG: needs {{4}}
public string Track(string code) => $"order {code}";
```

```
Program.cs(14,28): warning ASP0017: Route issue: In a route parameter, '{' and '}' must be escaped with
'{{' and '}}'. (https://aka.ms/aspnet/analyzers)
```

Pembangunan memberi peringatan ASP0017, dan bila peringatan itu diabaikan aplikasi berhenti saat menyala dengan `RoutePatternException`*: tidak ada satu permintaan pun yang dijawab. Perlakukan peringatan rute sebagai galat. Buku: Subbab 4.4.6 dan 4.4.7.*

Soal 4.3: Tabel Rute secara Manual

Tebak di kertas status untuk dua belas permintaan ke LaundryShowcase, lalu periksa tebakan itu.

Dua alasan berbeda bersembunyi di balik 404: **tidak ada rute yang cocok**, atau **aksi berjalan dan mengembalikan `NotFound()`**.

Pemeriksaannya: rute khusus pengembangan `/_routes` mendaftar setiap *endpoint* yang dikenal router, diambil dari `EndpointDataSource`.

```
laundry                                  ShowcaseController.Home
laundry/services                         ShowcaseController.Services
laundry/track                            ShowcaseController.TrackForm
laundry/track/{code:regex(^B\d{{4}}$)}   ShowcaseController.Track
```

Cuplikan dari daftar sungguhan. Minta kelas menebak `/laundry/Profile` (200, huruf besar-kecil diabaikan), `/laundry/track/X12` (404 dari perutean: batasannya gagal), dan `/laundry/track/B9999` (404 dari aksi: pesanannya tidak ada). Tabel di buku menebak dua belas permintaan, dan setiap tebakan terbukti benar. Buku: Subbab 4.5.1–4.5.5.

Penyelesaian Keliru yang Menggoda untuk 4.3

```
[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}";
```

```
500  /laundry/track/B0072
     Microsoft.AspNetCore.Routing.Matching.AmbiguousMatchException: The request matched multiple
endpoints. Matches:
```

Nama parameter tidak membedakan rute; hanya segmen literal dan batasan yang membedakannya. Pembangunan memberi peringatan (ASP0023) dan permintaan pertama yang cocok gagal dengan 500. Buku: Subbab 4.5.6 dan 4.5.7.

Soal 4.4: *Header* Waktu Respons

Kode 4.3 `Middleware/ResponseTimeMiddleware.cs`: *header* diatur di dalam *callback* `OnStarting`, tepat sebelum *header* dikirim.

```
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);
    }
}
```

Kode 4.3: HTTP mengirim header sebelum isi, sehingga begitu isi mulai dikirim, header tidak bisa diubah lagi. `OnStarting` *menjalankan kode kita tepat pada saat terakhir.* `curl -s -D - -o /dev/null http://localhost:4040/laundry` *memperlihatkan* `X-Response-Time: 2 ms` *di antara header sungguhan. Buku: Subbab 4.6.1–4.6.5.*

Penyelesaian Keliru yang Menggoda untuk 4.4

```
    await next(context);
    context.Response.Headers["X-Response-Time"] =         // WRONG: too late
        $"{clock.ElapsedMilliseconds} ms";
```

```
System.InvalidOperationException: Headers are read-only, response has already started.
```

Halaman tetap tiba dengan status 200, header-nya diam-diam hilang, dan log mencatat satu pengecualian per permintaan. Hanya membaca log atau melihat header dengan `curl -D -` yang menangkapnya. Buku: Subbab 4.6.6 dan 4.6.7.

Soal 4.5: Mode Pemeliharaan yang Tetap Bergaya

Sakelar `Showcase:Maintenance` di `appsettings.json`, mati secara bawaan, yang bisa dinyalakan tanpa membangun ulang: `Showcase__Maintenance=true dotnet run`.

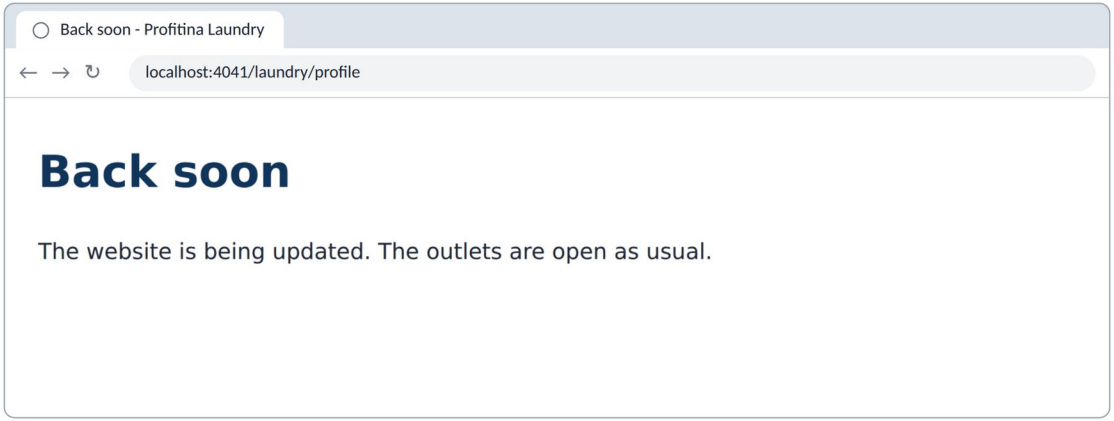
Selama menyala, setiap halaman menjawab **503** dengan `Retry-After: 3600` dan halaman “Back soon”, diberi gaya oleh lembar gaya situs sendiri, yang karenanya tetap harus dilayani.

Middleware-nya diletakkan **sesudah** `UseRouting`, tempat *endpoint* terpilih sudah diketahui, sehingga aset statis bisa diloloskan.

```
503  /laundry
503  /laundry/profile
200  /css/site.css
```

Garis bawah ganda pada variabel lingkungan mewakili titik dua pada kunci konfigurasi; variabel lingkungan mengalahkan `appsettings.json`. Buku: Subbab 4.7.1–4.7.5.

Halaman Pemeliharaan, dan Kembarannya yang Keliru



Gambar 4.5 Halaman pemeliharaan, dijawab dengan status 503 dan diberi gaya oleh lembar gaya situs sendiri, yang diloloskan mode pemeliharaan.

Gambar 4.5: penyelesaian keliru yang menggoda menaruh pemeriksaan sebelum `UseRouting`*. Di sana* `context.GetEndpoint()` *selalu null, sehingga tidak ada yang tampak sebagai aset, dan eksekusi sungguhan menjawab* `503 /css/site.css Back soon`*: halaman tanpa gaya. Menguji lembar gaya selama pemeliharaan menangkapnya. Buku: Subbab 4.7.6 dan 4.7.7.*

Lima Penyelesaian Keliru, Satu Pelajaran

Tabel 4.2 Lima penyelesaian keliru yang menggoda di bab ini, apa akibat masing-masing, dan apa yang menangkapnya.

Kesalahan	Gejala	Tertangkap oleh
`/` di depan rute aksi	alamat salah, 404 di alamat yang benar	tabel alamat
Kurung kurawal tunggal di `regex`	pengecualian saat menyala	peringatan ASP0017
Dua rute berbentuk sama	500 `AmbiguousMatchException`	peringatan ASP0023
Header sesudah `await next`	*header* hilang, pengecualian di log	log; `curl -D -`
Pemeriksaan sebelum `UseRouting`	lembar gaya ikut terblokir	menguji `/css/site.css`

Tabel 4.2: dua tertangkap oleh peringatan, tiga hanya oleh pengujian aplikasi yang berjalan. Tiga kesalahan umum lainnya: mengetik mati angka yang seharusnya berasal dari data, membiarkan perkakas pengembangan seperti `/_routes` menyala di Production (eksekusi Production menjawabnya dengan 404), dan halaman galat ramah yang lupa statusnya lalu menjawab 200. Buku: Subbab 4.9.

Memakainya Sungguhan

Etalase untuk usaha Anda sendiri: ganti `site.json` dan berkas-berkas data; controller dan view tetap.

Halaman pelacakan dan data pribadi: tampilkan hanya apa yang boleh dilihat orang asing yang memegang kode; jangan pernah nama, nomor telepon, atau alamat di halaman yang dijangkau lewat kode yang bisa ditebak.

Untuk penjahit atau bengkel servis, halaman pelacakan yang sama bekerja dengan "siap diambil" sebagai ganti nomor rak. Sakelar pemeliharaan berguna bagi usaha kecil mana pun pada hari ia memindahkan data atau memutakhirkan server: pelanggan melihat halaman sopan bergaya toko, dan alat pemantau melihat 503 yang jujur. Buku: Subbab 4.8.

Ringkasan: Lima Hal untuk Diingat

1. Ikuti rutinitas pada Gambar 4.1; tabel alamat, lengkap dengan alamat yang harus gagal, menangkap sebagian besar kesalahan.
2. Kata-kata di berkas konten, angka dihitung dari data, keduanya diinjeksikan sebagai *singleton*.
3. Gandakan kurung kurawal ekspresi reguler di dalam templat rute; jaga halaman 404 tetap jujur dan halaman publik bebas data pribadi.
4. Periksa tebakan terhadap daftar milik router sendiri, pada rute khusus pengembangan; rute yang ambigu gagal dengan 500.
5. Atur *header* di `Response.OnStarting`; letakkan *middleware* yang peduli *endpoint* sesudah `UseRouting`.

Perhatikan kesamaan kelima soal: tidak ada yang membutuhkan fitur kerangka kerja baru di luar Bab 3. Yang dibutuhkan adalah kecermatan tentang urutan (rute, middleware, header), tentang tepi (alamat yang harus gagal), dan tentang kejujuran (kode status yang mengatakan yang sebenarnya). Kecermatan itulah yang membedakan situs yang berjalan saat demonstrasi dari situs yang berjalan pada Senin yang sibuk. Buku: Subbab 4.10.

Pertemuan Berikutnya

Bab 5 — ASP.NET Core MVC II. Entity Framework Core dan CRUD sungguhan: data pindah ke SQLite, dan meja kasir akhirnya mengingat pelanggan, pesanan, dan pembayaran.

Sebelum itu: jalankan LaundryShowcase, coba sendiri setiap penyelesaian keliru yang menggoda, dan kerjakan soal latihan bab ini.

Buku: Subbab 4.11 dan 4.12.

Kuliah 5 — ASP.NET Core MVC II: Entity Framework Core dan CRUD Sungguhan

Mata kuliah: Pemrograman Berbasis Kerangka Kerja

Buku: Irfan Subakti, *Pemrograman Berbasis Kerangka Kerja: ASP.NET Core, Laravel, dan Spring Boot*, Bab 5

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini toko mendapat ingatan. Kita menghubungkan LaundryMvc ke basis data SQLite melalui Entity Framework Core 10: kelas model, `DbContext`, migrasi, data awal, kueri LINQ, dan halaman create-read-update-delete yang lengkap untuk pelanggan, pesanan, dan pembayaran. Katalog Bab 3 pindah ke basis data tanpa satu pun controller Bab 3 berubah. Buku: Bab 5, Subbab 5.1–5.17.

Cakupan Pertemuan Ini

1. Toko tanpa ingatan — buku nota karbon, dan mengapa situs web belum bisa membantu.

2. Gagasannya — tabel dan objek, saat desain dan saat berjalan, LINQ, mengapa SQLite.

3. Datanya — pelanggan, layanan, pesanan, pembayaran; rumus sisa tagihan; LINQ dulu, tanpa basis data.

4. EF Core — paket, kelas model, `DbContext`, data awal, migrasi.

5. CRUD sungguhan — `CustomersController` aksi demi aksi; pesanan, pembayaran, daftar harga, dan layar kasir.

Catatan jujur tentang apa yang dijalankan: mesin buku ini tidak bisa menjangkau NuGet, sehingga aplikasi web EF Core ditulis dengan sangat cermat berdasarkan dokumentasi resmi dan diperiksa, tetapi tidak di-restore atau dijalankan di sana. Program konsol `LinqDemo` dibangun dan dijalankan, dan keluarannya hari ini adalah keluaran sungguhan. Di komputer Anda, dengan akses internet, semuanya berjalan seperti yang dijelaskan. Buku: Bab 5 dan README-nya.

Toko Tanpa Ingatan

Pukul tujuh pagi di gerai Keputih: sekantong pakaian ditimbang (4,5 kg), nama dan nomor telepon pelanggan ditulis di nota karbon bernomor, kantongnya masuk keranjang biru.

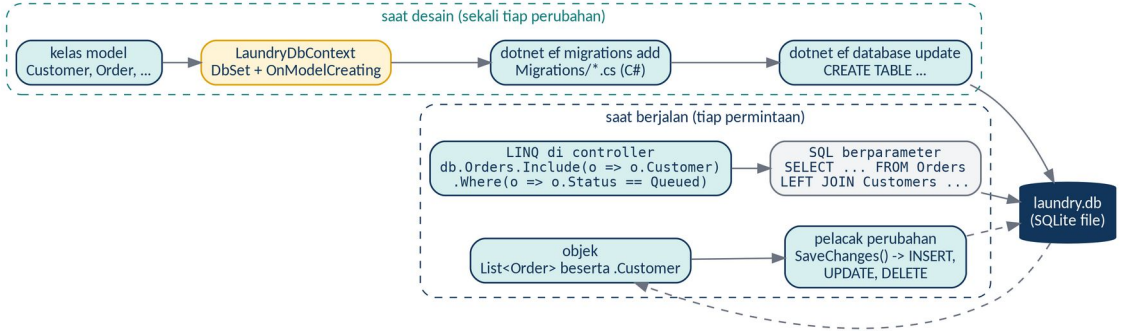
Siang hari seorang pelanggan bertanya apakah cuciannya sudah siap; kasir membolak-balik buku karbon lalu berjalan ke belakang untuk melihat.

Malam hari pemilik menjumlahkan isi laci dan mencocokkannya, halaman demi halaman, dengan nota.

Situs Bab 3 mengenal dua belas layanan dan tiga gerai dari berkas CSV yang dibaca sekali saat menyala. Ia tidak bisa mengingat satu pelanggan pun.

Yang dibutuhkan toko adalah buku karbon, papan harga, dan laci uang di dalam program: setiap pelanggan, pesanan, dan pembayaran dicatat saat terjadi, ditemukan kembali dalam sedetik, dan masih ada esok hari. Tempat itu adalah basis data. Buku: Subbab 5.1.

Tabel di Satu Sisi, Objek di Sisi Lain



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.

*Gambar 5.1: dua saat. Pada **saat desain**, perintah `dotnet ef` membaca kelas-kelas dan menulis **migrasi**, berkas C# yang membuat atau mengubah tabel. Pada **saat berjalan**, kueri **LINQ** pada sebuah `DbSet` diterjemahkan menjadi SQL, baris-barisnya kembali sebagai objek, dan **pelacak perubahan** mengingat apa yang Anda ubah agar `SaveChangesAsync` bisa menuliskannya. LINQ adalah bahasa yang sama, baik pada daftar di memori maupun pada tabel basis data. Buku: Subbab 5.2.1–5.2.3.*

Mengapa SQLite

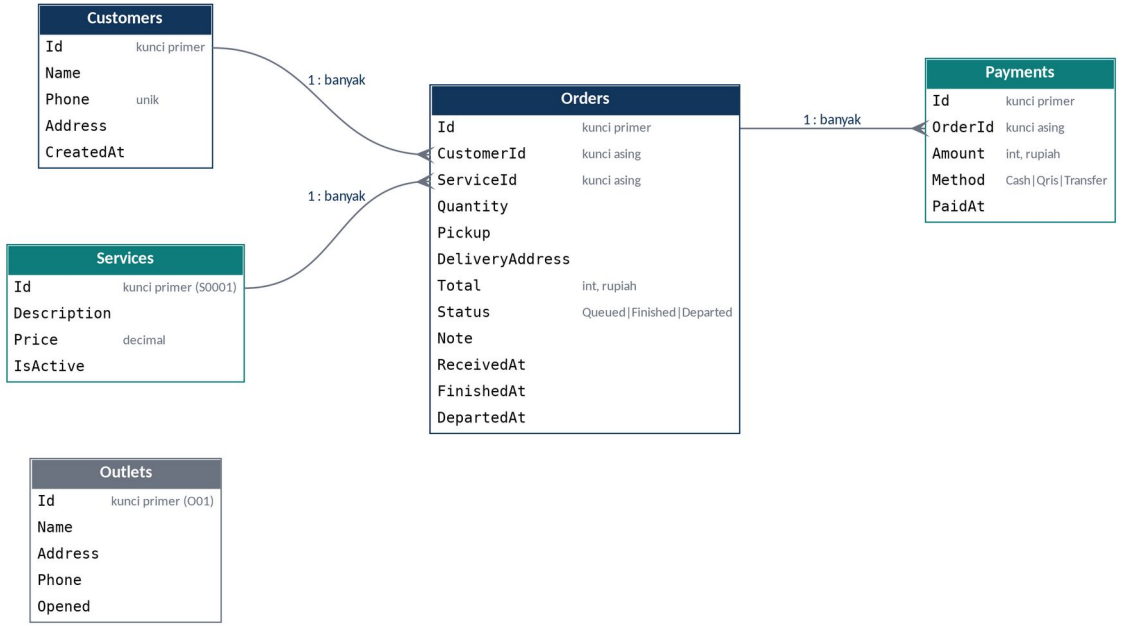
Seluruh basis data adalah satu berkas, `laundry.db`, di samping program: tidak ada yang perlu dipasang atau dinyalakan, mudah disalin dan mudah dibuang.

SQLite adalah basis data SQL sungguhan dengan transaksi, *foreign key*, dan indeks, dipakai di ponsel, peramban, dan banyak sekali program desktop.

Pindah ke MySQL atau SQL Server kelak berarti mengganti paket *provider* dan *connection string*; model, LINQ, dan controller tetap.

Untuk satu gerai dengan tiga kasir, SQLite lebih dari cukup. Bila beberapa gerai berbagi satu basis data lewat jaringan, basis data server adalah pilihan yang lebih baik. Buku: Subbab 5.2.4.

Merancang Datanya



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.

Gambar 5.2: lima tabel, `Services`, `Outlets`, `Customers`, `Orders`, dan `Payments`. Seorang pelanggan punya banyak pesanan; sebuah pesanan milik satu pelanggan dan satu layanan, dan punya banyak pembayaran. Pelanggan membayar bertahap, tunai, dengan QRIS, atau transfer, sehingga setiap pembayaran menjadi baris sendiri. Buku: Subbab 5.3.

Sisa Tagihan Dihitung, Tidak Pernah Disimpan

$$\text{Sisa} = \text{Total} - \sum_{p \in \text{pembayaran}} \text{Jumlah}_p \tag{5.1}$$

Menyimpan kolom `Paid` *dan* daftar pembayaran sekaligus mengundang keduanya untuk suatu hari tidak sepakat.

Persamaan (5.1) dihitung oleh dua properti baca-saja di `Order`: `Paid => Payments.Sum(p => p.Amount)` dan `Balance => Total - Paid`.

Uang disimpan dalam rupiah utuh sebagai `int`; `Total` pesanan ditetapkan saat pesanan dibuat, sehingga perubahan harga kemudian tidak menulis ulang sejarah.

Contoh dari buku: pesanan 2 seharga Rp 36.000 dan sudah dibayar Rp 20.000, jadi masih kurang Rp 16.000. Setelah dua pembayaran lagi sebesar Rp 10.000 dan Rp 6.000, `Paid` menjadi 36000 dan sisanya 0, dan formulir pembayaran hilang dari halaman pesanan. Buku: Subbab 5.3 dan Soal Latihan 5.1.

LINQ Dulu, Tanpa Basis Data

```
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
```

Keluaran sungguhan `LinqDemo`, yang menjalankan kueri-kueri bab ini pada daftar di memori dengan kelas model milik aplikasi web. `Where` menyimpan apa yang diterima sebuah lambda, `OrderBy` mengurutkan, `GroupBy` membuat satu kelompok per layanan, `Select` membentuk setiap kelompok menjadi ringkasan. Kueri yang sama pada `DbSet` menjadi SQL. Pelajari LINQ di sini, tempat tidak ada yang bisa salah dengan basis data. Buku: Subbab 5.4.

Memasang Entity Framework Core

```
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
```

Provider SQLite bekerja saat berjalan; paket Design dan perkakas global `dotnet-ef` membuat migrasi saat desain.

`appsettings.json` mendapat *connection string* `"Laundry": "Data Source=laundry.db"` dan kategori log `Microsoft.EntityFrameworkCore.Database.Command` pada tingkat `Information`, sehingga setiap perintah SQL tampil di terminal.

Paket dan perkakasnya memakai versi yang sama, 10.0.12, rilis EF Core 10 yang berlaku. Melihat SQL-nya adalah cara terbaik untuk mempelajari apa yang dikerjakan EF Core. Buku: Subbab 5.5.

Kelas-Kelas Model

Kode 5.1 Sebagian `Models/Order.cs`: pesanan dengan *foreign key*, status, dan pembayarannya; `Paid` dan `Balance` mewujudkan rumus sisa tagihan.

```
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 int Total { get; set; }               // rupiah, fixed at creation
    public OrderStatus Status { get; set; } = OrderStatus.Queued;

    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 5.1: kelas C# biasa. Menurut **konvensi**, `Id` adalah kunci, `CustomerId` adalah foreign key ke `Customer`, dan `Payments` adalah relasi satu-ke-banyak. Kelas lengkapnya menambah antar-jemput, alamat pengantaran, catatan, dan tiga cap waktu. `Service` berubah dari record Bab 3 menjadi kelas entitas dengan setter dan penanda `IsActive`. Buku: Subbab 5.6.*

DbContext

`LaundryDbContext` punya lima properti `DbSet`, satu per tabel: `Services`, `Outlets`, `Customers`, `Orders`, `Payments`.

`OnModelCreating` menambahkan apa yang tidak bisa ditebak konvensi: panjang maksimum, indeks unik pada `Customer.Phone`, indeks pada `Orders.Status`, *enum* disimpan sebagai teks dengan `HasConversion<string>()`.

Menghapus pelanggan atau layanan tidak boleh menghapus pesanan: kedua relasi memakai `OnDelete(DeleteBehavior.Restrict)`.

Bawaan EF Core untuk relasi wajib adalah cascade delete. Putuskan dengan sengaja, relasi demi relasi, apakah menghapus induk boleh menghapus anak-anaknya; di sebuah toko, pesanan adalah sejarah dan harus tetap ada. Buku: Subbab 5.7.

Data Awal, dan Menepati Janji Bab 3

`UseSeeding` (EF Core 9 ke atas) berjalan setiap kali basis data dibuat atau dimigrasi. `SeedData` memuat layanan dan gerai dari berkas CSV Bab 3, lalu tiga pelanggan contoh beserta pesanan dan pembayarannya; setiap langkah lebih dulu memeriksa apakah tabelnya kosong.

`EfCatalog` menjawab antarmuka lama `ILaundryCatalog` dari basis data.

`Program.cs` berubah di dua baris: `AddDbContext<LaundryDbContext>(…)` dengan SQLite dan *callback* data awal, serta `AddScoped<ILaundryCatalog, EfCatalog>()` sebagai ganti `CsvCatalog` yang *singleton*.

Tidak satu pun controller Bab 3 berubah. Katalog menjadi scoped karena kini memakai DbContext, yang scoped: satu per permintaan. Singleton yang memegang DbContext akan rusak begitu dua permintaan datang bersamaan. Buku: Subbab 5.8.

Migrasi: Dari Kelas ke Tabel

```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

Perintah pertama menulis `<timestamp>_InitialCreate.cs` dengan metode `Up` (membuat lima tabel, kunci, *foreign key*, dan indeks) dan metode `Down` yang membatalkannya, ditambah *snapshot* model.

Perintah kedua menerapkannya dan mencatatnya di tabel `__EFMigrationsHistory`.

Tambahkan `Rack` ke `Order`, jalankan `dotnet ef migrations add AddRackToOrders`, dan EF Core membandingkan model dengan *snapshot*: satu perbedaan, sehingga `Up` menambah satu kolom dan `Down` menghapusnya.

> **WAWASAN KUNCI:** Migrasi bagi basis data adalah seperti *commit* Git bagi kode: kecil, berurutan, bisa ditinjau, dan disimpan di repositori, sehingga setiap komputer bisa dibawa ke skema yang persis sama.

`dotnet ef migrations remove` menghapus migrasi terakhir bila belum diterapkan; `dotnet ef migrations script` mencetak SQL-nya untuk administrator basis data. Buku: Subbab 5.9.

CRUD Sungguhan untuk Pelanggan

Tabel 5.1 Aksi-aksi utama `CustomersController`, alamatnya, dan SQL yang ditimbulkannya.

Aksi	Alamat dan metode	Yang dikirim EF Core
`Index(search)`	`GET /Customers?search=Sari`	`SELECT … WHERE instr(Name, @search) > 0 …`
`Details(id)`	`GET /Customers/Details/3`	satu `SELECT` dengan `LEFT JOIN` ke `Orders`
`Create(customer)`	`POST /Customers/Create`	pemeriksaan telepon, lalu `INSERT`
`Edit(id, input)`	`POST /Customers/Edit/3`	`SELECT`, pemeriksaan telepon, `UPDATE`
`DeleteConfirmed(id)`	`POST /Customers/Delete/3`	`SELECT`, pemeriksaan pesanan, `DELETE`

Tabel 5.1: setiap aksi yang menulis adalah `POST`, dan setiap `GET` hanya membaca. Controller menerima `LaundryDbContext db` lewat konstruktor primernya, satu per permintaan; ia tidak pernah membuat, membuka, atau membuang koneksi. Nilai selalu berjalan sebagai parameter seperti `@search`, tidak pernah ditempelkan ke dalam SQL. Buku: Subbab 5.10 dan 5.10.1.

Create: Mengikat, Memeriksa, Menyimpan

Kode 5.2 Aksi `Create` untuk `POST` di `Controllers/CustomersController.cs`.

```
[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);

    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 });
}
```

*Kode 5.2: `[Bind]` mendaftar persis isian yang boleh diisi formulir; `[ValidateAntiForgeryToken]` menolak formulir yang bukan buatan situs kita sendiri. Pemeriksaan telepon menambah galat ke `ModelState`; bila ada yang salah, formulir kembali beserta nilainya. Setelah `SaveChangesAsync`, basis data sudah memilih `Id`, dan pola **Post/Redirect/Get** mengirim peramban ke halaman rincian, sehingga menekan refresh tidak memasukkan data dua kali. Buku: Subbab 5.10.3.*

Read, Update, Delete

Read: halaman daftar memakai `AsNoTracking()`; halaman rincian memakai `Include(c => c.Orders).ThenInclude(o => o.Service)` agar baris terkait dimuat dalam satu kueri.

Update: muat entitas yang dilacak dengan `FindAsync`, salin nilai baru ke atasnya, `SaveChangesAsync`: EF Core mengirim `UPDATE` hanya untuk kolom yang berubah.

Delete: sebuah `GET` menampilkan halaman konfirmasi; `POST` memeriksa pesanan lebih dulu dan menolak dengan sopan, dan `Restrict` menjaga di tingkat basis data.

Pencarian di SQLite peka huruf besar-kecil (`instr`), sehingga pencarian `sari` tidak menemukan Sari Wulandari; Bab 8 memperbaikinya di Soal 8.4. Pelanggan bernama `'; DROP TABLE Customers; --` hanyalah nama yang aneh, karena nilainya berjalan sebagai parameter. Buku: Subbab 5.10.2, 5.10.4, dan 5.10.5.

Pesanan, Pembayaran, Harga, dan Layar Kasir

OrdersController: pesanan baru dihargai oleh `PriceCalculator` Bab 3 (2 kg cuci, kering, dan lipat ditagih 3 kg, Rp 24.000); alur Queued → Finished → Departed dengan cap waktu; pembayaran dicatat di halaman pesanan.

PricesController: pemilik mengubah harga dan menambah layanan dengan kode otomatis; pesanan lama tetap dengan totalnya.

CounterController: empat angka untuk meja kasir (dalam antrean, siap diambil, diterima hari ini, uang diterima hari ini), masing-masing satu perintah SQL.

Aturan harga yang ditulis sekali di Bab 3 kini menghargai pesanan sungguhan, persis pemakaian ulang yang dijanjikan aturan controller ramping. Buku: Subbab 5.11.

Menjalankannya di Komputer Anda

Jalankan dua perintah `dotnet ef`, lalu `dotnet run`, dan buka `http://localhost:4050`. Beranda, layanan, dan gerai tampil seperti di Bab 3, kini dari basis data.

Coba: tambahkan pelanggan dua kali dengan satu nomor telepon; buat pesanan 2 kg; catat pembayaran Rp 10.000; tandai selesai lalu diambil; ubah sebuah harga; coba hapus pelanggan yang punya pesanan.

Perhatikan terminal: sebelum baris waktu setiap permintaan, tampil SQL yang dikirim EF Core, dengan parameter yang ditampilkan menurut nama seperti `@p0`.

Tujuh langkah ini adalah tur terpandu dari buku, dan latihan yang baik untuk dikerjakan berpasangan: satu orang mengeklik, yang lain membacakan SQL-nya keras-keras. Buku: Subbab 5.12.

Menjadikannya Lebih Baik

Baca tanpa pelacakan, tulis dengan pelacakan. Tetap asinkron: `ToListAsync`, `AnyAsync`, `SaveChangesAsync`.

Beri indeks pada yang dicari, ketika kueri nyata terasa lambat. **Perhatikan jumlah kueri** dan apa yang dimuat setiap `Include`.

Waspadai pemisah desimal: di komputer berbahasa Indonesia, `4.5` dari isian angka bisa terbaca sebagai 45.

Poin terakhir adalah jebakan nyata di komputer kasir yang diatur ke bahasa Indonesia, tempat koma menjadi pemisah desimal: biarkan culture server pada bawaannya, atau atur `CultureInfo.DefaultThreadCurrentCulture` secara eksplisit, dan selalu uji dengan nilai yang berdesimal. Buku: Subbab 5.13.

Kesalahan yang Sering Terjadi

Lupa `SaveChangesAsync()`: tidak ada yang sampai ke basis data. **Mengubah model tanpa migrasi**: sejak EF Core 9, `Migrate()` berhenti dengan pesan *pending changes*.

Properti navigasi yang tidak dimuat: tanpa `Include`, `order.Payments` kosong dan sisa tagihan diam-diam salah.

DbContext *singleton*; ***cascade delete* tanpa sengaja**; **menghapus lewat tautan** (sebuah `GET` tidak boleh mengubah data).

Menjumlahkan `decimal` **di SQLite**: EF Core tidak bisa menerjemahkannya dan melempar pengecualian saat berjalan; simpan uang yang dijumlahkan di kolom `int`.

Kesalahan properti navigasi paling berbahaya, karena tidak ada yang gagal: halaman menampilkan sisa tagihan, dan sisa itu salah. Bila sebuah angka tampak mencurigakan, baca SQL di terminal dan pastikan `JOIN`*-nya ada. Buku: Subbab 5.16.*

Ringkasan: Lima Hal untuk Diingat

1. EF Core memetakan kelas ke tabel dan objek ke baris (Gambar 5.1); konvensi ditambah `OnModelCreating` menggambarkan modelnya.
2. `DbContext` didaftarkan dengan `AddDbContext` dan diinjeksikan, satu per permintaan.
3. Migrasi adalah berkas C# yang berurutan dan bisa dibatalkan; `database update` menerapkannya; `UseSeeding` mengisi basis data yang kosong.
4. LINQ pada `DbSet` menjadi SQL berparameter ketika metode seperti `ToListAsync` meminta hasil; `Include` memuat baris terkait; `AsNoTracking` untuk membaca.
5. Create menambah dan menyimpan, update mengubah entitas yang dilacak, delete dikonfirmasi, diperiksa, dan dijaga; sisa tagihan selalu dihitung (Persamaan (5.1)).

Buku: Subbab 5.17.

Pertemuan Berikutnya

Bab 6 — ASP.NET Core MVC III. Validasi, Identity, dan *deployment*: aturan pada model, login dan peran untuk kasir dan pemilik, serta situs di server sungguhan.

Sebelum itu: jalankan `LinqDemo`, tambahkan kueri “tiga pelanggan terbaik” dari Soal Latihan 5.3, dan di komputer Anda sendiri jalankan aplikasi web lalu ikuti tur terpandunya.

Buku: Subbab 5.18 dan 5.19.

Kuliah 6 — ASP.NET Core MVC III: Validasi, Identity, dan Penerapan

Mata kuliah: Pemrograman Berbasis Kerangka Kerja

Buku: Irfan Subakti, *Pemrograman Berbasis Kerangka Kerja: ASP.NET Core, Laravel, dan Spring Boot*, Bab 6

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini aplikasi laundry mendapat aturan dan kunci, lalu meninggalkan laptop pemilik. Kita menambahkan validasi dengan anotasi data dan kelas formulir, login dengan ASP.NET Core Identity, dua peran dengan fallback policy yang melindungi setiap halaman secara bawaan, dan penerapan ke server Linux di belakang Nginx. Buku: Bab 6, Subbab 6.1–6.14.

Cakupan Pertemuan Ini

1. **Pintu yang terbuka** — tiga insiden kecil di minggu pertama.
2. **Validasi** — atribut pada model, aturan buatan sendiri, aturan tentang dua isian, kelas formulir.
3. **Identity** — pengguna dan peran di basis data toko, cara kata sandi disimpan, login.
4. **Otorisasi** — siapa boleh melakukan apa, *fallback policy*, *policy* bernama.
5. **Penerapan** — lingkungan, `dotnet publish`, migrasi di server, Nginx dan systemd, daftar periksa rilis.

Yang dijalankan di sini: program konsol `ValidationDemo`, yang memakai aturan milik aplikasi dan password hasher Identity dari kerangka kerja bersama, dibangun, dipublikasikan, dan dijalankan, dan keluarannya hari ini adalah keluaran sungguhan. Aplikasi webnya membutuhkan paket NuGet dan ditulis berdasarkan dokumentasi resmi; Anda menjalankannya di komputer sendiri. Buku: Bab 6 dan README-nya.

Posisi Kita Sekarang: Pintu yang Terbuka

Seorang kasir paruh waktu yang baru membuka *Prices*, mengetik di isian yang salah, dan menyimpan *Ironing only* seharga Rp 60 per kg; tidak ada yang sadar sampai malam.

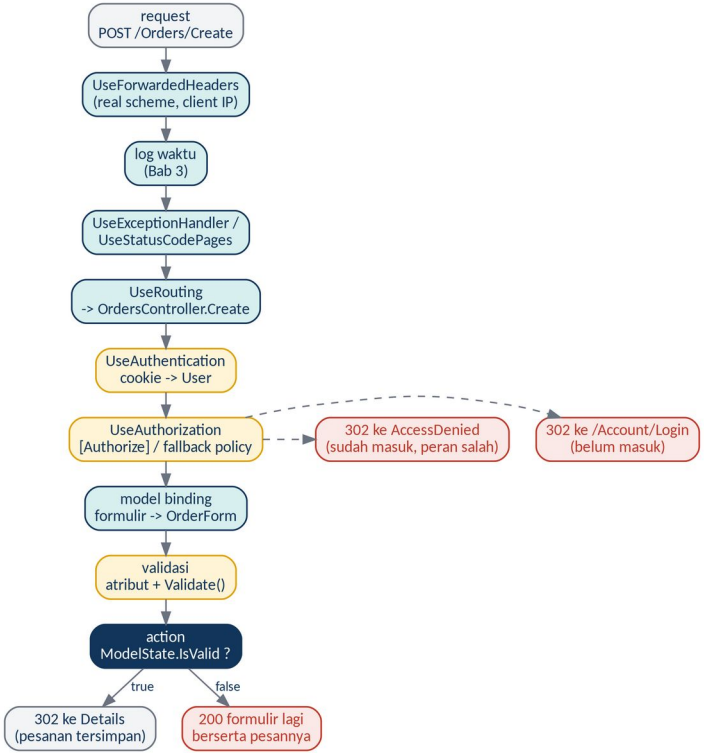
Nomor telepon tersimpan sebagai `0812 0000 1O4`, dengan huruf O, sehingga pesan WhatsApp “cucian Anda sudah siap” tidak pernah sampai.

Pesanan antar-jemput tersimpan tanpa alamat pengantaran.

Dan aplikasinya masih berjalan di laptop pemilik, dinyalakan dengan tangan, dan berhenti setiap kali laptop tertidur.

Tidak satu pun dari ini adalah galat pemrograman: kodenya mengerjakan persis apa yang diperintahkan. Yang kurang adalah seperangkat aturan yang harus dilewati setiap permintaan, dan rumah yang layak bagi aplikasi. Aturan itu membantu kasir, pelanggan, sopir pengantar, dan pemilik. Buku: Subbab 6.1.

Gagasannya: Jalur Pengiriman Formulir



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.

*Gambar 6.1: **autentikasi** memastikan siapa penggunanya (dari cookie); **otorisasi** memutuskan apa yang boleh dilakukannya. Baru setelah itu model binding mengisi kelas formulir dan validasi mengisi `ModelState`. Permintaan berakhir dengan salah satu dari empat cara: diarahkan ke halaman login, ditolak, formulir ditampilkan lagi dengan pesan, atau disimpan lalu dialihkan. Buku: Subbab 6.2.*

Validasi dengan Anotasi Data

Kode 6.1 Properti-properti pertama `Models/Customer.cs` dengan atribut validasi: nama wajib sepanjang 2 sampai 80 karakter dan nomor ponsel Indonesia yang wajib.

```
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; } = "";
```

Kode 6.1: aturan dinyatakan sekali, pada model, dengan kata-kata yang dimengerti kasir. `[Range]` menjaga harga layanan di antara Rp 1.000 dan Rp 1.000.000; `[RegularExpression]` mewajibkan deskripsi berakhiran "(per kg)", "(per item)", atau "(per pair)". `[Display]` memberi isian nama yang tampil di pesan. Buku: Subbab 6.3.1.

Atribut Validasi Buatan Sendiri

`IndonesianMobileAttribute` diturunkan dari `ValidationAttribute` dan menimpa `IsValid`: setelah spasi dan tanda hubung dibuang, nomornya harus nomor ponsel seperti `0812…` atau `+62 812…`.

Sebuah fungsi pembantu menyimpan setiap nomor dalam satu ejaan: `+62 812-0000-0104` menjadi `081200000104`, sehingga pencarian dan pengiriman pesan berjalan.

Huruf O pada `0812 0000 104` gagal, dan kasir langsung melihat pesannya, alih-alih pelanggan tidak pernah menerima pesannya.

Soal Latihan 6.2 menulis satu atribut lagi, `NotInFutureAttribute`, untuk tanggal: lima baris, dan setiap properti tanggal di aplikasi bisa memakainya. Buku: Subbab 6.3.2 dan Soal Latihan 6.2.

Menjalankan Aturan Tanpa Server Web

```
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.
5. Order: pickup without an address: INVALID
   [DeliveryAddress] Pickup and delivery needs a delivery address.
```

Sebagian keluaran sungguhan `ValidationDemo`, yang memanggil `Validator.TryValidateObject` pada kelas-kelas milik aplikasi. Kasus 3 persis insiden daftar harga di minggu pertama, kini ditolak dengan dua pesan yang jelas. Aturan yang bisa dijalankan di konsol juga bisa diuji tanpa peramban. Buku: Subbab 6.3.3.

Apa yang Dikerjakan MVC dengan Aturan Itu

Model binding memvalidasi setiap objek yang diikat dan mencatat setiap kegagalan di `ModelState`, lengkap dengan nama properti dan pesannya.

Action yang memutuskan: `if (!ModelState.IsValid) return View(form);` menampilkan formulir lagi, dengan nilai dan pesan dari `asp-validation-for` serta ringkasannya.

> **PERINGATAN:** Validasi tidak menghentikan *action* dengan sendirinya. *Action* yang menyimpan tanpa bertanya `ModelState.IsValid` menyimpan data yang tidak valid.

Controller menambahkan aturan yang butuh basis data, seperti "nomor telepon ini sudah dimiliki pelanggan lain", dengan `ModelState.AddModelError`, sehingga muncul di tempat yang sama. Pola ini akan kita temui lagi di Laravel dan Spring Boot dengan nama berbeda: aturan dinyatakan di satu tempat, kegagalan dikumpulkan, dan formulir dikembalikan beserta pesannya. Buku: Subbab 6.3.4.

Kelas Formulir, Bukan Entitas

Entitas yang diikat langsung ke formulir membiarkan permintaan rekaan mengisi properti *apa pun*, seperti `Total` atau `Status`: inilah yang disebut ***overposting***.

`OrderForm` memuat persis apa yang boleh dikirim formulir pesanan baru (pelanggan, layanan, jumlah, antar-jemput, alamat pengantaran, catatan) beserta aturan formulir itu.

“Antar-jemput membutuhkan alamat pengantaran” melibatkan dua isian, sehingga `OrderForm` mengimplementasikan **`IValidatableObject`**, yang metode `Validate`-nya bisa `yield return` aturan apa pun.

Action menyalin dari formulir ke entitas baru dan menghitung total dengan `PriceCalculator`, sehingga harga tidak pernah bisa datang dari peramban. `PaymentForm` melakukan hal yang sama untuk pembayaran: antara Rp 1.000 dan Rp 10.000.000, metode tunai, QRIS, atau transfer. Buku: Subbab 6.4.

Pemeriksaan di Peramban Juga

Atribut yang sama menghasilkan atribut `data-val-*` di HTML; *jQuery Validation* dan *jQuery Validation Unobtrusive* memeriksanya sebelum formulir dikirim.

Pemeriksaan di peramban hanya untuk kenyamanan: kasir melihat pesan tanpa bolak-balik ke server.

Aturan yang hanya diperiksa JavaScript bukanlah aturan: siapa pun bisa mematikan JavaScript atau mengirim permintaan dengan `curl`. Server selalu memeriksa lagi.

Aturan tentang dua isian dan aturan yang butuh basis data hanya berjalan di server, satu alasan lagi mengapa pemeriksaan di server adalah pemeriksaan yang sesungguhnya. Buku: Subbab 6.5.

Identity: Siapa yang Ada di Meja Kasir?

ASP.NET Core Identity menyimpan pengguna, *hash* kata sandi, peran, dan data penguncian akun; `LaundryDbContext` diturunkan dari `IdentityDbContext`, sehingga pengguna tinggal di basis data yang sama dengan pesanan.

`Program.cs` menambahkan Identity dengan aturan kata sandi dan penguncian, serta ***fallback authorization policy***: setiap halaman mewajibkan pengguna yang sudah login kecuali halaman itu menyatakan `[AllowAnonymous]`.

Dua peran, `Owner` (harga, menghapus pelanggan) dan `Cashier` (pesanan dan pembayaran); `IdentitySeeder` menerapkan migrasi serta membuat peran dan akun pertama saat aplikasi menyala.

Akun demo hanya ada di `appsettings.Development.json` di komputer pengembang, misalnya pemilik `owner@profitina-laundry.example`; di server, pemilik pertama dibuat sekali dari variabel lingkungan yang diatur dengan `systemctl edit`. Identity juga membawa cookie login, penguncian setelah kegagalan berulang, dan security stamp, sehingga semua itu tidak ditulis dengan tangan. Haruskah Anda menulis penyimpanan kata sandi sendiri? Ini salah satu dari sedikit pertanyaan di mata kuliah ini yang jawabannya sederhana: tidak. Buku: Subbab 6.6.1, 6.6.2, dan 6.6.4.

Yang Disimpan sebagai Ganti Kata Sandi

```
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
```

*Keluaran sungguhan kasus terakhir `ValidationDemo`. Basis data tidak pernah menyimpan kata sandi, hanya **hash bergaram** (salted hash): PBKDF2 dengan HMAC-SHA512, 100.000 iterasi, garam acak 16 byte, dan kunci 32 byte. Kata sandi yang sama di-hash dua kali menghasilkan teks berbeda karena garamnya berbeda, sehingga dua pengguna dengan kata sandi yang sama tidak bisa dikenali. Pemeriksaan menghitung ulang hash dengan garam yang tersimpan. Buku: Subbab 6.6.3.*

Login dan Menjaga Pintu Tetap Tertutup

Tabel 6.1 Siapa boleh memakai halaman mana setelah bab ini.

Halaman	Belum login	Kasir	Pemilik
Beranda, layanan, penawaran, gerai, halaman galat	ya	ya	ya
Layar kasir, pesanan, pembayaran	halaman login	ya	ya
Pelanggan: daftar, tambah, ubah	halaman login	ya	ya
Pelanggan: hapus	halaman login	ditolak	ya
Harga	halaman login	ditolak	ya

Tabel 6.1: `AccountController` *melakukan login dengan* `SignInManager`*, yang juga menghitung kegagalan dan mengunci akun sementara setelah terlalu banyak gagal.* `[Authorize(Roles = AppRoles.Owner)]` *membatasi harga dan penghapusan; kasir yang mengetik alamatnya melihat halaman "Not allowed" yang sopan.* ***Policy*** *memberi nama pada sebuah aturan, sehingga controller menyatakan apa yang dibutuhkannya, bukan bagaimana memeriksanya. Buku: Subbab 6.7.*

Menyembunyikan Tautan Bukanlah Melindungi Halaman

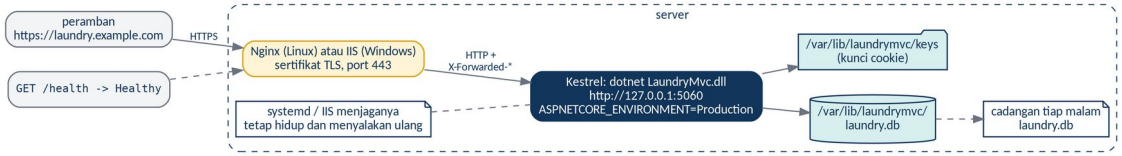
Layout hanya menampilkan tautan *Prices* kepada pemilik. Itu merapikan menu; tidak melindungi apa pun.

Siapa pun bisa mengetik `/Prices`. Atribut pada controller-lah yang menolak permintaannya.

Urutan di *pipeline*: `UseRouting()`, lalu `UseAuthentication()`, lalu `UseAuthorization()`. Dalam urutan lain, aturannya tidak melihat *endpoint* atau tidak melihat pengguna.

Fallback policy adalah bawaan yang aman: controller baru yang ditulis tahun depan tertutup sampai seseorang membukanya dengan sengaja, bukan terbuka sampai seseorang ingat menutupnya. Buku: Subbab 6.6.2 dan 6.7.2.

Penerapan: Dari Laptop ke Server



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.

*Gambar 6.2: **lingkungan** menentukan pengaturan: `appsettings.json`, lalu `appsettings.Production.json`, lalu variabel lingkungan, setiap lapisan menimpa lapisan sebelumnya. Di Production, basis data dan kunci cookie tinggal di `/var/lib/laundrymvc`. `dotnet publish -c Release -o <folder>` menghasilkan berkas-berkas yang disalin. Buku: Subbab 6.8.1 dan 6.8.2.*

systemd Menjaganya Tetap Berjalan

Kode 6.2 Bagian *service* dari `deploy/laundrymvc.service`: unit systemd yang menjaga aplikasi tetap berjalan.

```
[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
```

*Kode 6.2: systemd menyalakan aplikasi saat server boot, menyalakannya lagi lima detik setelah mogok, dan menjalankannya sebagai pengguna tanpa hak istimewa `laundry`. Kestrel hanya mendengarkan alamat loopback; **Nginx** di depannya menerima lalu lintas publik, memegang sertifikat TLS dari certbot, dan meneruskan dengan `X-Forwarded-For` serta `X-Forwarded-Proto`. Di Windows, IIS dengan ASP.NET Core Module memainkan peran Nginx. Buku: Subbab 6.8.4 dan 6.8.5.*

Hal-Hal Kecil yang Membuatnya Awet

Migrasi di server sebelum melayani: aplikasi menerapkannya saat menyala, atau `dotnet ef migrations script` memberi SQL untuk ditinjau.

Forwarded headers, atau pengalihan menunjuk ke `http://` dan log menampilkan setiap pengunjung sebagai `127.0.0.1`. **Kunci Data Protection yang disimpan**, atau setiap *restart* membuat semua orang keluar.

***Endpoint* kesehatan** `/health` untuk pemantauan; **cadangan setiap malam** untuk berkas basis data; **daftar periksa rilis** yang diikuti setiap kali.

Daftar periksa di buku dimulai dengan "salin berkas basis data" karena suatu alasan: migrasi yang menghapus kolom tidak bisa dibatalkan `Down` *dengan cara apa pun yang mengembalikan datanya. Buku: Subbab 6.8.3, 6.8.6, dan 6.8.7.*

Memakainya Sungguhan

Jalankan di komputer Anda: login sebagai pemilik demo dan kasir demo, coba halaman harga sebagai masing-masing, dan coba tiga insiden minggu pertama.

Untuk toko, klinik, atau kebun Anda sendiri: tuliskan aturan yang disampaikan kepada karyawan baru di hari pertamanya; masing-masing menjadi atribut, aturan formulir, atau peran.

Usaha kecil jarang membutuhkan lebih dari dua atau tiga peran. Mulailah dengan jumlah paling sedikit yang memisahkan “boleh mengubah aturan uang” dari “boleh melayani pelanggan”. Buku: Subbab 6.9–6.12.

Kesalahan yang Sering Terjadi

Memercayai peramban; lupa `if (!ModelState.IsValid)`; mengikat formulir ke entitas.

***Middleware* dalam urutan yang salah; melindungi halaman satu per satu** alih-alih *fallback policy*; **menyembunyikan tautan alih-alih melindungi halaman.**

Kata sandi demo di produksi; kehilangan kunci *cookie*; tanpa *forwarded headers* di belakang *proxy*; memutakhirkan tanpa cadangan.

Sebagian besar kesalahan ini tidak terlihat di laptop pengembang: satu pengguna, satu komputer, tanpa proxy, tanpa restart. Semuanya muncul pada hari sibuk pertama di server, itulah sebabnya daftar periksa rilis dan gladi di server uji layak diberi satu sore. Buku: Subbab 6.13.

Ringkasan: Lima Hal untuk Diingat

1. Validasi berjalan di server untuk setiap permintaan: atribut untuk nilai tunggal, `IValidatableObject` untuk beberapa nilai, controller untuk aturan yang butuh basis data; semuanya berakhir di `ModelState`.
2. Kelas formulir memuat persis apa yang boleh dikirim formulir, sehingga *overposting* hilang dengan sendirinya.
3. Autentikasi menjawab siapa; otorisasi menjawab apa (Gambar 6.1). Identity menyimpan *hash* PBKDF2-HMAC-SHA512 bergaram dan mengunci akun setelah kegagalan berulang.
4. *Fallback policy* melindungi setiap halaman secara bawaan; `[AllowAnonymous]` membuka halaman dengan sengaja; peran membatasinya.
5. Penerapan: konfigurasi berlapis, `dotnet publish`, migrasi lebih dulu, *reverse proxy* dengan *forwarded headers*, kunci yang disimpan, pemeriksaan kesehatan, dan cadangan (Gambar 6.2).

Perhatikan bahwa ketiga insiden minggu pertama kini masing-masing punya penjaga: harga Rp 60 ditolak oleh `[Range]`, huruf O ditolak oleh `IndonesianMobile`, alamat yang kosong ditolak oleh `IValidatableObject`, dan kasir baru bahkan tidak lagi bisa membuka halaman harga. Setiap penjaga ditulis sekali, di tempat yang jelas, dan berlaku untuk setiap permintaan. Buku: Subbab 6.14.

Pertemuan Berikutnya

Bab 7 — Laravel I. Kerangka kerja kedua: Artisan, perutean, Blade, dan Eloquent, dengan *laundry* yang sama dalam PHP 8.5.

Sebelum itu: jalankan `ValidationDemo`, tambahkan `NotInFutureAttribute` dari Soal Latihan 6.2, dan di komputer Anda login ke aplikasi web dengan kedua peran.

Semua yang kita kerjakan di ASP.NET Core selama empat bab punya padanan di Laravel; simpan tabel terjemahan Bab 2 di dekat Anda. Buku: Subbab 6.15 dan 6.16.

Kuliah 7 — Laravel I: Artisan, Perutean, Blade, dan Eloquent

Mata kuliah: Pemrograman Berbasis Kerangka Kerja

Buku: Irfan Subakti, *Pemrograman Berbasis Kerangka Kerja: ASP.NET Core, Laravel, dan Spring Boot*, Bab 7

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini kita berkenalan dengan kerangka kerja kedua. Kita membangun ulang halaman staf Profitina Laundry di Laravel 13 dengan PHP 8.5: Artisan, migrasi yang ditulis dalam PHP, model Eloquent, rute dalam satu berkas, controller dengan validasi, dan templat Blade. Rancangan, halaman, dan aturannya sama dengan Bab 3 dan 5; yang berubah hanyalah tempat tinggal setiap bagian dan namanya. Buku: Bab 7, Subbab 7.1–7.16.

Cakupan Pertemuan Ini

1. **Toko yang sama, bahasa lain** — apa yang terbawa dari ASP.NET Core, dan apa yang tidak.
2. **Siklus hidup permintaan dan proyeknya** — *front controller*, *middleware*, *router*; di mana segala sesuatu tinggal.
3. **Artisan dan migrasi** — baris perintah yang menulis kode; tabel yang ditulis dalam PHP.
4. **Eloquent** — *enum*, model, relasi, *cast*; Eloquent dan EF Core berdampingan.
5. **Rute, controller, Blade** — `Route::resource`, validasi, *escaping*, formulir dengan token CSRF.

Catatan jujur: Packagist tidak bisa dijangkau di tempat buku ini ditulis, sehingga tidak ada proyek Laravel yang bisa dibuat atau dijalankan di sana. Setiap berkas PHP diperiksa dengan `php -l`, berkas Blade diperiksa keseimbangan direktifnya, dan `tools/enum_demo.php` dijalankan dengan PHP biasa; keluarannya hari ini adalah keluaran sungguhan. Anda menjalankan aplikasinya di komputer sendiri. Buku: Bab 7 dan Subbab 7.11.

Toko yang Sama, Bahasa Lain

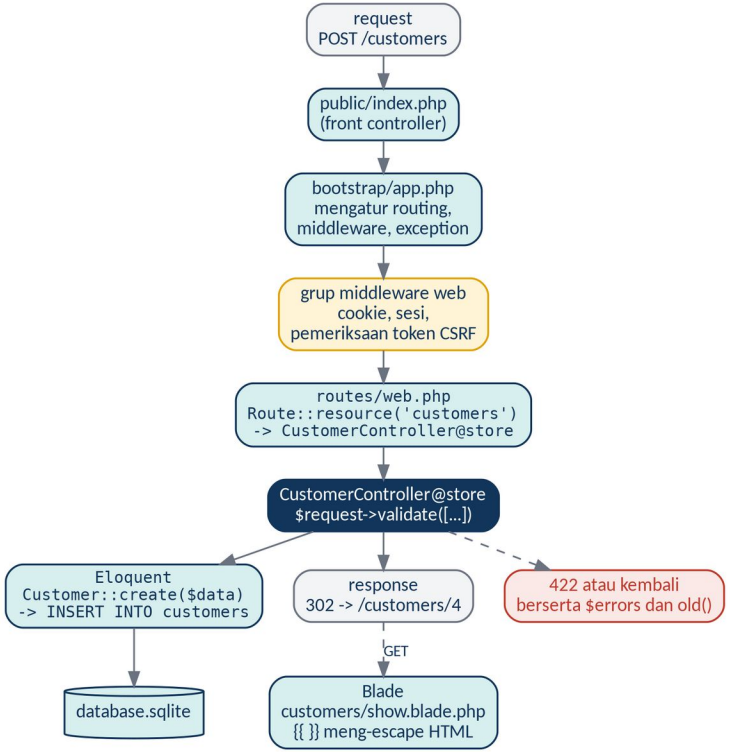
Keponakan pemilik, **Raka**, menguasai PHP. Gerai Wiyung mungkin kelak menjalankan salinan halaman staf sendiri di *hosting* PHP murah yang sudah dibayarnya.

Pertanyaannya: *berapa banyak dari yang sudah dibangun di C# yang harus saya ciptakan ulang?* Sedikit sekali pemikirannya, dan seluruh ketikannya.

Lima tabel yang sama, halaman yang sama, alur `queued → finished → departed` yang sama, aturan harga yang sama: minimal 3 kg, barang dihitung utuh, antar-jemput Rp 10.000 di bawah Rp 100.000.

Mempelajari kerangka kerja kedua sebagian besar berarti mempelajari di mana segala sesuatu tinggal. Melihat fitur yang sama di dua kerangka kerja adalah cara tercepat memahami apa itu kerangka kerja, dan Bab 8 lalu menyelesaikan soal-soal yang sama di keduanya. Buku: Subbab 7.1.

Satu Permintaan Melalui Laravel



Gambar 7.1 Satu permintaan melalui aplikasi Laravel: *front controller*, grup *middleware* web, *router*, *controller*, Eloquent, dan Blade.

*Gambar 7.1: setiap permintaan masuk lewat satu **front controller**, `public/index.php`. **Grup middleware web** mengurus cookie, sesi, dan token CSRF. Router memilih metode controller; controller meminta data kepada **Eloquent** dan mengembalikan view **Blade**. Ini pipeline Bab 3 dengan nama-nama yang berbeda. Buku: Subbab 7.2.*

Memasang dan Membuat Proyek

```
composer create-project laravel/laravel profitina-laundry "^13.0"
cd profitina-laundry
php artisan serve --port=4070
```

Laravel 13 (Maret 2026) berjalan di PHP 8.3 sampai 8.5; **Composer** memasangnya beserta paket-paketnya dari Packagist.

Proyek baru memakai **SQLite** secara bawaan, di `database/database.sqlite`; pengaturan tinggal di `.env`.

Tempat tinggal: `routes/web.php` (rute), `app/Http/Controllers`, `app/Models`, `database/migrations`, `resources/views`, `public/` (satu-satunya folder yang boleh dijangkau web).

Tabel di buku memetakan setiap folder ke padanannya di ASP.NET Core: `routes/web.php` untuk rute di `Program.cs` dan atribut, `resources/views` untuk `Views`, `.env` dan `config/` untuk `appsettings*.json`, `public/` untuk `wwwroot/`, `bootstrap/app.php` untuk sebagian besar `Program.cs`. Buku: Subbab 7.3.

Artisan, Baris Perintah yang Menulis Kode

`php artisan make:model Customer -m` — model beserta migrasinya; `make:controller CustomerController --resource --model=Customer` — controller dengan tujuh metode CRUD; `make:enum` — *backed enum*.

`php artisan migrate`, `migrate:fresh --seed` — menjalankan migrasi, atau menghapus semuanya dan membangun ulang dengan data contoh.

`php artisan route:list --except-vendor` — daftar milik *router* sendiri; `php artisan tinker` — konsol di dalam aplikasi.

Artisan memainkan peran `dotnet new` dan `dotnet ef` sekaligus. `route:list` adalah kembaran Laravel untuk halaman `/_routes` di Bab 4. Buku: Subbab 7.4.

Migrasi: Tabel yang Ditulis dalam PHP

Kode 7.1 Sebagian `2026_09_01_000003_create_orders_table.php`: pesanan dengan *foreign key*, status, dan cap waktunya.

```
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->unsignedInteger('total');         // rupiah, fixed at creation
    $table->string('status', 12)->default('queued')->index();
    $table->timestamp('received_at');
    $table->timestamp('finished_at')->nullable();
    $table->timestamps();
});
```

Kode 7.1: di EF Core migrasi dibangkitkan dari kelas; di Laravel migrasi ditulis dengan tangan memakai schema builder, dan model mengikuti tabel. Migrasi berjalan menurut urutan nama berkasnya. `foreignId(...)->constrained()` *membuat foreign key;* `restrictOnDelete` *mempertahankan pesanan ketika seseorang mencoba menghapus pelanggan, sedangkan pembayaran memakai* `cascadeOnDelete` *dan ikut terhapus bersama pesanannya. Buku: Subbab 7.5.*

Enum dan Uang, Dijalankan dengan PHP Biasa

Kode 7.2 Inti `app/Enums/PaymentMethod.php`: tunai, QRIS, dan transfer.

```
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',
        };
    }
}
```

```
Queued    -> Finished
Finished  -> Departed
Departed  -> (end)
enum(App\Enums\OrderStatus::Finished)
NULL
QRIS is stored as "qris"
Rp 36.000 Rp 1.250.000
```

Kode 7.2: ***backed enum*** *menyimpan nilai string-nya di basis data.* `OrderStatus` *mengenal alur kerjanya (*`next()` *memberi status berikutnya,* `NULL` *di akhir), dan* `Money::rp` *memformat rupiah dengan* `number_format($amount, 0, ',', '.')`*, teks yang sama dengan* `Money.Rp` *di C#. Keluarannya adalah eksekusi sungguhan* `tools/enum_demo.php`*. Buku: Subbab 7.6.1.*

Model Eloquent

Kode 7.3 Inti `app/Models/Customer.php`: seorang pelanggan dan relasi ke pesanan-pesanannya.

```
#[Fillable(['name', 'phone', 'address'])]
class Customer extends Model
{
    public function orders(): HasMany
    {
        return $this->hasMany(Order::class);
    }
}
```

*Kode 7.3: Eloquent adalah **active record**: setiap model mengenal tabelnya (`customers`, menurut konvensi) dan objek menyimpan dirinya sendiri. Atribut `#[Fillable]` mendaftar kolom yang boleh diisi sebuah permintaan sekaligus. `Order` menambahkan `casts()` (jumlah sebagai desimal, antar-jemput sebagai boolean, status sebagai `OrderStatus`), relasi `belongsTo` dan `hasMany`, serta `paid()` dan `balance()`, rumus sisa tagihan Bab 5 dalam PHP. Buku: Subbab 7.6.2.*

Eloquent dan EF Core Berdampingan

Tabel 7.1 Operasi sehari-hari di EF Core dan di Eloquent.

Tugas	EF Core	Eloquent
menyaring dan mengurutkan	`Where(...).OrderBy(c => c.Name)`	`where(...)->orderBy('name')`
menjalankan kueri	`await query.ToListAsync()`	`$query->get()`
baris terkait	`.Include(o => o.Customer)`	`->with('customer')`
menambah	`Add(c); await SaveChangesAsync()`	`Customer::create($data)`
mengubah	ubah, lalu `SaveChangesAsync()`	`$customer->update($data)`
menghapus	`Remove(c); await SaveChangesAsync()`	`$customer->delete()`
ada atau tidak	`AnyAsync(o => o.CustomerId == id)`	`$customer->orders()->exists()`

Tabel 7.1: EF Core punya unit of work (pelacak perubahan, satu `SaveChanges` untuk banyak perubahan); Eloquent menyimpan setiap objek ketika Anda memintanya. Keduanya mengirim parameter, tidak pernah menempelkan nilai. Buku: Subbab 7.6.3.

Data Awal dari Berkas Bab 3

`DatabaseSeeder` membaca `Service.csv` dan `Outlet.csv` yang sama dengan Bab 3, lalu membuat pelanggan, pesanan, dan pembayaran contoh.

`php artisan migrate:fresh --seed` membangun ulang basis data beserta data contohnya dalam satu perintah, sempurna di laptop.

> **PERINGATAN:** `migrate:fresh` menghapus setiap tabel. Di server toko, pakai `php artisan migrate --force`, yang hanya menjalankan migrasi baru.

Data yang sama di kedua aplikasi berarti halaman yang sama menampilkan angka yang sama, dan Bab 8 memanfaatkannya untuk membandingkan kedua versi baris demi baris. Buku: Subbab 7.7.

Perutean: Satu Berkas untuk Semuanya

Kode 7.4 Sebagian `routes/web.php`: halaman staf dan satu aksi tambahan untuk pesanan.

```
// 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');
```

*Kode 7.4: setiap rute punya **nama**, dan view membangun URL dengan `route('customers.show', $customer)`, sama seperti tag helper bertanya kepada router di Razor. `Route::resource` menyatakan tujuh rute CRUD sekaligus; `only` dan `except` memangkasnya. Buku: Subbab 7.8.*

Tujuh Rute *Resource*

Tabel 7.2 Tujuh rute dari `Route::resource('customers', …)`.

Metode dan alamat	Metode controller	Nama rute
`GET /customers`	`index`	`customers.index`
`GET /customers/create`	`create`	`customers.create`
`POST /customers`	`store`	`customers.store`
`GET /customers/{customer}`	`show`	`customers.show`
`GET /customers/{customer}/edit`	`edit`	`customers.edit`
`PUT /customers/{customer}`	`update`	`customers.update`
`DELETE /customers/{customer}`	`destroy`	`customers.destroy`

Tabel 7.2: ***route model binding*** *mengubah* `{customer}` *menjadi* `Customer` *yang dimuat berdasarkan kuncinya, atau 404 bila tidak ada, sebelum metode controller berjalan. Formulir HTML hanya bisa mengirim* `GET` *dan* `POST`*, sehingga* `PUT` *dan* `DELETE` *dikirim sebagai isian tersembunyi yang ditulis* `@method('PUT')`*. Buku: Subbab 7.8.*

Controller: Validasi dan CRUD

Kode 7.5 Metode `store` dan aturan-aturan di `app/Http/Controllers/CustomerController.php`.

```
public function store(Request $request)
{
    $customer = Customer::create($request->validate($this->rules()));

    return redirect()->route('customers.show', $customer)
        ->with('message', "Customer {$customer->name} added.");
}

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'],
    ];
}
```

Kode 7.5: `$request->validate(...)` *hanya mengembalikan isian yang sudah divalidasi. Bila sebuah aturan gagal, metode itu tidak kembali sama sekali: ia* ***mengalihkan kembali*** *ke formulir beserta galat dan masukan lama, sehingga controller tidak butuh* `if`*.* `Rule::unique(...)->ignore($customer)` *adalah pemeriksaan "telepon sudah dipakai" dari Bab 5 dalam satu baris.* `with('message', …)` *menitipkan pesan untuk halaman berikutnya. Buku: Subbab 7.9.1.*

Pesanan, Pembayaran, Harga, Layar Kasir

`OrderController` menghargai pesanan baru dengan aturan Bab 3 dan memajukannya di alur kerja dengan `advance`; `PaymentController` mencatat pembayaran untuk sebuah pesanan.

`PriceController` adalah daftar harga milik pemilik; `CounterController` adalah controller *invokable* dengan empat angka untuk meja kasir.

Melihat SQL-nya: `DB::listen` di lingkungan pengembangan mencetak setiap kueri, seperti log perintah EF Core.

Aturan harganya adalah angka-angka yang sama dengan `PriceCalculator` Bab 3, sehingga sebuah pesanan berharga sama di kedua aplikasi. Buku: Subbab 7.9.2 dan 7.9.3.

Blade: *Escaping*, Formulir, dan Masukan Lama

Kode 7.6 Formulir di `resources/views/customers/edit.blade.php`, dengan token CSRF dan isian metodenya.

```
<form method="post" action="{{ route('customers.update', $customer) }}"
      class="form">
    @csrf
    @method('PUT')
    @include('customers._form')
    <button type="submit">Save</button>
</form>
```

Kode 7.6: `{{ }}` meng-escape semua yang dicetaknya. `@csrf` menambahkan token yang tanpanya Laravel menjawab 419 Page Expired. Isian bersama di `_form.blade.php` memakai `old('name', $customer->name)` untuk mengembalikan apa yang diketik pengguna setelah validasi gagal, dan `@error('name')` menampilkan pesannya. Layout memakai `@extends`, `@section`, dan `@yield`. Buku: Subbab 7.10.1 dan 7.10.2.

Razor dan Blade Berdampingan

Tabel 7.3 Konstruksi Razor dan Blade untuk tugas yang sama.

Tugas	Razor	Blade
cetak, ter-*escape*	`@Model.Name`	`{{ $customer->name }}`
cetak, mentah	`@Html.Raw(x)`	`{!! $x !!}`
layout	`_Layout`, `@RenderBody()`	`@extends`, `@yield`
partial	`<partial name="_Form" />`	`@include('customers._form')`
anti-pemalsuan	*tag helper*, otomatis	`@csrf`

Tabel 7.3: gagasannya adalah gagasan MiniFrame di Bab 2; hanya ejaannya yang berubah. Halaman pesanan menampilkan tombol alur kerja, daftar pembayaran, dan formulir pembayaran hanya selama masih ada sisa tagihan. Buku: Subbab 7.10.3 dan 7.10.4.

Menjalankannya 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
```

Buat proyek baru, salin berkas-berkas bab ini ke atasnya, bangun basis data beserta data contoh, lihat rutenya, lalu nyalakan server. Kemudian ulangi tur terpandu Bab 5: tambahkan pelanggan dua kali dengan satu nomor telepon, buat pesanan 2 kg, catat pembayaran, majukan pesanan di alur kerja, ubah harga, coba hapus pelanggan yang punya pesanan. Klik yang sama harus memberi angka dan pesan yang sama dengan versi ASP.NET Core. Bila tidak, perbedaannya adalah bug di salah satu dari keduanya, dan menemukannya adalah latihan yang sangat baik. Buku: Subbab 7.11.

Menjadikannya Lebih Baik dan Memakainya Sungguhan

Nyalakan `Model::preventLazyLoading()` di lingkungan pengembangan, agar `with(...)` yang terlupa gagal dengan jelas alih-alih mengirim satu kueri per baris.

Atur `'timezone' => 'Asia/Jakarta'` di `config/app.php`: dengan UTC bawaan, “hari ini” di Surabaya baru dimulai pukul tujuh pagi.

Di *hosting* PHP murah, arahkan akar web hanya ke `public/`; `.env` dan berkas SQLite tidak boleh pernah bisa dijangkau.

Tiga pengaturan ini hanya butuh semenit dan mencegah tiga insiden Laravel yang paling umum. Buku: Subbab 7.12–7.14.

Kesalahan yang Sering Terjadi

***Mass assignment* tanpa `#[Fillable]`**, atau `$guarded = []` yang membiarkan permintaan mengisi kolom apa pun. **Lupa `@csrf`**: 419.

`{!! !!}` untuk data pengguna: *cross-site scripting*. **Kueri N+1**: perulangan yang membaca `$order->customer->name` tanpa `with('customer')`.

`migrate:fresh` di server; **server web diarahkan ke folder proyek**; **`APP_DEBUG=true` di produksi**; **zona waktu yang salah**.

Perhatikan betapa banyak di antaranya adalah kesalahan Bab 6 dengan baju baru: mass assignment adalah overposting, `@csrf` yang hilang adalah token anti-pemalsuan yang hilang, `{!! !!}` adalah `Html.Raw`. Bahayanya sama di setiap kerangka kerja; hanya nama penjaganya yang berubah. Buku: Subbab 7.15.

Ringkasan: Lima Hal untuk Diingat

1. Laravel 13 di PHP 8.3–8.5: Composer memasangnya, Artisan membuat berkas dan menjalankan migrasi, SQLite sebagai bawaan (Gambar 7.1).
2. Migrasi ditulis dengan tangan dan berjalan menurut urutan nama berkas; *foreign key* memutuskan dengan `restrictOnDelete` atau `cascadeOnDelete`.
3. Model Eloquent adalah *active record*: `#[Fillable]`, `casts()`, relasi, nilai terhitung.
4. `routes/web.php` menyimpan setiap rute; `Route::resource` memberi tujuh rute CRUD; *route model binding* memberi model atau 404.
5. `$request->validate` mengalihkan kembali dengan galat dan masukan lama; Blade meng-*escape* dengan `{{ }}` dan melindungi formulir dengan `@csrf`.

Versi Laravel dan versi ASP.NET Core berbagi rancangan basis data, halaman, dan aturan; keduanya berbeda dalam tempat tinggal setiap bagian dan seberapa banyak yang dikerjakan lewat konvensi. Buku: Subbab 7.16.

Pertemuan Berikutnya

Bab 8 — Evaluasi dan Pembahasan II. Himpunan fitur yang sama di ASP.NET Core dan Laravel, soal demi soal.

Sebelum itu: jalankan `tools/enum_demo.php`, bangun aplikasi Laravel di komputer Anda, dan kerjakan Soal Latihan 7.3 (rute penawaran harga) dan 7.5 (migrasi yang menambah kolom rak).

Saat mengerjakannya, simpan tabel terjemahan Bab 2 dan tabel folder Laravel–ASP.NET Core di dekat Anda. Setiap kali menemui sesuatu yang baru di Laravel, tanyakan dulu padanannya di ASP.NET Core: rute, controller, model, migrasi, view, pengaturan. Hampir selalu ada padanannya, dan menemukannya jauh lebih cepat daripada mempelajari hal baru dari nol. Buku: Subbab 7.17 dan 7.18.

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

Mata kuliah: Pemrograman Berbasis Kerangka Kerja

Buku: Irfan Subakti, *Pemrograman Berbasis Kerangka Kerja: ASP.NET Core, Laravel, dan Spring Boot*, Bab 8

Dosen: Irfan Subakti — yifana@gmail.com

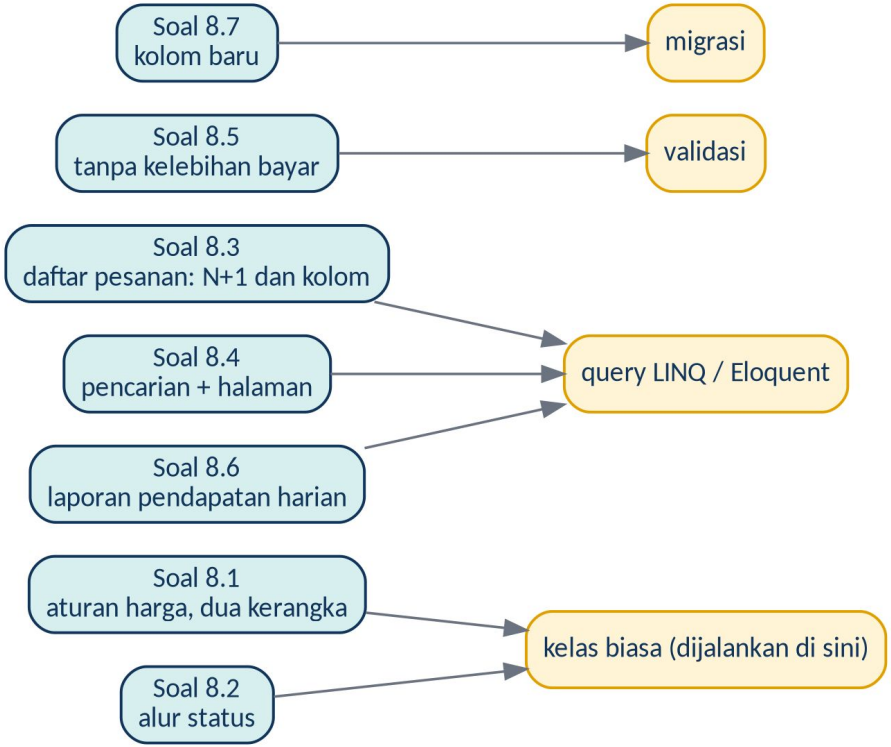
Kini dua aplikasi mengerjakan hal yang sama dengan data yang sama: versi ASP.NET Core dari Bab 5 dan 6 serta versi Laravel dari Bab 7. Justru pada saat seperti itulah perbedaan muncul. Hari ini kita membahas tujuh soal yang diajukan pemilik dalam minggu-minggu setelah keduanya dipakai, dan menyelesaikan masing-masing di kedua kerangka kerja. Buku: Bab 8, Subbab 8.1–8.13.

Cakupan Pertemuan Ini

1. **Rutinitasnya** — enam langkah, dan lapisan tempat setiap soal berada.
2. **Aturan di kelas biasa** — 8.1 satu aturan harga, 8.2 alur kerja pesanan.
3. **Kueri** — 8.3 daftar pesanan tanpa N+1, 8.4 pencarian yang menemukan Sari.
4. **Batas dan laporan** — 8.5 tanpa kelebihan bayar, 8.6 laporan pendapatan harian.
5. **Perubahan skema** — 8.7 rak untuk setiap kantong yang selesai; lalu ketujuhnya berdampingan.

Yang dijalankan: dua program pemeriksa kecil, `CoreCheck` dalam C# dan `tools/core_check.php` dalam PHP biasa, serta demonstrasi PHP untuk kesalahan float. Keluaran mereka hari ini adalah keluaran sungguhan. Aplikasi webnya tidak bisa dijalankan di tempat buku ini ditulis (tanpa NuGet, tanpa Packagist): kode C# dikompilasi terhadap pengganti EF Core yang hanya untuk kompilasi, dan setiap berkas PHP lolos `php -l`. Buku: Subbab 8.2.2.

Enam Langkah, Tujuh Soal



Gambar 8.1 Tujuh soal di bab ini dan lapisan tempat masing-masing berada.

Gambar 8.1: enam langkahnya: nyatakan soal dengan kata-kata toko; kerjakan contoh dengan tangan; tentukan lapisannya (aturan murni, kueri, controller, view, skema); tulis di satu kerangka kerja; pindahkan dan bandingkan; coba patahkan. Menentukan lapisan lebih dulu itulah yang menjaga kedua versi tetap sama. Rutinitas ini sama dengan rutinitas Bab 4, dengan satu langkah tambahan: pindahkan dan bandingkan. Membandingkan dua implementasi adalah cara yang sangat ampuh untuk menemukan bug, karena setiap perbedaan keluaran pasti berarti salah satu dari keduanya keliru, dan sering kali keduanya. Buku: Subbab 8.2.1.

Soal 8.1: Satu Aturan Harga, Dua Kerangka Kerja

Aturan Bab 3: layanan per kg minimal 3 kg; per potong atau pasang dihitung utuh, dibulatkan ke atas; antar-jemput Rp 10.000, gratis mulai subtotal Rp 100.000.

C# menghitungnya di `PriceCalculator` dengan `decimal`. Kode Laravel Bab 7 menghitungnya langsung dengan *float* PHP, dan hasilnya cocok hanya berkat `round`.

Dengan tangan: 4,5 kg dengan Rp 8.000 adalah Rp 36.000; 2 kg ditagih 3 kg, Rp 24.000; 4,02 kg adalah Rp 32.160; 1,5 pasang sepatu dengan Rp 45.000 ditagih 2 pasang, Rp 90.000.

Kekhawatiran pemilik sederhana: pelanggan yang memakai kedua gerai tidak boleh melihat dua harga untuk kantong yang sama. Minta kelas menghitung 4,02 kg dan 1,5 pasang dengan tangan sebelum melihat kode apa pun; jawaban tangan itu adalah kasus uji program pemeriksa. Buku: Subbab 8.3.1 dan 8.3.2.

Aturan dalam Bilangan Bulat

$$b = \begin{cases} \max(h,\ 300) & \text{per kg} \\ 100 \cdot \lceil h/100 \rceil & \text{per potong atau pasang} \end{cases} \tag{8.1}$$

$$S = \left\lfloor \frac{P \cdot b + 50}{100} \right\rfloor \tag{8.2}$$

Meja kasir menerima paling banyak dua desimal, sehingga setiap jumlah adalah bilangan bulat h dalam perseratus: 4,02 kg adalah 402. Persamaan (8.1) memberi jumlah tertagih b dalam perseratus; Persamaan (8.2) memberi subtotal dalam rupiah untuk harga P, dibulatkan setengah ke atas. Untuk 4,02 kg: $b = 402$ dan $S = \lfloor 3216050/100 \rfloor = 32160$. Hanya perkalian, penjumlahan, dan pembagian bilangan bulat: tanpa kejutan pembulatan. Buku: Subbab 8.3.2.

Rancangannya: Kelas Biasa dan Program Pemeriksa

Aturan ini tidak butuh permintaan maupun basis data, jadi tempatnya di **kelas biasa**: `App\Domain\PriceCalculator` di PHP, di samping `PriceCalculator` di C#.

Kelas PHP menerima jumlah sebagai *string*, persis seperti datang dari formulir, dan mengubahnya ke perseratus tanpa pernah menjadi *float*.

Program pemeriksa di setiap bahasa menjalankan sembilan kasus yang sama dan mencetak baris yang sama; keluaran yang sama membuktikan aturan yang sama.

```
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
  Shoes deep clean          1.50 pair          90000
```

Sebagian keluaran sungguhan, identik dari `CoreCheck` dan dari `core_check.php`. Perhatikan bed cover: Rp 70.000 ditambah biaya antar-jemput, karena subtotalnya di bawah Rp 100.000. Buku: Subbab 8.3.3–8.3.5.

Penyelesaian Keliru yang Menggoda: *Float*

```
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
```

Keluaran sungguhan `tools/wrong_price.php`. 4,02 tidak bisa disimpan persis sebagai float biner, sehingga `8000 * 4.02` menjadi 32159,999..., dan mengubahnya dengan `(int)` kehilangan satu rupiah. `round` menutupinya di sini, tetapi aturan yang benar "berkat `round`" itu rapuh. Uang dan jumlah paling aman sebagai bilangan bulat. Buku: Subbab 8.3.6.

Soal 8.2: Alur Kerja Pesanan di Satu Tempat

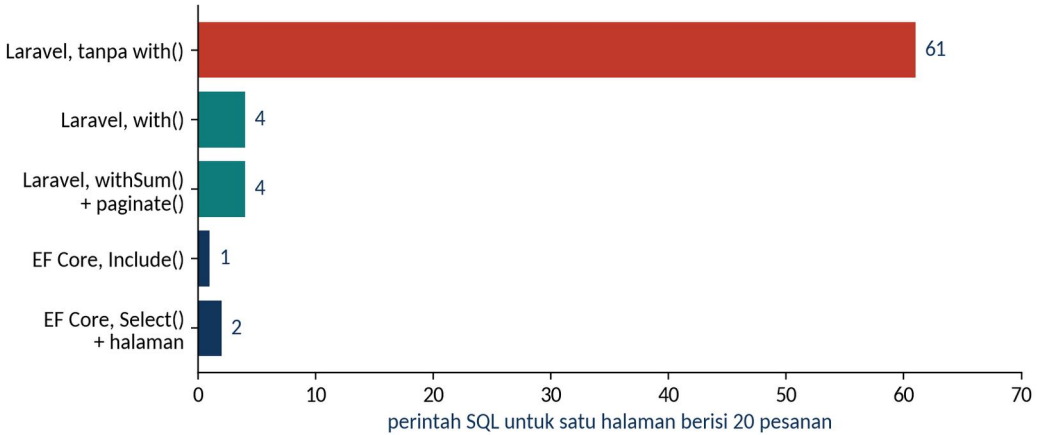
Seorang pelanggan pulang membawa cuciannya tanpa membayar Rp 16.000 yang masih terutang. Aturan baru: *pesanan tidak boleh ditandai sudah diambil selama masih ada utang.*

Alur queued → finished → departed pindah ke kelas biasa, `OrderWorkflow`, di kedua bahasa, lengkap dengan aturan baru itu.

```
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 sungguhan kedua program pemeriksa. Penyelesaian keliru yang menggoda di C#: memuat pesanan dengan `FindAsync`, seperti di Bab 5, lalu menanyakan `order.Balance`. `Payments` tidak dimuat, sehingga `Paid` bernilai 0 dan setiap pesanan ditolak saat diambil, lunas atau tidak. Perbaikannya adalah `Include(o => o.Payments)`; di Laravel, `sum('amount')` pada relasinya. Buku: Subbab 8.4.

Soal 8.3: Daftar Pesanan Tanpa N+1



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.

Gambar 8.2: di Laravel tanpa eager loading, view menyentuh tiga relasi per pesanan: 1 + 3 × 20 = 61 perintah. Dengan `with([...])`, seperti di Bab 7: 4. Di EF Core dengan tiga `Include`: 1 perintah, tetapi berisi setiap kolom dari empat tabel dan setiap baris pembayaran, padahal halaman hanya butuh tujuh nilai per pesanan. Buku: Subbab 8.5.2.

Paging dan Proyeksi di Basis Data

ASP.NET Core: `Select` ke *record* kecil `OrderRow` berisi persis kolom daftar, dan `PagedList` dengan `Skip` dan `Take`: dua perintah, satu untuk hitungan dan satu untuk halamannya.

Laravel: `select` kolom-kolomnya, `with` hanya pada kolom yang dibutuhkan, `withSum('payments', 'amount')`, dan `paginate(20)`: empat perintah.

Pertahankan kueri sebagai `IQueryable` sampai langkah terakhir; `ToList` sebelum `Where`, `Skip`, atau `Sum` membaca seluruh tabel ke memori.

Halaman yang sama, angka yang sama, dengan sebagian kecil pekerjaan. Dengan ribuan pesanan, bedanya adalah beda antara halaman yang terbuka dan halaman yang kehabisan waktu. Buku: Subbab 8.5.3–8.5.6.

Soal 8.4: Pencarian Pelanggan yang Menemukan Sari

Kasir mengetik `sari`: aplikasi ASP.NET Core tidak menemukan siapa pun, aplikasi Laravel menemukan Sari Wulandari. `Contains` menjadi `instr`, peka huruf besar-kecil di SQLite; `LIKE` tidak.

Pemilik meminta pencarian yang sama di keduanya: abaikan huruf besar-kecil, cocokkan nama atau telepon, 15 pelanggan per halaman, pertahankan pencarian saat berpindah halaman.

C#: `EF.Functions.Like`, `OrderBy` lalu `ThenBy`, satu *partial* `_Pager` bersama. **Laravel:** `where` yang dikelompokkan, `paginate(15)->withQueryString()`.

Kenali fungsi string milik provider Anda: LINQ yang sama bisa berarti SQL yang berbeda di SQLite, SQL Server, atau MySQL. Buku: Subbab 8.6.1–8.6.5.

Dua Penyelesaian Keliru yang Menggoda untuk 8.4

`orWhere` tanpa kelompok: `where name like ? or phone like ?` benar sampai seseorang menambah "hanya gerai ini": `and` mengikat lebih kuat daripada `or`, sehingga setiap nama yang cocok muncul, apa pun gerainya. *Closure* `where(fn ($q) => ...)` menambahkan tanda kurungnya.

Paging tanpa `OrderBy`: SQLite bisa mengembalikan baris dalam urutan berbeda untuk halaman 1 dan halaman 2, sehingga seorang pelanggan muncul di keduanya atau tidak di mana pun. EF Core bahkan memperingatkannya di log.

Urutkan berdasarkan kolom unik di akhir, misalnya `Id`, agar urutannya lengkap. Kedua kesalahan ini lolos semua uji dengan sepuluh pelanggan. Buku: Subbab 8.6.6.

Soal 8.5: Tanpa Kelebihan Bayar

Seorang kasir mengetik Rp 60.000 alih-alih Rp 16.000, dan pesanan menampilkan sisa tagihan −Rp 44.000. Aturannya: *pembayaran tidak boleh melebihi sisa utang, dan pesannya menyebut berapa sisanya.*

C#: hitung sisa tagihan dalam satu kueri di server, lalu `ModelState.AddModelError`. **Laravel:** aturan `"max:{$balance}"`, dibangun dari sisa tagihan yang baru dibaca.

Keliru: membandingkan dengan *total* (menerima Rp 36.000 di atas Rp 20.000 yang sudah dibayar), atau dengan sisa tagihan dari isian tersembunyi (siapa pun bisa mengubahnya).

Batas yang bergantung pada data tersimpan dihitung di server pada saat permintaan, tidak pernah diambil dari formulir. Meja kasir laundry jarang punya dua kasir untuk satu pesanan, jadi penyelesaiannya menerima risiko itu; sistem yang sibuk akan menjalankan pemeriksaan dan penyimpanan dalam satu transaksi dengan kunci, atau memeriksa ulang sisa tagihan setelah menyimpan. Buku: Subbab 8.7.

Soal 8.6: Laporan Pendapatan Harian

Setiap malam pemilik mencocokkan laci uang, mutasi QRIS, dan daftar transfer bank dengan sistem: *uang diterima per hari, dipisah menurut metode pembayaran, untuk satu minggu pilihan.*

C#: `GroupBy` tanggal dan metode di basis data ke dalam `DailyRevenueRow`, lalu diputar di memori. **Laravel:** `selectRaw` dengan `groupByRaw`, lalu *collection*.

Hari adalah rentang dari tengah malam ke tengah malam di zona waktu toko, `Asia/Jakarta`.

Keliru: mengelompokkan berdasarkan `PaidAt` alih-alih tanggalnya menghasilkan satu baris per pembayaran. Menyimpan jumlah sebagai `decimal` akan membuat `Sum` gagal di SQLite pada EF Core; laporan ini berjalan karena Bab 5 menyimpan rupiah utuh. Dan aplikasi Laravel yang dibiarkan pada UTC menghitung tujuh jam pertama sebuah hari sebagai kemarin. Agregasikan di SQL, bentuk di memori. Buku: Subbab 8.8.

Soal 8.7: Rak untuk Setiap Kantong yang Selesai

Kantong yang selesai menunggu di rak bernomor. Isian *rak* seperti `B-03` (satu huruf kapital, tanda hubung, dua angka), opsional, diisi di meja kasir dan tampil di halaman pesanan.

C#: sebuah properti, aturan di `OrderForm`, `dotnet ef migrations add`. **Laravel:** migrasi tulisan tangan dengan `Schema::table`, `up` menambah kolom dan `down` menghapusnya.

Keliru: menyunting migrasi yang dulu membuat tabel pesanan. Di laptop, `migrate:fresh` menutupinya; di server, migrasi lama tercatat sudah diterapkan dan dilewati, kolomnya tidak pernah muncul, dan pesanan pertama yang memakai rak gagal.

Migrasi yang sudah meninggalkan komputer Anda adalah sejarah. Perubahan skema selalu berupa migrasi baru. Buku: Subbab 8.9.

Soal-Soal Berdampingan

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`	perseratus bulat	aturan sama, terbukti sama
8.2 alur kerja	`OrderWorkflow` + `Include`	`OrderWorkflow` + `sum`	muat yang dipakai menghitung
8.3 daftar	`OrderRow`, `PagedList`	`withSum`, `paginate`	paging dan proyeksi di basis data
8.4 pencarian	`EF.Functions.Like`	`where` berkelompok	kenali *provider* Anda
8.5 pembayaran	sisa tagihan, `ModelState`	`"max:{$balance}"`	batas dari basis data
8.6 laporan	`GroupBy`, diputar	`selectRaw`, *collection*	agregasi di SQL
8.7 rak	`migrations add`	`Schema::table`	selalu migrasi baru

Tabel 8.1: bacakan kolom terakhir keras-keras. Tidak ada pelajaran yang milik satu kerangka kerja saja; masing-masing adalah kebiasaan yang membuat kerangka kerja apa pun aman dipakai. Buku: Subbab 8.10.

Kesalahan yang Sering Terjadi

Memindahkan kode, bukan kebutuhannya; **menghitung dari relasi yang tidak dimuat**; **memindahkan pekerjaan ke memori tanpa sengaja**.

Paging tanpa urutan yang lengkap; `orWhere` **tanpa kelompok**; **memercayai angka dari formulir**.

Mengelompokkan berdasarkan cap waktu, bukan tanggalnya, dan zona waktu yang salah; **menyunting migrasi yang sudah dijalankan**.

Kesalahan pertama adalah akar dari yang lain. Kode harga Laravel di Bab 7 adalah salinan setia dari kode C#, lengkap dengan float-nya, dan cocok hanya berkat `round`*. Pindahkan aturannya, yang dinyatakan dengan kata-kata toko dan dikerjakan dengan tangan, lalu buktikan kesamaannya pada kasus tetap dengan program pemeriksa. Kebiasaan itu akan menangkap Soal 8.1, 8.2, dan 8.4 sebelum ada pelanggan yang menemukannya. Buku: Subbab 8.12.*

Ringkasan: Lima Hal untuk Diingat

1. Enam langkah: nyatakan, kerjakan dengan tangan, tentukan lapisan, tulis, pindahkan dan bandingkan, coba patahkan (Gambar 8.1).
2. Aturan yang tidak butuh web maupun basis data tinggal di kelas biasa, diperiksa oleh program konsol; bilangan bulat untuk uang (Persamaan (8.2)).
3. Daftar di-*paging* dan diproyeksikan di basis data: 2 perintah di EF Core, 4 di Eloquent, alih-alih 61 (Gambar 8.2).
4. Batas berasal dari data tersimpan di server; laporan mengelompokkan di SQL per hari menurut zona waktu toko.
5. Perubahan skema adalah migrasi baru, tidak pernah suntingan pada migrasi lama.

Kedua aplikasi kini sama dalam perilaku, bukan hanya dalam tampilan. Itulah standar untuk setiap pemindahan antar-kerangka kerja: masukan yang sama memberi keluaran yang sama, dan sebuah program membuktikannya. Buku: Subbab 8.13.

Pertemuan Berikutnya

Bab 9 — Laravel II. Migrasi dan *seeder* lebih dalam, *middleware*, dan *form request*.

Sebelum itu: jalankan `CoreCheck` dan `core_check.php` lalu bandingkan keluarannya baris demi baris; jalankan `wrong_price.php`; kerjakan soal latihan bab ini.

Bab 9 kembali ke Laravel saja dan memperdalam alat-alat yang baru kita pakai sekilas: migrasi pada tabel yang sudah berisi data, data uji yang meyakinkan, middleware untuk aturan toko, dan form request sebagai rumah bagi pemeriksaan seperti Soal 8.5. Latihan yang baik untuk berpasangan: satu mahasiswa menambah aturan harga baru di C#, yang lain di PHP, tanpa saling melihat; lalu kedua program pemeriksa harus tetap mencetak baris yang identik. Buku: Subbab 8.14 dan 8.15.

Kuliah 9 — Laravel II: Migrasi, Seeder, Middleware, dan Form Request

Mata kuliah: Pemrograman Berbasis Kerangka Kerja

Buku: Irfan Subakti, *Pemrograman Berbasis Kerangka Kerja: ASP.NET Core, Laravel, dan Spring Boot*, Bab 9

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini aplikasi Laravel menjadi dewasa. Pesanan belajar gerai mana yang menerimanya, lewat migrasi pada tabel yang sudah berisi data; sebulan data uji yang meyakinkan dibuat oleh factory dan seeder; middleware menutup pesanan yang sudah meninggalkan toko; dan form request menaruh daftar periksa meja kasir di satu tempat. Buku: Bab 9, Subbab 9.1–9.12.

Cakupan Pertemuan Ini

1. **Gerai ketiga** — empat hal yang mulai terasa sakit bersamaan.
2. **Migrasi lebih dalam** — *batch*, mengubah tabel berisi data, perintah-perintahnya, empat aturan.
3. ***Factory* dan *seeder*** — data karangan yang masuk akal, data sungguhan selalu, data karangan hanya di tempatnya.
4. ***Middleware*** — bawang dalam PHP biasa, *middleware* “sebelum” dan “sesudah”, mendaftarkan dan memasangnya.
5. ***Form request*** — daftar periksa untuk satu jenis permintaan, dan controller yang memercayai masukannya.

Seperti di Bab 7, Laravel tidak bisa dipasang di tempat buku ini ditulis; langkah-langkahnya ditulis dari kode dan dokumentasi Laravel 13, setiap berkas PHP diperiksa, dan demo pipeline dalam PHP biasa dijalankan, sehingga keluarannya hari ini adalah keluaran sungguhan. Buku: Bab 9 dan Subbab 9.8.

Gerai Ketiga Bu Wati

Rungkut sejak 2019, Keputih sejak 2022, Wiyung sejak Januari 2026. Pelanggan berpindah-pindah gerai, tetapi aplikasi tidak tahu di gerai mana sebuah pesanan diterima.

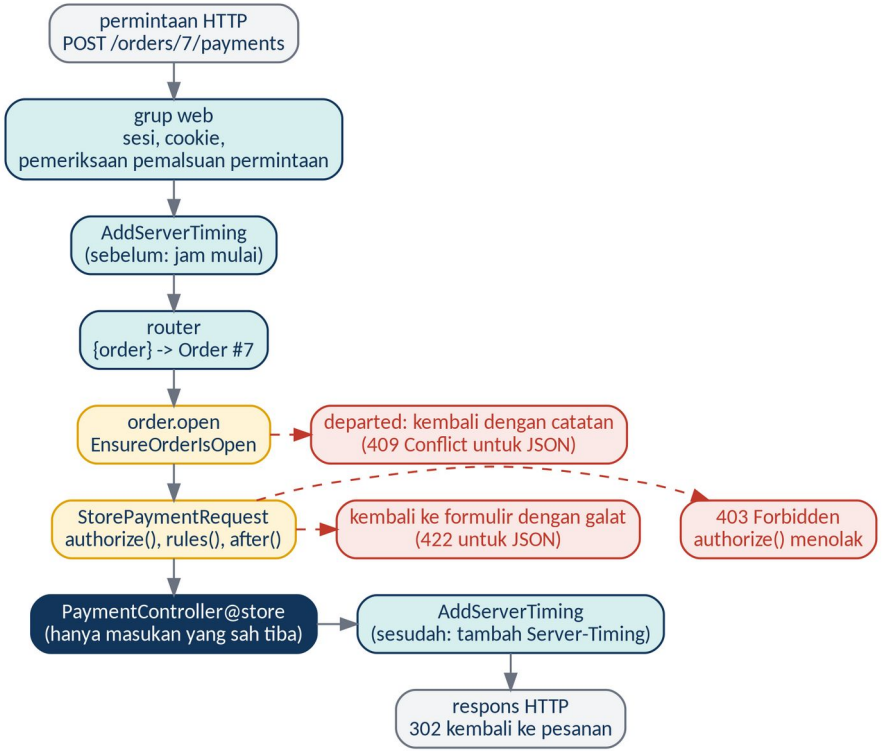
Basis data: Dimas mencoba perubahan di laptopnya, lalu di komputer toko; seminggu kemudian tidak ada yang tahu komputer mana punya kolom apa.

Data uji: halaman kasir dan daftar pesanan tidak bisa dinilai dengan tiga pesanan contoh.

Uang: sebuah pesanan yang selesai dibayar dua kali, tunai dan lewat QRIS, dan tidak ada yang mencegah pembayaran pada pesanan yang sudah dibawa pulang pelanggan. Sementara itu, daftar aturan validasi terus bertambah di dalam controller.

Setiap rasa sakit punya perkakas Laravel: migrasi sebagai buku catatan, factory dan seeder untuk sebulan yang meyakinkan, middleware sebagai pos pemeriksaan, form request sebagai daftar periksa meja kasir. Buku: Subbab 9.1.

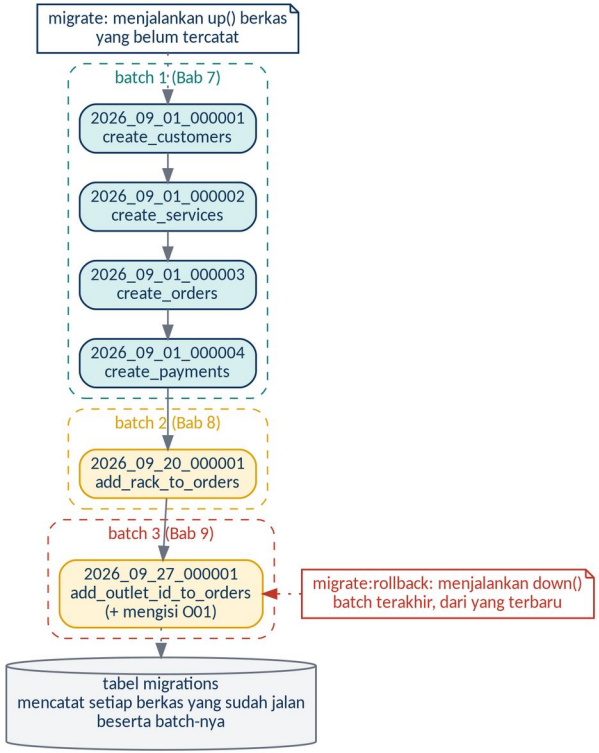
Gagasannya: Pos Pemeriksaan Sebelum Controller



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*.

Gambar 9.1: middleware berjalan lebih dulu, route model binding mengubah `{order}` *menjadi* `Order`*, form request memeriksa masukan, dan baru setelah itu controller berjalan. Setiap kotak merah bisa menghentikan permintaan: pesanan yang sudah tutup, pesanan yang tidak ada (404), masukan yang tidak valid (kembali ke formulir). Controller yang tersisa pendek dan memercayai apa yang diterimanya. Urutan kotak-kotak itu penting. Karena middleware berjalan lebih dulu, pesanan yang sudah diambil ditolak sebelum masukannya diperiksa; karena form request berjalan sebelum controller, controller tidak pernah menerima jumlah pembayaran yang melebihi sisa tagihan. Setiap aturan diperiksa tepat satu kali, di tempat yang paling awal yang bisa memeriksanya. Buku: Subbab 9.2.*

Apa yang Diingat Basis Data



Gambar 9.2 Riwayat migrasi proyek: setiap kali `migrate` dijalankan terbentuk satu *batch*, tabel `migrations` mencatatnya, dan `migrate:rollback` hanya membatalkan *batch* terakhir.

*Gambar 9.2: migrasi adalah perubahan yang tercatat dan bisa dibatalkan. Tabel `migrations` mengingat berkas mana yang sudah dijalankan dan dalam **batch** mana; `migrate` menjalankan berkas yang belum sebagai satu batch baru, `migrate:rollback` menjalankan `down()` dari batch terakhir, dimulai dari yang terbaru. Karena itulah kedua komputer Dimas bisa dibawa ke keadaan yang sama: masing-masing tahu persis apa yang dimilikinya. Buku: Subbab 9.4.1.*

Mengubah Tabel yang Sudah Berisi Pesanan

Kode 9.1 Dua metode `2026_09_27_000001_add_outlet_id_to_orders_table.php`: kolom baru dengan *foreign key*, dan pesanan lama diisi.

```
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');
    });
}
```

*Kode 9.1: kolom wajib tidak bisa ditambahkan ke tabel yang berisi baris, jadi kolomnya ditambahkan **nullable**, lalu diisi: semua pesanan sebelumnya milik Rungkut, `001`. Perubahan data memakai `DB::table()`, bukan model `Order`, karena model bisa berubah kelak dengan cara yang tidak bisa dipenuhi tabel lama. `down()` menghapus foreign key dan kolomnya: pembatalan yang sesungguhnya. Buku: Subbab 9.4.2.*

Perintah-Perintahnya, dan Empat Aturan

Tabel 9.1 Beberapa perintah migrasi Artisan dan kapan memakai masing-masing.

Perintah	Pakai ketika
`php artisan migrate:status`	sebelum dan sesudah setiap perubahan
`php artisan migrate`	menerapkan perubahan sebagai satu *batch*
`php artisan migrate --pretend`	meninjau SQL-nya di basis data sungguhan
`php artisan migrate:rollback`	membatalkan *batch* terakhir
`php artisan migrate:fresh --seed`	hanya di komputer Anda sendiri

Tabel 9.1. Empat aturannya, yang juga berlaku untuk EF Core: jangan pernah menyunting migrasi yang sudah berjalan di mana pun selain komputer Anda sendiri; jadikan setiap `down()` pembatalan yang sesungguhnya dan uji sekali; tambahkan kolom sebagai nullable atau dengan nilai bawaan, lalu isi; jaga perubahan data di dalam migrasi tetap kecil dan ditulis dengan query builder. Buku: Subbab 9.4.3 dan 9.4.4.

Factory: Satu Rekaman yang Masuk Akal

Factory menggambarkan satu rekaman karangan yang masuk akal: Faker untuk nama dan alamat, *factory* terkait atau *closure* untuk *foreign key*.

`OrderFactory` menghargai setiap pesanan dengan **aturan sungguhan** di `afterMaking`, alih-alih mengarang total yang tidak mungkin dihasilkan aplikasi.

State memberi nama pada variasi: `finished()`, `departed()`. `BusyMonthSeeder` memberi setiap pesanan riwayat yang masuk akal: tujuh dari sepuluh lunas saat diantar, selesai setelah dua hari, diambil setelah lima hari hanya bila lunas, karena alur kerja Bab 8 tidak mengizinkan yang lain.

Modelnya butuh trait `HasFactory`. Nilai acak di tempat yang seharusnya ditentukan aturan menghasilkan data yang berbohong: laporan yang dibangun di atas total karangan menampilkan angka yang tidak mungkin dimiliki toko sungguhan. Buku: Subbab 9.5.1 dan 9.5.2.

Seeder: Data Sungguhan Selalu, Data Karangan Hanya di Sini

`PriceListSeeder` memasukkan daftar harga dan gerai sungguhan ke setiap basis data dengan `updateOrCreate`, sehingga aman dijalankan lagi (***idempoten***).

`BusyMonthSeeder` mengarang empat puluh pelanggan dan sebulan pesanan yang bergerak di alur kerja seperti pesanan sungguhan.

`DatabaseSeeder` selalu memanggil daftar harga, dan memanggil bulan sibuk hanya di komputer pengembang atau komputer uji, dijaga oleh lingkungannya.

> **PERINGATAN:** *Seeder* yang memanggil *seeder* demo tanpa syarat memasukkan pelanggan karangan ke sistem sungguhan toko pada kali pertama seseorang menjalankan `db:seed` di sana.

Sebulan yang meyakinkan membuat halaman kasir, daftar pesanan dengan halamannya, dan laporan harian Bab 8 layak diamati. Buku: Subbab 9.5.3 dan 9.5.4.

Penjaga di `DatabaseSeeder`

Kode 9.2 `database/seeders/DatabaseSeeder.php`: data sungguhan selalu, data karangan hanya di komputer pengembang atau komputer uji.

```
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
        }
    }
}
```

Kode 9.2: dua baris memuat seluruh kebijakannya. `app()->environment('local', 'testing')` bernilai benar di laptop pengembang dan di test runner, dan salah di komputer toko, tempat `APP_ENV=production`. Jadi `php artisan db:seed` aman diketik di mana saja: daftar harga dan gerai diperbarui, dan tidak ada data karangan yang muncul di tempat pemilik menghitung uang sungguhan. Gagasan yang sama melindungi akun demo di Bab 6, yang hanya tinggal di `appsettings.Development.json`. Buku: Subbab 9.5.3.

Middleware: Bawang, Tanpa Laravel

```
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"}
```

Keluaran sungguhan `tools/pipeline_demo.php`: empat puluh baris PHP biasa, tempat setiap lapisan adalah closure yang menerima permintaan dan `$next`, dan `array_reduce` membungkus controller dengan lapisan-lapisan itu. Untuk pesanan yang sudah diambil, lapisan dalam tidak pernah memanggil `$next`, sehingga controller tidak berjalan, tetapi lapisan pengukur waktu di luar tetap menyelesaikan tugasnya dalam perjalanan keluar. Inilah bawang Bab 3 dalam PHP. Buku: Subbab 9.6.1.

Middleware "Sebelum" untuk Aturan Toko

Kode 9.3 Metode `handle` di `app/Http/Middleware/EnsureOrderIsOpen.php`: tidak ada yang boleh mengubah pesanan yang sudah meninggalkan toko.

```
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);
}
```

Kode 9.3: pesanan yang sudah meninggalkan toko adalah sejarah. Orang mendapat catatan di halaman asalnya; program mendapat ***409 Conflict****. Setiap jalur mengembalikan respons: miliknya sendiri, atau respons dari* `$next`*. Parameter rute tersedia hanya karena ini middleware rute; middleware global berjalan sebelum perutean dan akan melihat* `null`*. Buku: Subbab 9.6.2.*

Middleware "Sesudah", Mendaftarkan dan Memasang

`AddServerTiming` memanggil `$next` lebih dulu, lalu menambahkan *header* `Server-Timing` berisi waktu yang dipakai sisa *pipeline*.

Di `bootstrap/app.php`: `AddServerTiming` masuk grup `web`; `EnsureOrderIsOpen` mendapat nama pendek (*alias*) `order.open`.

Di `routes/web.php`, rute-rute yang mengubah pesanan, *advance* dan pembayaran, dikelompokkan di balik `Route::middleware('order.open')`.

Middleware rute bawaan Laravel yang dipakai buku ini antara lain `auth`, `guest`, `throttle:5,1`, `can:manage-prices`, `verified`, dan `signed`. Kelompokkan rutenya sekali, maka rute baru yang ditambahkan ke grup itu tahun depan terlindungi tanpa ada yang perlu ingat. Buku: Subbab 9.6.3–9.6.5.

Form Request: Daftar Periksa Meja Kasir

Form request menyimpan daftar periksa untuk satu jenis permintaan: `authorize`, `prepareForValidation` (merapikan masukan), `rules`, `messages`, `attributes`, dan `after`.

`StoreOrderRequest` merapikan masukan lebih dulu (koma pada `4,5` menjadi titik, rak dipangkas dan dijadikan huruf kapital), lalu memeriksa isian-isiannya dan menamainya dengan kata-kata kasir.

Controller memintanya sebagai parameter bertipe; bila ada yang gagal, controller tidak pernah berjalan dan pengguna kembali ke formulir beserta pesannya.

Ini adalah kelas formulir Bab 6 dan `$request->validate` Bab 7 yang digabung menjadi satu kelas, mudah ditemukan menurut namanya di `app/Http/Requests`. Buku: Subbab 9.7.1.

Aturan yang Membutuhkan Pesanannya

Kode 9.4 Metode `after` di `app/Http/Requests/StorePaymentRequest.php`: pemeriksaan terhadap sisa utang, dijalankan setelah aturan sederhana lolos.

```
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.');
            }
        },
    ];
}
```

Kode 9.4: aturan sederhana didahulukan (jumlah berupa bilangan bulat minimal 1.000, metode salah satu kasus `PaymentMethod`); hanya bila lolos, `after` memuat pembayaran pesanan dan membandingkannya dengan sisa tagihan. Itulah Soal 8.5 di rumah yang semestinya, dan `PaymentController::store` menyusut menjadi beberapa baris yang memakai `$request->validated()`. Buku: Subbab 9.7.2 dan 9.7.3.

Menjalankannya dan Menjadikannya Lebih Baik

Di komputer Anda: `migrate:status`, `migrate`, `migrate:rollback`, `migrate` lagi; lalu `migrate:fresh --seed` dan lihat bulan yang sibuk.

Coba membayar pesanan yang sudah diambil, dari halaman dan dengan permintaan JSON; coba membayar lebih dari sisa tagihan.

Lihat *header* `Server-Timing` di perkakas pengembang peramban, pada tab jaringan.

Soal latihan memperluas kedua perkakas: migrasi yang mengisi kolom baru baris demi baris dalam potongan-potongan, dan middleware `LogSlowRequests` *yang menerima parameter, jumlah milidetik. Buku: Subbab 9.8, 9.9, dan 9.14.*

Memakainya Sungguhan

Setiap usaha dengan lebih dari satu lokasi butuh “di mana ini terjadi?” pada catatannya: klinik dengan dua cabang, toko dengan lapak di pasar.

Aturan seperti “catatan yang sudah ditutup tidak boleh berubah” tempatnya di *middleware* atau *policy*, bukan di setiap controller: faktur yang lunas, paket yang sudah terkirim, resep yang sudah ditandatangani.

Pola migrasi hari ini (nullable, isi, perketat kemudian) adalah cara setiap sistem nyata bertumbuh tanpa berhenti. Buku: Subbab 9.10.

Kesalahan yang Sering Terjadi

Menyunting migrasi yang sudah dijalankan; **kolom wajib pada tabel yang berisi baris**; **model di dalam migrasi**; `down()` **yang tidak membatalkan** `up()`.

Data karangan di basis data sungguhan; **nilai acak di tempat yang ditentukan aturan**.

Lupa `return $next($request)`: setiap permintaan mendapat respons kosong. ***Middleware* global yang butuh parameter rute**.

`$request->all()` **setelah validasi**: pakai `validated()` atau `safe()`. **Pemeriksaan basis data di** `rules()` **tanpa penjaga**: taruh di `after()`.

Pembayaran ganda di Subbab 9.1 adalah cerita di balik dua perkakas ini: middleware menutup pesanan yang sudah diambil, dan form request menolak jumlah di atas sisa tagihan. Keduanya tidak membutuhkan satu baris pun di controller. Buku: Subbab 9.11.

Ringkasan: Lima Hal untuk Diingat

1. Migrasi adalah perubahan yang tercatat dan bisa dibatalkan, dalam *batch* (Gambar 9.2); tabel berisi data mendapat kolom *nullable* yang diisi dengan *query builder*.

2. *Factory* menggambarkan satu rekaman yang masuk akal dengan aturan sungguhan; *seeder* menaruh data acuan sungguhan di mana-mana dan data karangan hanya di tempatnya.

3. *Middleware* membungkus controller seperti bawang; "sebelum" boleh menghentikan permintaan, "sesudah" mengubah respons.

4. `EnsureOrderIsOpen` menutup pesanan yang sudah diambil pada setiap rute yang mengubahnya: catatan untuk orang, 409 untuk program.

5. *Form request* adalah daftar periksa satu permintaan; controller berjalan hanya bila semuanya lolos (Gambar 9.1).

Bandingkan dengan ASP.NET Core: di sana migrasi dibangkitkan dari kelas, di sini ditulis dengan tangan, tetapi keduanya menyimpan tabel riwayat dan keduanya butuh empat aturan yang sama; di sana middleware didaftarkan di `Program.cs`, di sini di `bootstrap/app.php`; form request memainkan peran kelas formulir ditambah `ModelState`. Gagasannya berpindah; hanya alamatnya yang berubah. Buku: Subbab 9.12.

Pertemuan Berikutnya

Bab 10 — Laravel III. Autentikasi, *API resource*, dan pengujian.

Sebelum itu: jalankan `tools/pipeline_demo.php` dan tambahkan lapisan ketiga yang mencatat nomor pesanan; di komputer Anda, jalankan migrasinya lalu batalkan sekali.

Perhatikan urutan pengerjaannya: jalankan `migrate:status` sebelum dan sesudah setiap perubahan, dan biasakan membaca riwayat batch-nya. Kebiasaan kecil ini mencegah kebingungan Dimas tentang komputer mana yang punya kolom apa, dan berlaku sama di EF Core dengan tabel `__EFMigrationsHistory`. Buku: Subbab 9.13 dan 9.14.

Kuliah 10 — Laravel III: Autentikasi, API Resource, dan Pengujian

Mata kuliah: Pemrograman Berbasis Kerangka Kerja

Buku: Irfan Subakti, *Pemrograman Berbasis Kerangka Kerja: ASP.NET Core, Laravel, dan Spring Boot*, Bab 10

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini aplikasi Laravel belajar siapa yang bertanya. Staf login dengan sesi, hanya pemilik yang boleh mengubah harga, ponsel para sopir mendapat API dengan token dari Laravel Sanctum, API resource menentukan persis apa isi JSON-nya, dan pengujian otomatis memeriksa setiap aturan, terutama penolakannya. Buku: Bab 10, Subbab 10.1–10.12.

Cakupan Pertemuan Ini

1. **Daftar harga yang bisa diubah siapa saja** — dan sopir yang butuh aplikasi, bukan peramban.
2. **Pengguna dan peran** — pemilik dan kasir; login yang ditulis sendiri; rute dan sebuah *gate*.
3. **API untuk program** — token Sanctum, rute API, satu cara membuat pesanan.
4. ***API resource*** — bentuk JSON sebuah pesanan, ditentukan dengan sengaja.
5. **Pengujian** — *unit test* untuk aturan harga, *feature test* untuk login dan API.

Laravel tidak bisa dipasang di tempat buku ini ditulis, sehingga kodenya ditulis berdasarkan dokumentasi Laravel 13 dan Sanctum dan diperiksa dengan `php -l`*; pengujiannya dimaksudkan untuk Anda jalankan dengan* `php artisan test` *di komputer sendiri. Buku: Bab 10 dan Subbab 10.8.*

Daftar Harga yang Bisa Diubah Siapa Saja

Suatu Sabtu di bulan September, gerai Keputih menagih Rp 6.000 per kilogram alih-alih Rp 8.000. Seorang kasir paruh waktu yang baru membuka **Prices** untuk *melihat*, mengeklik **Edit** di baris yang salah, lalu menyimpan.

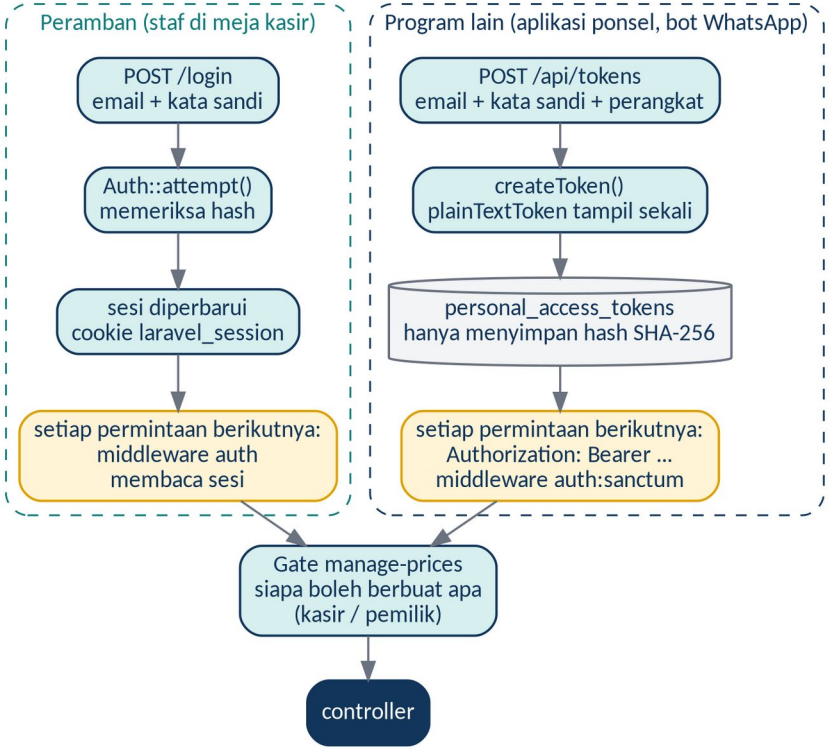
Aplikasi tidak bertanya siapa dia, jadi tidak bisa mencegahnya, dan tidak bisa menjawab sesudahnya siapa yang melakukannya.

Pada minggu yang sama Bu Wati mempekerjakan dua sopir. Mereka perlu melihat pesanan yang harus dijemput, menandai kantong yang sudah diantar, dan mencatat pembayaran di depan pintu pelanggan, dari aplikasi ponsel kecil.

Aplikasi bukan peramban: ia tidak bisa mengisi formulir Blade. Ia butuh **API**, sekumpulan alamat yang berbicara JSON, dan cara aman untuk membuktikan siapa yang bertanya.

Dua cerita, satu tema: aplikasi harus tahu siapa yang bertanya dan memutuskan apa yang boleh dilakukannya, baik untuk orang di depan layar maupun untuk program di ponsel. Buku: Subbab 10.1.

Sesi untuk Peramban, Token untuk Program



Gambar 10.1 Dua cara membuktikan siapa yang bertanya: peramban login sekali dan membawa *cookie* sesi; program menukar kata sandi dengan token dan mengirimkannya pada setiap permintaan; keduanya berakhir di pemeriksaan otorisasi yang sama.

*Gambar 10.1: **autentikasi** menjawab "siapa Anda?", **otorisasi** menjawab "bolehkah Anda melakukan ini?". Peramban login sekali dan cookie sesi ikut dalam setiap permintaan. Program menukar surel dan kata sandi dengan **token** lalu mengirimkannya sebagai `Authorization: Bearer …` pada setiap panggilan. Kedua jalan berakhir di pemeriksaan yang sama. Di ASP.NET Core, Bab 6 memakai cookie dari Identity untuk peramban; Bab 14 akan memakai token bergaya Sanctum yang sama di Spring Boot. Pola dua jalan ini, sesi untuk manusia dan token untuk program, adalah pola umum di hampir setiap aplikasi web modern yang juga punya aplikasi ponsel. Buku: Subbab 10.2.1 dan 10.2.2.*

Pengguna dan Peran

`App\Enums\Role`: `Owner` dan `Cashier`, *backed enum* yang disimpan di kolom baru `role`; setiap pengguna adalah kasir kecuali dinyatakan lain.

`User` memakai `#[Hidden(['password', 'remember_token'])]` dan *cast* `hashed`: kata sandi hanya disimpan sebagai *hash*, dan tidak pernah muncul di JSON.

`UserFactory` membuat kasir secara bawaan dan punya *state* `owner()`; `DatabaseSeeder` membuat dua akun demo hanya di komputer pengembang.

Akun demo, `cashier@laundry.test` dan `owner@laundry.test` dengan kata sandi `password`, mengikuti penjaga Bab 9: keduanya tidak pernah sampai ke basis data sungguhan toko. Buku: Subbab 10.3.

Controller Login

Kode 10.1 Metode `store` di `app/Http/Controllers/Auth/LoginController.php`: 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'));
}
```

Kode 10.1: `Auth::attempt` mencocokkan kata sandi dengan hash-nya. Satu pesan untuk "pengguna tidak ada" dan "kata sandi salah", sehingga orang asing tidak mengetahui akun mana yang ada. `regenerate()` memberi id sesi baru setelah login, menutup pintu bagi session fixation. `intended` mengembalikan pengguna ke halaman yang semula dimintanya. Logout adalah `POST` yang mengeluarkan pengguna, membatalkan sesi, dan membuat ulang token CSRF. Buku: Subbab 10.4.1.

Menjaga Rute, dan Sebuah *Gate*

`routes/web.php` punya empat kelompok: halaman **publik**; login **khusus tamu**, dengan `throttle`; halaman **staf** di balik `auth`; **daftar harga** di balik `can:manage-prices`.

Kode 10.2 Satu baris di `boot()` dalam `app/Providers/AppServiceProvider.php`: *gate* `manage-prices`, hanya untuk pemilik.

```
// Only the owner changes the price list.
Gate::define('manage-prices', fn (User $user) => $user->isOwner());
```

*Kode 10.2: **gate** adalah aturan ya-atau-tidak bernama tentang pengguna saat ini, sepupu Laravel dari policy bernama di Bab 6. Rute memakainya sebagai middleware `can:manage-prices`, dan Blade sebagai `@can('manage-prices')` untuk menampilkan tautannya hanya kepada pemilik. Menyembunyikan tautan hanyalah kerapian; middleware-lah perlindungannya. Buku: Subbab 10.4.2 dan 10.4.3.*

Penolakan, Masing-Masing dengan Kodenya

Tabel 10.1 Penolakan dalam aplikasi bab ini: kode status, arti, dan contohnya.

Kode	Arti	Contoh
302 ke `/login`	tamu meminta halaman staf	membuka `/counter` tanpa login
401	tidak ada sesi atau token yang sah (JSON)	panggilan API tanpa token
403	sudah login, tetapi tidak diizinkan	kasir membuka `/prices`
409	pesanan sudah ditutup	membayar pesanan yang sudah diambil lewat API
419	token CSRF hilang atau kedaluwarsa	formulir dibiarkan terbuka semalaman
422	masukan melanggar aturan	membayar lebih dari sisa tagihan
429	terlalu banyak percobaan	login keenam dalam semenit

Tabel 10.1: setiap penolakan punya kodenya sendiri, dan program di ponsel bisa bertindak atas masing-masing: meminta kata sandi lagi pada 401, menampilkan “tidak diizinkan” pada 403, menampilkan pesan isian pada 422, menunggu pada 429. Pelajari tabel ini; isinya sama di ASP.NET Core dan Spring Boot. Buku: Subbab 10.4.

API untuk Program: Token Sanctum

Kode 10.3 Inti `TokenController::store` di `app/Http/Controllers/Api/TokenController.php`: kata sandi masuk, token keluar.

```
$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);
```

*Kode 10.3: `php artisan install:api` memasang **Sanctum** dan membuat `routes/api.php`. `POST /api/tokens` dengan surel, kata sandi, dan nama perangkat mengembalikan 201 dan sebuah token, ditampilkan sekali saja; Sanctum hanya menyimpan hash-nya. `DELETE /api/tokens/current` mencabut token perangkat ini, sehingga ponsel yang hilang bisa dikeluarkan tersendiri. Buku: Subbab 10.5.1 dan 10.5.2.*

Rute API, dan Satu Cara Membuat Pesanan

`routes/api.php`: token, pelanggan, pesanan, dan pembayaran, di balik `auth:sanctum`, memakai ulang *form request* dan *middleware* `order.open` dari Bab 9.

Sebuah **kelas aksi**, `CreateOrder`, adalah satu-satunya tempat masukan yang sudah diperiksa menjadi pesanan yang dihargai; meja kasir web dan API sama-sama memanggilnya.

`UpdateOrderStatusRequest` membolehkan program meminta status bernama; `OrderWorkflow` Bab 8 yang memutuskan apakah itu langkah berikutnya yang diizinkan.

Tanpa kelas aksi, API akan menumbuhkan salinannya sendiri untuk "membuat pesanan", dan kedua salinan akan saling menjauh persis seperti aturan harga sebelum Bab 8. Buku: Subbab 10.5.3 dan 10.5.4.

Kelas Aksi `CreateOrder`

Kode 10.4 Kelas di `app/Actions/CreateOrder.php`: satu-satunya tempat masukan yang sudah diperiksa menjadi pesanan yang dihargai.

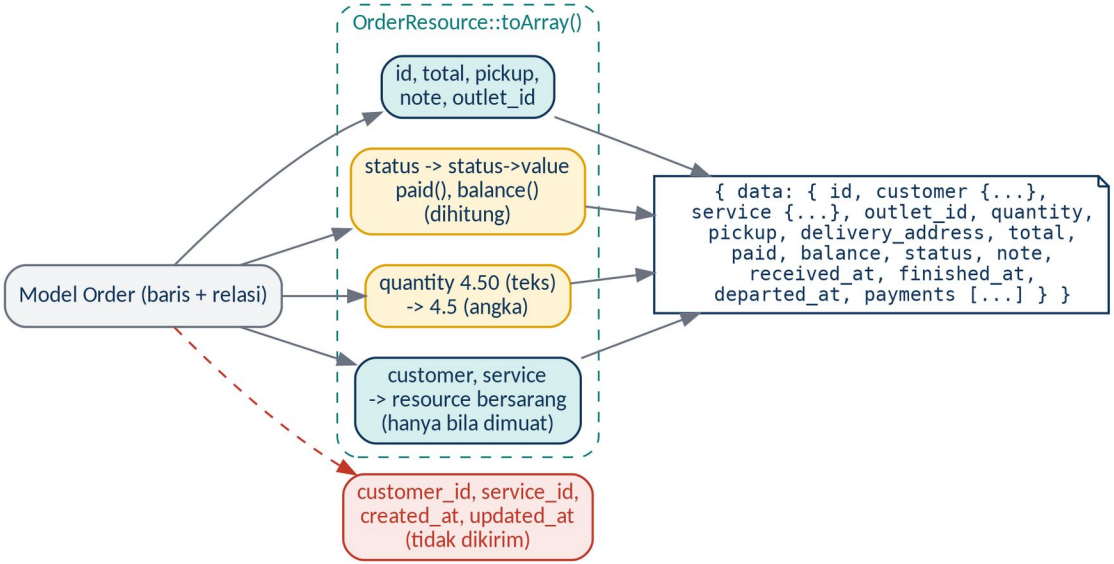
```
// One place that turns checked input into a priced, saved order.
// The web page and the API both call it, so they can never disagree.
class CreateOrder
{
    public function handle(array $data): Order
    {
        $service = Service::findOrFail($data['service_id']);
        $pickup = (bool) ($data['pickup'] ?? 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(),
        ]);
    }
}
```

Kode 10.4: masukannya tiba dalam keadaan sudah diperiksa oleh form request, dari meja kasir web maupun dari API. Aksi ini mencari layanannya (404 bila sudah hilang), menghitung total dengan `PriceCalculator` bilangan bulat dari Bab 8, dan menyimpan pesanan

berstatus antre beserta waktu penerimaannya. Total tidak pernah diambil dari permintaan: aplikasi ponsel tidak bisa mengirim harganya sendiri. `OrderController::store` *di web dan* `OrderController::store` *di API masing-masing menyusut menjadi pemanggilan* `handle` *dan respons dalam bahasanya sendiri, pengalihan atau resource dengan 201. Buku: Subbab 10.5.4.*

API Resource: Menentukan Isi Sebuah Jawaban



Gambar 10.2 Apa yang dilakukan `OrderResource` terhadap sebuah pesanan: isian yang diteruskan, dikonversi, atau dihitung, relasi yang disarangkan bila sudah dimuat, dan kolom internal yang ditinggalkan.

Gambar 10.2: mengembalikan model dari API membuat JSON mengikuti basis data, termasuk kolom yang tidak dimaksudkan untuk diterbitkan. ***Resource*** *mendaftar setiap isian dengan sengaja: sebagian diteruskan, sebagian dikonversi (jumlah sebagai angka, status sebagai nilai string-nya), sebagian dihitung (`paid`, `balance`), relasi disarangkan hanya bila sudah dimuat, dan kolom internal ditinggalkan. Buku: Subbab 10.6.*

`OrderResource`

Kode 10.5 Sebagian `toArray` di `app/Http/Resources/OrderResource.php`: bentuk JSON sebuah pesanan.

```
return [
    'id' => $this->id,
    'customer' => new CustomerResource($this->whenLoaded('customer')),
    'service' => new ServiceResource($this->whenLoaded('service')),
    'quantity' => (float) $this->quantity,
    'total' => $this->total,
    'paid' => $this->paid(),
    'balance' => $this->balance(),
    'status' => $this->status->value,
    'received_at' => $this->received_at,
    'payments' => PaymentResource::collection(
        $this->whenLoaded('payments')),
];
```

Kode 10.5: `whenLoaded('customer')` menyertakan pelanggan hanya bila controller sudah memuatnya dengan `with('customer')`, sehingga resource tidak pernah memicu satu kueri per pesanan. Koleksi yang di-paginate menambahkan `links` dan `meta`, yang dipakai aplikasi ponsel untuk mengambil halaman berikutnya. Buku: Subbab 10.6.

Pengujian: Sepasang Tangan Kedua yang Tak Kenal Lelah

Unit test: `PriceCalculatorTest` memeriksa enam kasus harga pada kelas biasa, tanpa basis data dan tanpa web.

Feature test mengirim permintaan sungguhan melalui seluruh aplikasi: `RefreshDatabase` memberi setiap pengujian basis data yang bersih; `actingAs($user)` dan `Sanctum::actingAs` melakukan login tanpa formulir.

Pengujian memeriksa kode status, jalur JSON, dan isi basis data sesudahnya.

`php artisan test` menjalankan semua yang ada di `tests/Unit` dan `tests/Feature`; `--filter=OrderApiTest` menjalankan satu kelas. Bila sebuah pengujian gagal, laporannya menyebut nama metode, baris assertion, dan apa yang diharapkan. Buku: Subbab 10.2.4, 10.7.1, dan 10.7.4.

Feature Test untuk Login

Kode 10.6 Dua metode `tests/Feature/AuthTest.php`: tamu dan daftar harga milik pemilik.

```
public function test_guests_are_sent_to_the_sign_in_page(): void
{
    $this->get('/counter')->assertRedirect('/login');
}

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();
}
```

Kode 10.6: pengujian kedua adalah insiden hari Sabtu itu, ditulis sebagai aturan yang diperiksa setiap kali siapa pun mengubah apa pun. Kelas lengkapnya juga memeriksa login, kata sandi yang salah, dan token API yang membuka API dan bisa dicabut. `OrderApiTest` *memeriksa harga, validasi, alur kerja, pesanan yang sudah ditutup, pembayaran, dan penyaring lewat API. Buku: Subbab 10.7.2 dan 10.7.3.*

Menjalankannya sebagai Setiap Tokoh dalam Cerita

Buka `/counter`: halaman login. Login sebagai kasir: tidak ada tautan **Prices**, dan mengetik `/prices` menghasilkan 403. Login sebagai pemilik: tautannya berfungsi.

Kata sandi salah enam kali dalam semenit: percobaan keenam ditolak dengan 429 bahkan sebelum kata sandinya diperiksa.

Sebagai sopir: minta token dengan `curl` (di Windows `curl.exe`), daftar pesanan dengannya, tambahkan `?status=finished`.

Jalankan `php artisan install:api` sekali, salin berkas-berkas bab ini, `migrate:fresh --seed`, lalu nyalakan server di port 4100, seperti dijelaskan README bab ini. Ekstensi REST Client di VS Code adalah alternatif yang ramah untuk `curl`. Buku: Subbab 10.8.

Menjadikannya Lebih Baik dan Memakainya Sungguhan

Ability* token** seperti `orders:read` dan `payments:write`, diperiksa dengan `tokenCan`; **token yang kedaluwarsa** lewat `expiration` di `config/sanctum.php`; `throttle:api` untuk API; ***policy ketika satu *gate* tidak lagi cukup.

Sungguhan: satu akun per orang, tidak pernah dipakai bersama; kata sandi panjang yang hanya diketahui pemiliknya; HTTPS di mana pun selain laptop Anda; satu token per perangkat, dicabut ketika perangkatnya tidak ada lagi.

Nama, nomor telepon, dan alamat pelanggan adalah data pribadi menurut UU PDP 27/2022: mengirim hanya isian yang dibutuhkan di setiap resource dan mencabut akses dengan segera adalah bagian dari memperlakukan data itu dengan hati-hati. Dan setiap kali sebuah kesalahan sampai ke toko, tulis pengujian yang akan menangkapnya sebelum Anda memperbaikinya. Buku: Subbab 10.9 dan 10.10.

Kesalahan yang Sering Terjadi

Memeriksa peran hanya di view; memberi tahu orang asing bagian mana yang salah; lupa `session()->regenerate()`; logout lewat GET.

Mengembalikan model, bukan *resource*; `whenLoaded` tanpa memuat; menyimpan atau mencatat token polos.

Pengujian yang saling bergantung atau bergantung pada data Anda; hanya menguji jalur yang mulus.

Hanya menguji jalur yang mulus adalah kesalahan yang diam. Sebagian besar pengujian bab ini memeriksa penolakan: 401, 403, 409, 422, 429. Itulah aturan yang melindungi toko, dan justru aturan itulah yang rusak diam-diam, karena pengguna yang keliru diizinkan tidak pernah mengeluh. Buku: Subbab 10.11.

Ringkasan: Lima Hal untuk Diingat

1. Autentikasi menjawab siapa (gagal: 401), otorisasi menjawab apa (gagal: 403); tamu di peramban dialihkan ke halaman login (Gambar 10.1).
2. Peran sebagai *backed enum*; kata sandi hanya sebagai *hash*; login tulisan sendiri yang membuat ulang sesi; sebuah *gate* untuk daftar harga.
3. Token Sanctum: di-*hash*, dikirim sebagai *bearer token*, dicabut per perangkat; rute API di balik `auth:sanctum`.
4. *Resource* menentukan JSON isian demi isian (Gambar 10.2); kelas aksi membuat pesanan dengan satu cara saja.
5. *Unit test* untuk kelas biasa, *feature test* untuk seluruh aplikasi dengan `RefreshDatabase`, `actingAs`, dan `Sanctum::actingAs`.

Dengan bab ini, versi Laravel sudah menyusul versi ASP.NET Core di Bab 6, dan melangkah satu tahap lebih jauh dengan API dan rangkaian pengujian. Buku: Subbab 10.12.

Pertemuan Berikutnya

Bab 11 — Spring Boot I. Kerangka kerja ketiga: Spring Initializr, controller REST, dan Spring Data JPA, dalam Java 25.

Sebelum itu: jalankan pengujiannya di komputer Anda, rusak *gate*-nya dengan sengaja dan lihat pengujian mana yang gagal, lalu kerjakan soal latihan yang memeriksa daftar harga di setiap rute.

Pengujian yang gagal karena gate dirusak dengan sengaja adalah bukti terbaik bahwa pengujiannya berguna: pengujian yang tidak pernah bisa gagal tidak melindungi apa pun. Setelah itu, perbaiki gate-nya dan pastikan semua pengujian kembali lolos. Bab 11 lalu membangun API yang sama di Spring Boot, dan Bab 12 membandingkan keduanya isian demi isian. Buku: Subbab 10.13 dan 10.14.

Kuliah 11 — Spring Boot I: Initializr, REST Controller, dan Spring Data JPA

Mata kuliah: Pemrograman Berbasis Kerangka Kerja

Buku: Irfan Subakti, *Pemrograman Berbasis Kerangka Kerja: ASP.NET Core, Laravel, dan Spring Boot*, Bab 11

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini kita berkenalan dengan kerangka kerja ketiga. Kita membangun API Profitina Laundry di Spring Boot 4.1 dengan Java 25: empat tabel yang sama, daftar harga yang sama dari `Service.csv` yang sama, aturan yang sama, dan nama isian JSON yang sama dengan API Laravel di Bab 10. Sepanjang jalan: application context, starter dan konfigurasi otomatis, entitas JPA, repository Spring Data, dan controller REST. Buku: Bab 11, Subbab 11.1–11.13.

Cakupan Pertemuan Ini

1. **Pertanyaan Raka** — apakah ini akan tampak sama di Spring Boot?
2. **Gagasannya** — container yang membangun objek Anda, bawaan dari Boot, entitas dan *repository*, controller yang berbicara JSON.
3. **Proyeknya** — Spring Initializr, `pom.xml`, titik masuk, pengaturan.
4. **Domainnya** — empat entitas, status yang disimpan sebagai kata yang disepakati, aturan dalam Java biasa.
5. ***Repository* dan controller** — kueri dari nama metode, *record* masuk dan keluar, memuat daftar harga.

Yang dijalankan: paket Maven tidak bisa diunduh di tempat buku ini ditulis, sehingga aplikasi Spring tidak dinyalakan di sana. `LaundryRules.java` dan program pemeriksa `RulesCheck.java` hanya butuh JDK; keduanya dikompilasi dengan `javac` dan dijalankan, dan keluarannya hari ini adalah keluaran sungguhan. Buku: Bab 11 dan Subbab 11.8.

Pertanyaan Raka

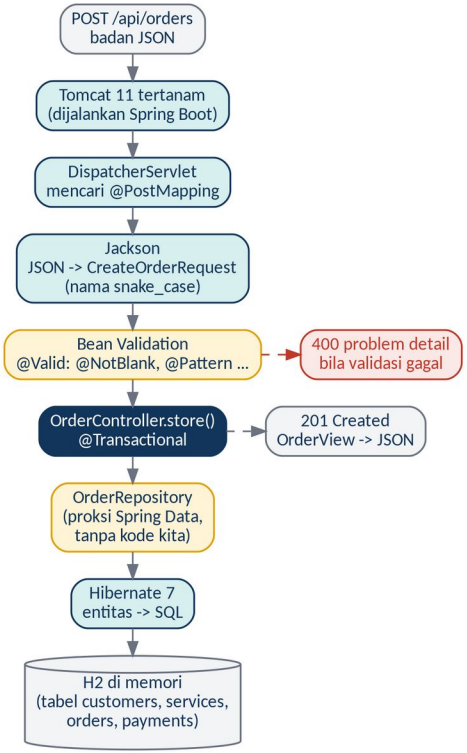
Raka, yang menulis aplikasi ponsel para sopir, belajar informatika dengan Java di sisi *back end*. Membaca API Laravel, ia bertanya: *apakah ini akan tampak sama di Spring Boot?*

Bu Wati punya versinya sendiri: sebuah kantor akuntan yang mengurus pembukuan beberapa *laundry* di Surabaya ingin menghubungkan sistemnya, dan sistem itu ditulis dalam Java.

Empat tabel yang sama, daftar harga yang sama, aturan yang sama (minimal 3 kg, barang utuh, biaya antar-jemput, gratis mulai Rp 100.000, cucian keluar hanya bila lunas), nama JSON yang sama.

Spring Boot 4.0 terbit November 2025, 4.1 pada Juni 2026; buku ini memakai 4.1.1. Ia berjalan di Java 17 ke atas; kita memakai Java 25, JDK dengan dukungan jangka panjang yang berlaku. Ia menanamkan Tomcat 11, dan di bawah akses datanya ada Hibernate ORM 7. Bab 12 kemudian membandingkan kedua API isian demi isian. Buku: Subbab 11.1.

Satu Permintaan Melalui Spring Boot



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.

*Gambar 11.1: Tomcat menerima byte-nya; **DispatcherServlet** adalah front controller Spring MVC yang memilih metodenya; **Jackson** mengubah isi JSON menjadi record; **Bean Validation** memeriksanya; controller memanggil repository **Spring Data**; **Hibernate** menulis SQL-nya. Delapan langkah yang sama dengan Bab 1, dengan nama baru. Buku: Subbab 11.2.4.*

Gagasannya dalam Kata-Kata Sederhana

Application context adalah container DI milik Spring: ia memindai *package* di bawah `@SpringBootApplication`, membuat ***bean*** dari kelas bertanda `@RestController`, `@Service`, `@Repository`, dan menginjeksikannya lewat konstruktor.

Boot menambahkan bawaan yang masuk akal, dinyalakan oleh apa yang Anda pakai: taruh `spring-boot-starter-webmvc` di *build* dan server web muncul; tambahkan H2 dan sumber data muncul. Inilah **konfigurasi otomatis** (*auto-configuration*).

Spring Data JPA menuliskan kelas *repository* untuk Anda dari antarmuka; **Hibernate** menulis SQL-nya.

Empat fondasi Bab 2 lagi: router-nya adalah DispatcherServlet dengan `@GetMapping`, container-nya adalah application context, ORM-nya adalah JPA dengan Hibernate, dan hari ini tidak ada templat karena API menjawab JSON; Thymeleaf datang di Bab 13. Buku: Subbab 11.2.1–11.2.3.

Pekerjaan yang Sama di Laravel dan Spring Boot

Tabel 11.1 Pekerjaan yang sama di Laravel dan di Spring Boot.

Pekerjaan	Laravel	Spring Boot
memulai proyek	`composer create-project`	Spring Initializr
dependensi	`composer.json`	`pom.xml` (Maven)
pengaturan	`.env`, `config/*.php`	`application.properties`
model	model Eloquent	`@Entity` JPA
kueri	`Order::where(...)->get()`	metode *repository*
rute	`routes/api.php`	`@GetMapping`, `@PostMapping`
aturan masukan	*form request*	*record* dengan Jakarta Validation
bentuk JSON	*API resource*	*record* respons (Jackson)
data awal	*seeder*	*bean* `CommandLineRunner`

Tabel 11.1: satu baris berbeda jenisnya. Laravel sejak awal punya migrasi; di sini skemanya berasal dari `ddl-auto` Hibernate, cukup untuk basis data di memori, sedangkan basis data sungguhan kelak butuh Flyway atau Liquibase. Buku: Subbab 11.2.

Membuat Proyeknya

Spring Initializr (start.spring.io, atau ekstensi Spring Boot di VS Code): Maven, Java 25, Spring Boot 4.1.1, dependensi untuk web, JPA, validasi, H2, dan pengujian.

`pom.xml` mewarisi `spring-boot-starter-parent` 4.1.1, yang menetapkan versi teruji untuk setiap pustaka, sehingga lima dependensi kita tidak butuh nomor versi.

`LaundryApplication.java` memakai `@SpringBootApplication` dan `main` yang memanggil `SpringApplication.run`; semua yang lain ditemukan lewat pemindaian.

Setiap kelas harus tinggal di `com.profitina.laundry` atau di bawahnya. Controller di package saudara tidak pernah terpindai, dan URL-nya menjawab 404 tanpa galat apa pun saat aplikasi menyala. Buku: Subbab 11.3.

Pengaturan

Kode 11.1 `application.properties`: basis data di memori, JSON *snake_case*, dan *problem details* untuk galat.

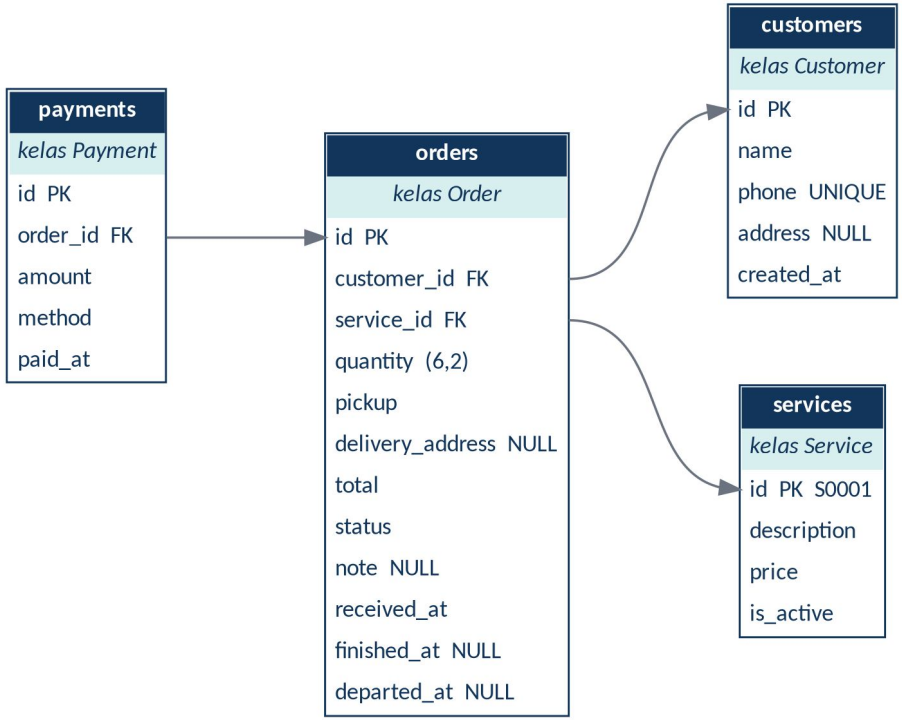
```
spring.application.name=profitina-laundry
server.port=8080

# 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
```

Kode 11.1: H2 di memori dibuat saat menyala dan dibuang saat berhenti. `open-in-view=false` *berarti relasi lazy hanya bisa dibaca di dalam transaksi, yang memaksa kita memuat apa yang dibutuhkan dengan sengaja.* `SNAKE_CASE` *membuat* `customerId` *tampil sebagai* `customer_id`*, seperti di Laravel. Galat keluar sebagai* ***problem details****, bentuk JSON yang baku. Buku: Subbab 11.3.4.*

Domainnya: Empat Entitas



Gambar 11.2 Empat tabel Profitina Laundry dan kelas entitas yang memetakannya; panah mengarah dari *foreign key* ke kunci primer yang dirujuknya.

Gambar 11.2: `Customer`, `Service`, `Order`, `Payment`. Tabel pesanan bernama `orders`, karena `ORDER` adalah kata kunci SQL. `Service` memakai kodenya sendiri seperti `S0001` sebagai kunci, jadi tanpa `@GeneratedValue`; dan namanya bertabrakan dengan anotasi `@Service` milik Spring, sehingga berkas yang memakai keduanya harus menulis salah satunya lengkap, `@org.springframework.stereotype.Service`, seperti di Bab 13. Mempertahankan nama itu sepadan: kata yang sama berarti hal yang sama di ketiga versi toko. Buku: Subbab 11.4.1 dan 11.4.3.

Sebuah Entitas: `Customer`

Kode 11.2 Awal `Customer.java`: seorang pelanggan, dengan nomor telepon yang unik.

```java
@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;

    protected Customer() {                // for JPA only
    }
```

Kode 11.2: `@Entity` memetakan kelasnya, `@Id` dengan `IDENTITY` membiarkan basis data menomori baris, `@Column` menyatakan panjang dan aturan. Hibernate butuh konstruktor tanpa argumen untuk membangun objek dari baris; menjadikannya `protected` mencegah kode kita sendiri memakainya. `Order` menambahkan `@ManyToOne(fetch = LAZY)` ke pelanggan dan layanan serta `@OneToMany` ke pembayaran. Buku: Subbab 11.4.2 dan 11.4.4.

Status Disimpan Seperti di Kerangka Kerja Lain

`EnumType.ORDINAL` menyimpan 0, 1, 2 dan rusak bila ada konstanta yang disisipkan; `EnumType.STRING` menyimpan `QUEUED`, yang tidak diharapkan basis data Laravel.

`OrderStatus` memberi setiap konstanta kata yang disepakati; `OrderStatusConverter`, sebuah `AttributeConverter`, menyimpan `queued`, `finished`, `departed` di setiap kolom status.

Uang bertipe `long` untuk rupiah utuh; jumlah bertipe `BigDecimal`, tidak pernah `double`.

Ketiga aplikasi kini bisa membaca data satu sama lain: kata-kata yang sama, bilangan bulat yang sama. Buku: Subbab 11.4.5.

Aturannya, dalam Java Biasa

```
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
  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.
```

Sebagian keluaran sungguhan `RulesCheck`, dikompilasi dengan `javac -d out -sourcepath src/main/java check/RulesCheck.java` dan dijalankan dengan `java -cp out RulesCheck`. `LaundryRules` memakai aritmetika Bab 8 langkah demi langkah: `movePointRight(2)` mengubah 4,5 menjadi 450 perseratus tanpa pembulatan, `(h + 99) / 100 * 100` membulatkan barang ke atas, menambah 50 sebelum pembagian membulatkan setengah ke atas. Dibandingkan byte demi byte dengan `php tools/core_check.php`, keluarannya identik: tiga kerangka kerja, dua bahasa, satu daftar harga. Buku: Subbab 11.4.6.

Repository: Kueri dari Nama Metode

Kode 11.3 Antarmuka di `OrderRepository.java`: halaman-halaman pesanan, disaring menurut status, dan satu pesanan beserta semua yang dibutuhkannya.

```
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);
}
```

Kode 11.3: tidak ada kelas yang perlu ditulis. `JpaRepository` memberi save, find, delete, dan count; setiap nama metode menjadi kueri: saring menurut status, urutkan menurut tanggal lalu id (urutan yang lengkap, seperti diajarkan Bab 8), satu halaman. `@EntityGraph` memuat relasi yang disebut dalam kueri yang sama, versi JPA dari `Include` dan `with`. Buku: Subbab 11.5.

Record Masuk, *Record* Keluar

Keluar: `OrderView` adalah ***record*** Java yang dibangun dari entitas, padanan `OrderResource` di Laravel; `PageView` menjawab satu halaman sebagai `data` dan `meta`, menghitung halaman mulai 1 seperti Laravel.

Masuk: `CreateOrderRequest` adalah *record* dengan anotasi Jakarta Validation (`@NotNull`, `@NotBlank`, `@DecimalMin`, `@Digits`, `@Size`) dan satu aturan tentang dua isian, metode `@AssertTrue`: antar-jemput butuh alamat.

Jangan pernah menyerialisasi entitas: Jackson bisa mengikuti relasi *lazy* di luar transaksi lalu gagal, atau berputar dari pesanan ke pembayaran ke pesanan lagi.

Record itu pendek, tidak bisa diubah, dan persis selebar JSON-nya. Entitas untuk basis data, record untuk kabel. Ini ketiga kalinya kita bertemu gagasan ini: kelas formulir Bab 6, API resource Bab 10, dan kini record Java. Setiap kerangka kerja mengejanya berbeda; tidak ada yang membiarkan basis data menentukan apa yang dilihat orang asing. Buku: Subbab 11.6.1 dan 11.6.2.

Controller Pesanan

Kode 11.4 Metode `store` di `OrderController.java`: membuat pesanan di dalam transaksi.

```
@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 11.4: `@RestController` dengan `@RequestMapping("/api/orders")`; repository tiba lewat konstruktor. `@Valid @RequestBody` mengikat dan memeriksa JSON; aturan yang dilanggar dijawab 400 dengan isi problem details sebelum metodenya berjalan, sedangkan pelanggan atau layanan yang tidak ada dijawab 422, dan Bab 12 memutuskan kode mana yang benar. Konstruktor `Order` menghargai dirinya sendiri dengan `LaundryRules`, sehingga tidak ada harga yang datang dari permintaan. Controller ini juga mendaftar (`?status=finished&page=2`), menampilkan, dan memajukan pesanan satu langkah dengan `PATCH /api/orders/{id}/status`. Buku: Subbab 11.6.3.

Memuat Daftar Harga, Menjalankan, dan Menguji

`DemoData`, sebuah *bean* `CommandLineRunner`, membaca daftar harga sungguhan dari `Service.csv` dan menambahkan tiga pelanggan Bab 5 dan 7 beserta pesanannya.

Di komputer Anda: `mvn spring-boot:run` di folder `spring` bab ini (atau **Run** di atas `main` di VS Code); eksekusi pertama mengunduh pustakanya. Lalu `curl` daftar harganya dan buat pesanan: 2,5 kg cuci, kering, dan lipat dengan antar-jemput menjawab `total` 34000.

`LaundryRulesTest` menguji aturan harga dan alur kerja dengan JUnit, tanpa menyalakan Spring: cepat, dan bisa dijalankan pada setiap perubahan.

Kasus uji yang sama dengan program pemeriksa Bab 8, kini sebagai pengujian JUnit yang gagal dengan lantang. Buku: Subbab 11.7–11.9.

Menjadikannya Lebih Baik dan Memakainya Sungguhan

Pindah dari H2 di memori ke berkas atau basis data server; ganti `ddl-auto` dengan migrasi **Flyway** sebelum ada data sungguhan.

Untuk kantor akuntan: nama isian yang sama di kedua API berarti satu program klien bisa berbicara dengan versi toko mana pun.

`ddl-auto=update` menambah kolom tetapi tidak pernah menghapus atau mengganti namanya dan tidak bisa memindahkan data: cukup untuk demonstrasi, berbahaya untuk pembukuan toko. Buku: Subbab 11.10 dan 11.11.

Kesalahan yang Sering Terjadi

Kelas di luar *package* aplikasi; tabel bernama `order`; tanpa konstruktor tanpa argumen; relasi *eager* di mana-mana.

Menyerialisasi entitas langsung; *enum* menurut posisi atau nama Java; uang dalam `double`.

Halaman mulai 0 di URL; lupa `@Transactional` untuk relasi *lazy*: `LazyInitializationException`; `ddl-auto=update` untuk basis data sungguhan.

Beberapa di antaranya adalah kesalahan Bab 5 dan Bab 8 dalam baju Java: relasi eager adalah masalah N+1, entitas yang diserialisasi adalah model yang dikembalikan tanpa resource, uang `double` adalah float Bab 8. Buku: Subbab 11.12.

Ringkasan: Lima Hal untuk Diingat

1. Spring Boot 4.1 di Java 25: *starter*, konfigurasi otomatis, dan Tomcat 11 tertanam; proyek dimulai di Spring Initializr (Gambar 11.1).

2. *Application context* memindai *package*, membuat *bean*, dan menginjeksikannya lewat konstruktor.

3. Entitas JPA memetakan empat tabel (Gambar 11.2); relasi *lazy*, tabel `orders`, *converter* untuk kata yang disepakati.

4. *Repository* adalah antarmuka; kueri berasal dari nama metode; `@EntityGraph` memuat relasi bersama kueri utama.

5. Metode `@RestController` membaca JSON ke *record* yang divalidasi dan menjawab dengan *record* sebagai JSON *snake_case*; `LaundryRules` mencetak persis apa yang dicetak pemeriksa PHP Bab 8.

Tiga kerangka kerja kini menjawab pertanyaan yang sama tentang toko yang sama. Bab berikutnya menaruh dua di antaranya berdampingan, permintaan demi permintaan, dan memutuskan di mana keduanya harus sepakat. Buku: Subbab 11.13.

Pertemuan Berikutnya

Bab 12 — Evaluasi dan Pembahasan III. Proyek REST Laravel dan Spring Boot, dibandingkan isian demi isian.

Sebelum itu: kompilasi dan jalankan `RulesCheck` dengan `javac` lalu bandingkan dengan `core_check.php`; di komputer Anda nyalakan API-nya dan buat pesanan dengan `curl`; kerjakan soal latihan tentang pesanan seorang pelanggan.

Saat membandingkan keluaran `RulesCheck` dan `core_check.php`, pakai perkakas pembanding seperti `diff` atau fitur bandingkan berkas di VS Code, bukan mata: satu spasi yang berbeda sudah cukup untuk membuktikan bahwa kedua aturan tidak sama. Buku: Subbab 11.14 dan 11.15.

Kuliah 12 — Evaluasi dan Pembahasan III: Satu REST API di Laravel dan Spring Boot

Mata kuliah: Pemrograman Berbasis Kerangka Kerja

Buku: Irfan Subakti, *Pemrograman Berbasis Kerangka Kerja: ASP.NET Core, Laravel, dan Spring Boot*, Bab 12

Dosen: Irfan Subakti — yifana@gmail.com

Aturan API Laravel di Bab 10 dan API Spring di Bab 11 sama sampai ke rupiahnya; API-nya sendiri tidak. Hari ini kita menulis satu kontrak untuk keduanya, menyelesaikan tujuh soal yang menyelaraskan masing-masing API dengan kontrak itu, dan memeriksa server mana pun terhadapnya dengan sebuah program kecil. Buku: Bab 12, Subbab 12.1–12.14.

Cakupan Pertemuan Ini

1. **Di mana kedua API berbeda** — amplop, waktu, bentuk galat, kode status.
2. **Kontraknya** — *endpoint*, amplop, nama, format, penolakan.
3. **Tujuh soal** — daftar harga, waktu, satu bentuk galat, semua galat sekaligus, memajukan pesanan, pembayaran, laporan harian.
4. **Memeriksa kontrak** — `check_api.php` dan eksekusi sungguhan.
5. **Pelajaran** — apa yang ditetapkan kontrak, dan kesalahan yang dicegahnya.

*Laravel maupun Spring Boot tidak bisa dipasang di tempat buku ini ditulis, tetapi PHP bisa dijalankan. Karena itu bab ini menyertakan **server acuan**, `contract/reference/server.php`, sekitar 240 baris PHP biasa dengan SQLite yang memakai berkas aturan Bab 8, dan keluaran pemeriksa hari ini adalah eksekusi sungguhan terhadapnya. Di komputer Anda, pemeriksa yang sama dijalankan terhadap aplikasi Laravel dan Spring. Buku: Subbab 12.2.*

Posisi Kita Sekarang: Aturan Sama, API Berbeda

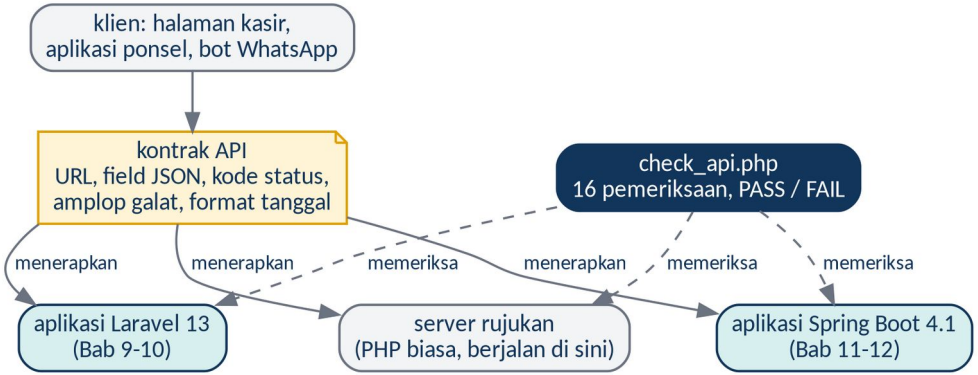
Amplop: *resource* Bab 10 membungkus setiap jawaban dalam `data`; controller Bab 11 mengembalikan objek dan daftar telanjang.

Waktu: bawaan Laravel adalah UTC dengan mikrodetik; Spring mengirim *local date-time* Java.

Galat: Laravel menjawab aturan yang dilanggar dengan 422 dan daftar galat per isian; Spring menjawab 400 dengan isi *problem details* dan nama isian Java.

Aplikasi ponsel Raka harus bekerja dengan server mana pun, dan sistem Java milik kantor akuntan akan membaca server mana pun yang dijalankan Bu Wati. Setiap perbedaan adalah satu baris kode khusus di aplikasi, dan satu tempat bagi bug. Obatnya: putuskan setiap hal sekali, tuliskan, dan buat kedua server menaatinya. Buku: Subbab 12.1.

Satu Kontrak, Tiga Implementasi, Satu Pemeriksa



Gambar 12.1 Satu kontrak, tiga implementasi, dan satu pemeriksa: klien hanya bergantung pada kontrak, dan `check_api.php` memverifikasi setiap server terhadapnya.

*Gambar 12.1: para klien (aplikasi sopir, kantor akuntan) hanya bergantung pada **kontrak**, tidak pernah pada kerangka kerja. Setiap server diperiksa terhadap kontrak oleh program yang sama. Pendekatan untuk setiap soal: nyatakan, kerjakan contoh dengan tangan, bandingkan apa yang dilakukan kedua kerangka kerja hari ini, ubah keduanya sesuai kontrak, cari penyelesaian keliru yang menggoda. Buku: Subbab 12.2.*

Kontraknya: *Endpoint*

Tabel 12.1 *Endpoint* kontrak API Profitina Laundry.

Metode dan URL	Berhasil
`GET /api/services`	200, `data`: layanan aktif, menurut kode
`POST /api/customers`	201, `data`: pelanggannya
`GET /api/orders?status=&page=`	200, `data`: pesanan, `meta`: nomor halaman
`POST /api/orders`	201, `data`: pesanannya
`GET /api/orders/{id}`	200, `data`: pesanan beserta `payments`
`PATCH /api/orders/{id}/status`	200, `data`: pesanannya
`POST /api/orders/{id}/payments`	201, `data`: pesanan beserta `payments`
`GET /api/reports/daily?date=YYYY-MM-DD`	200, `data`: angka-angka hari itu

Tabel 12.1: selain endpoint, kontrak menetapkan amplop `data`, nama snake_case, kata status huruf kecil (`queued`, `finished`, `departed`), rupiah utuh, dan satu format tanggal-waktu dalam waktu toko. Kontrak ini ditulis dengan memilih, untuk setiap hal, pilihan yang paling sedikit mengejutkan klien, dan sering kali itu adalah bawaan Laravel. Tidak ada yang salah dengan itu: yang penting bukan kerangka kerja mana yang menang, melainkan bahwa pilihannya dibuat sekali, dengan sengaja, dan ditulis. Buku: Subbab 12.3.

Kontraknya: Penolakan dan Sisa Tagihan

Tabel 12.2 Penolakan dalam kontrak, dan isi yang dibawa masing-masing.

Status	Kapan	Isi
401	tidak ada token yang sah	`message`
404	pesanan, layanan, atau pelanggan tidak ada	`message`
409	pesanan sudah diambil dan ditutup	`message`
422	masukan melanggar aturan, termasuk langkah status yang ditolak	`message`, `errors`

$$balance = total - \sum_{p \in P} amount_p \tag{12.1}$$

Tabel 12.2: 400 untuk isi yang sama sekali tidak bisa dibaca, 422 untuk isi yang melanggar aturan, 409 untuk resource yang keadaannya melarang permintaan itu. Persamaan (12.1) adalah sisa tagihan Bab 5, kini ditulis ke dalam kontrak agar kedua server menghitungnya dengan cara yang sama. Buku: Subbab 12.3.

Soal 12.1: Daftar Harga untuk Program

Laravel belum punya API untuk daftar harga; Spring punya, tetapi tanpa amplop `data`.

Laravel: `Api/ServiceController` menjawab layanan aktif sebagai koleksi `ServiceResource`. **Spring:** `DataView`, *record* dengan satu isian `data`, membungkus daftar yang sama.

Keliru yang menggoda: mengembalikan modelnya, `Service::all()` atau `services.findAll()`. Ia mengirim `description` dan `is_active` (dan di Laravel cap waktunya), tidak menyertakan `name` dan `unit` yang dihitung, dan mengikat setiap ponsel pada kolom-kolom hari ini.

Hari ketika pemilik memecah deskripsi menjadi dua kolom, setiap klien jawaban berbentuk model akan rusak. Resource atau record menepati janji sementara basis datanya berubah. Buku: Subbab 12.4.

Soal 12.2: Satu Cara Menulis Waktu

Kontraknya: `YYYY-MM-DDTHH:MM:SS`, dalam waktu toko, `Asia/Jakarta`.

`2026-09-25T01:15:00Z` dan `2026-09-25T08:15:00` adalah saat yang sama dengan teks yang berbeda; klien yang membandingkannya sebagai teks akan keliru.

Laravel: format eksplisit di *resource*. **Spring:** `@JsonFormat` dengan pola tetap pada isian waktu di *record*.

Keliru yang menggoda: mengganti zona waktu Laravel menjadi UTC "agar sama dengan Spring". Laporan harian lalu memotong hari pada pukul tujuh pagi waktu Surabaya. Zona waktu dipilih untuk manusia, format untuk program; jangan mengubah yang pertama untuk memperbaiki yang kedua. Buku: Subbab 12.5.

Soal 12.3: Satu Bentuk untuk Setiap Galat

422 dalam kontrak: sebuah `message` dan `errors`, objek dari setiap nama isian, seperti dieja permintaannya, ke daftar pesan. Laravel sudah begitu.

Spring: `ApiExceptionHandler`, sebuah `@RestControllerAdvice`, mengubah galat validasi, aturan yang dilanggar, pengecualian status, dan isi yang tidak terbaca ke dalam bentuk kontrak.

Ia mengubah nama isian ke *snake_case*: `serviceId` menjadi `service_id`, sehingga klien bisa menaruh pesannya di samping isian yang tepat.

Keliru yang menggoda: mengandalkan pengaturan snake_case Jackson dari Bab 11 dan berharap nama galat ikut. Tidak ikut: nama di `FieldError` berasal dari validator, bukan dari Jackson, dan tetap `serviceId`. Konversinya harus terjadi di tempat isi galat dibangun. Buku: Subbab 12.6.

Penangan untuk Galat Validasi

Kode 12.1 Satu metode `ApiExceptionHandler.java` (Spring): galat validasi dalam bentuk kontrak.

```
@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);
}
```

Kode 12.1: kelas ini memakai `@RestControllerAdvice`, sehingga metodenya menangani pengecualian yang dilempar oleh setiap controller. Metode ini mengumpulkan setiap `FieldError` di bawah namanya dalam snake_case, beberapa pesan per isian bila perlu, dan `unprocessable` membangun isi `{"message": "The given data was invalid.", "errors": {...}}` dengan status 422, kata demi kata sama dengan yang dikirim Laravel. Metode-metode saudaranya menjawab aturan yang dilanggar dengan bentuk yang sama, `ResponseStatusException` dengan kodenya dan sebuah `message`, dan isi yang tidak terbaca dengan 400. Satu kelas, dan setiap controller aplikasi berbicara bahasa galat kontrak. Buku: Subbab 12.6.4.

Soal 12.4: Semua Galat Sekaligus

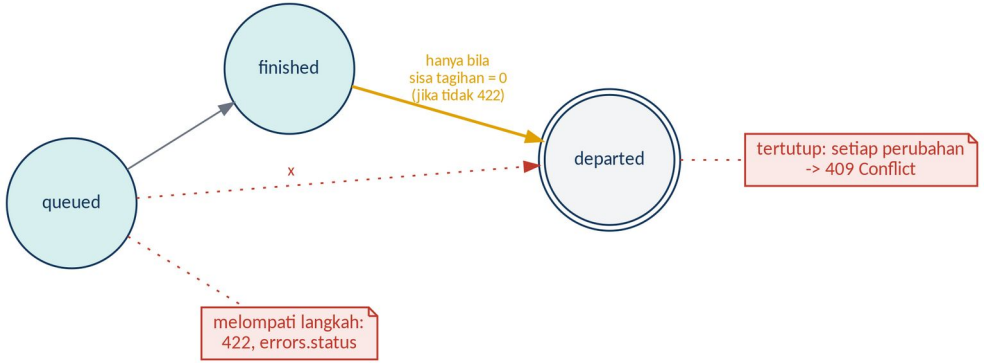
`@Valid` di Spring memeriksa aturan satu isian lebih dulu; pemeriksaan basis data (apakah pelanggannya ada? apakah layanannya ditawarkan?) datang di putaran kedua.

Klien yang memperbaiki tiga galat yang ditunjukkan lalu menerima galat keempat merasa dicurangi.

Spring: `CreateOrderRequest` hanya menyimpan aturan satu isian; controller memvalidasi sendiri, menambahkan aturan dua isian dan pemeriksaan basis data ke **peta yang sama**, lalu menjawab sekali.

Keliru yang menggoda: mengganti nama metode `@AssertTrue` agar propertinya menjadi `deliveryAddress`. Ia bertabrakan dengan komponen yang sesungguhnya dan tetap meninggalkan pemeriksaan basis data di putaran kedua. Aturan yang perlu melihat segalanya tempatnya di kode yang bisa melihat segalanya. Buku: Subbab 12.7.

Soal 12.5: Memajukan Pesanan, dan Pesanan yang Ditutup



Gambar 12.2 Alur kerja pesanan sebagaimana dinyatakan kontrak: satu langkah setiap kali, diambil hanya bila sisa tagihan 0, dan pesanan yang sudah diambil tertutup bagi setiap perubahan.

*Gambar 12.2: langkah yang ditolak, seperti `queued` ke `departed` atau diambil selagi masih berutang, adalah **422** dengan `errors.status`: permintaannya melanggar aturan. Perubahan pada pesanan yang sudah diambil adalah **409**: keadaan resource melarang perubahan apa pun. Klien menyebut status yang diinginkan, sehingga permintaan aman diulang. Keliru yang menggoda: "maju" tanpa nama, seperti tombol web; dua sopir yang menekan "diambil" bersamaan bisa memajukan pesanan dua langkah. Buku: Subbab 12.8.*

Soal 12.6: Pembayaran

`POST /api/orders/{id}/payments` dengan `amount` dan `method`; 201 dengan pesanan beserta pembayarannya; melebihi sisa tagihan adalah 422 dengan `errors.amount`.

Spring: `CreatePaymentRequest` dan `PaymentView`, dengan waktunya dalam format kontrak; setiap pembayaran langsung disimpan dengan `payments.save`.

Keliru yang menggoda: mengandalkan *cascade* saat *commit*. Basis datanya benar, tetapi jawaban ditulis *sebelum commit*, dan klien menerima `"id": null`.

Orang yang mengeklik aplikasi tidak pernah melihat bug itu; pengujian yang memeriksa isian yang dipakai klien langsung menangkapnya. Buku: Subbab 12.9.

Soal 12.7: Laporan Harian

$$\sum_{s \in S} n_s = N \tag{12.2}$$

`GET /api/reports/daily?date=…`: hitungan per status, total dan pembayaran hari itu, pada hari setengah-terbuka $[d, d+1)$ dalam waktu toko.

Setiap status hadir, bahkan dengan nol pesanan, sehingga hitungan per status berjumlah sama dengan banyaknya pesanan, Persamaan (12.2), dan tidak ada klien yang harus menebak kunci yang hilang.

Laravel: `Api/ReportController`. **Spring:** dua kueri *repository* baru dan `LinkedHashMap` yang mendaftar setiap status secara berurutan.

Keliru yang menggoda: `WHERE date(received_at) = ?`*. Enak dibaca, tetapi menerapkan fungsi pada kolom, sehingga basis data tidak bisa memakai indeks, dan sebagian basis data menafsirkan waktu tersimpan menurut zona server. Interval* `received_at >= :from AND received_at < :to` *cepat dan tepat. Buku: Subbab 12.10.*

Memeriksa Kontrak: Eksekusi Sungguhan

```
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  finished -> departed before payment: 422, errors.status
PASS  paying more than the balance -> 422, errors.amount
PASS  a departed order is closed: 409 with a message
PASS  GET /api/reports/daily -> 200, the counts add up

16 passed, 0 failed
```

Sebelas dari enam belas baris eksekusi sungguhan `contract/check_api.php` terhadap server acuan. Pemeriksa menerima URL dasar, sehingga program yang sama memeriksa server Laravel dan server Spring di komputer Anda. Ini gagasan "program pemeriksa" Bab 8, satu tingkat lebih tinggi: bukan satu aturan, melainkan seluruh API. Buku: Subbab 12.11.1 dan 12.11.2.

Pengujian di Dalam Setiap Kerangka Kerja

Laravel: `ContractTest` memeriksa hal-hal baru dalam kontrak dengan jam yang dibekukan, agar waktu dan laporan harian bisa diramalkan.

Spring: `OrderApiTest` dengan MockMvc: aturan harga, semua galat sekaligus, seluruh riwayat sebuah pesanan, dan laporannya.

Uji isian yang dipakai klien, bukan hanya kode status: jawaban dengan status yang benar dan `"id": null` tetap merusak aplikasi.

Pemeriksa membuktikan bahwa server yang berjalan menepati kontrak; pengujian di dalam setiap kerangka kerja membuktikannya pada setiap perubahan, sebelum apa pun berjalan di server. Buku: Subbab 12.11.3.

Memakainya Sungguhan

Tuliskan kontraknya sebelum klien kedua datang: kantor akuntan, aplikasi ponsel baru, gerai mitra.

Terbitkan di tempat yang bisa dibaca klien; beri versi bila harus berubah; simpan pemeriksanya di repositori dan jalankan sebelum setiap rilis.

Kontrak juga janji kepada orang-orang yang membangun di atas sistem Anda: ia membuat mereka bisa bekerja tanpa bertanya kepada Anda apa yang dilakukan server hari ini. Buku: Subbab 12.12.

Kesalahan yang Sering Terjadi

Membiarkan kerangka kerja memilih kontrak; membandingkan waktu sebagai teks lintas zona; mencampuradukkan 400, 409, dan 422.

Kunci galat dengan nama milik server; galat dalam dua putaran; kunci yang hilang untuk nol; hanya memeriksa kode status dalam pengujian.

Setiap kesalahan ini tidak terlihat di dalam satu kerangka kerja dan langsung jelas begitu server kedua harus menjawab klien yang sama. Itulah nilai membangun API yang sama dua kali. Buku: Subbab 12.13.

Ringkasan: Lima Hal untuk Diingat

1. Kontrak menetapkan *endpoint* (Tabel 12.1), amplop, nama, kata, uang, format waktu, dan penolakan (Tabel 12.2); klien hanya bergantung padanya (Gambar 12.1).

2. *Resource* dan *record*, bukan model; satu format waktu dalam waktu toko; zona waktu untuk manusia, format untuk program.

3. Satu bentuk galat dengan nama isian milik permintaan; semua galat sekaligus.

4. 422 untuk langkah yang ditolak, 409 untuk pesanan yang ditutup (Gambar 12.2); simpan pembayaran sebelum menjawab; setiap status hadir di laporan.

5. `check_api.php` memeriksa server acuan: 16 lolos, 0 gagal; setiap kerangka kerja punya pengujiannya sendiri untuk hal yang sama.

Aplikasi ponsel tidak lagi perlu tahu sedang berbicara dengan server yang mana. Itulah inti bab ini, dan batu uji bagi setiap API yang akan dipakai lebih dari satu program. Buku: Subbab 12.14.

Pertemuan Berikutnya

Bab 13 — Spring Boot II. Arsitektur berlapis, validasi, dan halaman CRUD lengkap dengan Thymeleaf.

Sebelum itu: jalankan server acuan dan pemeriksanya; di komputer Anda, jalankan pemeriksa terhadap server Laravel dan server Spring; kerjakan soal latihan bab ini.

Saat menjalankan pemeriksa terhadap server Anda sendiri, baca kegagalan pertama lebih dulu: sering kali satu penyebab, misalnya nama isian yang berbeda atau kode status 400 alih-alih 422, menjatuhkan beberapa pemeriksaan berikutnya. Perbaiki di server, jalankan lagi, dan ulangi sampai semuanya lolos. Buku: Subbab 12.15 dan 12.16.

Kuliah 13 — Spring Boot II: Arsitektur Berlapis, Validasi, dan CRUD Lengkap dengan Thymeleaf

Mata kuliah: Pemrograman Berbasis Kerangka Kerja

Buku: Irfan Subakti, *Pemrograman Berbasis Kerangka Kerja: ASP.NET Core, Laravel, dan Spring Boot*, Bab 13

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini aplikasi Spring Boot mendapat halaman untuk meja kasir, dan agar tidak perlu menyalin aturan, kita membaginya menjadi lapisan-lapisan: controller yang berbicara HTTP, service yang mengerjakan pekerjaan toko di dalam transaksi, repository yang berbicara SQL. API JSON dan halaman Thymeleaf yang baru memanggil service yang sama. Buku: Bab 13, Subbab 13.1–13.15.

Cakupan Pertemuan Ini

1. **Satu kelas yang mengerjakan segalanya** — mengapa controller Bab 12 tidak bisa begitu saja mendapat halaman.
2. **Tiga lapisan, satu arah** — apa yang boleh diketahui setiap lapisan; transaksi.
3. **Lapisan *service*** — tiga pengecualian, `CustomerService`, `OrderService`, laporan.
4. **Formulir dan halaman** — *record* formulir dengan Bean Validation, templat Thymeleaf, Post/Redirect/Get.
5. **Galat, SQLite, pengujian** — satu penasihat galat per lapisan, basis data berupa berkas, *service* diuji dengan Mockito.

Yang dijalankan: mesin tempat buku ini ditulis tidak bisa mengunduh pustaka Spring. Setiap berkas Java dikompilasi terhadap pengganti kecil untuk tipe-tipe Spring, Jakarta, dan JUnit yang dipakainya, tanpa galat, dan aturannya dikompilasi serta dijalankan sungguhan dengan `javac`; keluarannya sama dengan di Bab 8 dan 11. Anda menjalankan aplikasinya di komputer sendiri. Buku: Subbab 13.9.

Satu Kelas yang Mengerjakan Segalanya

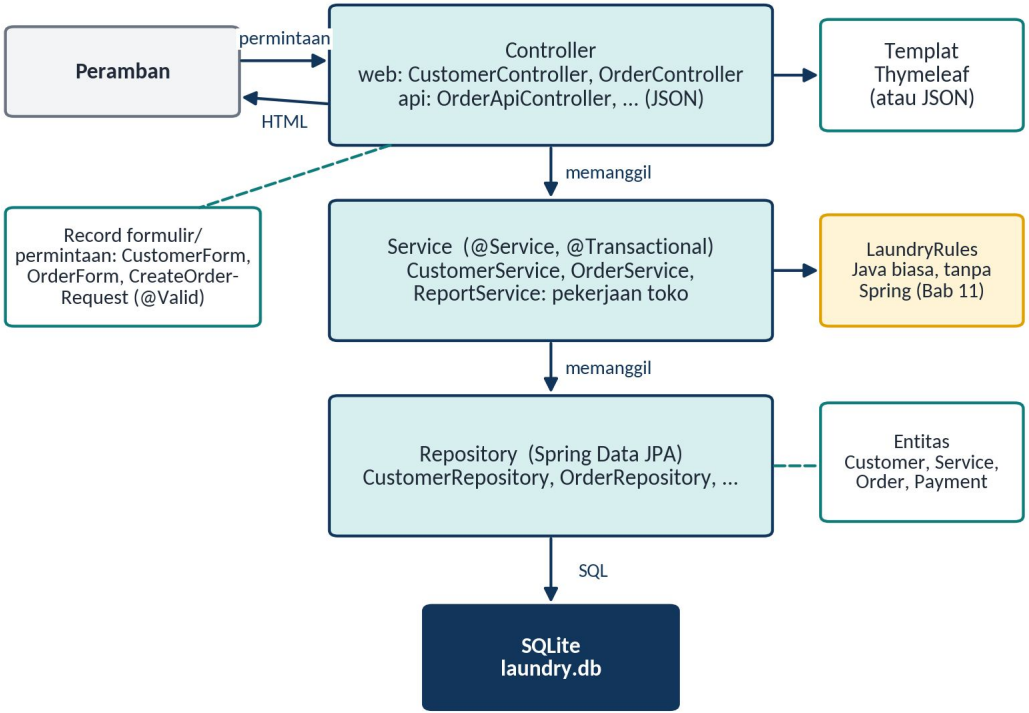
`OrderController.store` di Bab 12 memvalidasi permintaan, mencari pelanggan dan layanan, memeriksa alamat antar-jemput, mengumpulkan galat, membangun pesanan, menyimpannya, dan membungkusnya dalam JSON: semuanya dalam satu metode.

Kini meja kasir menginginkan **halaman**: antrean, formulir untuk kantong baru, halaman pesanan dengan tombol langkah berikutnya dan formulir pembayaran.

Salin logikanya ke controller halaman, maka setiap aturan punya dua tempat. Hari ketika satu berubah dan yang lain tidak, toko kembali punya dua harga.

Ini adalah keinginan pertama Bu Wati di Bab 1, “satu perubahan di satu tempat”, kini di dalam satu kerangka kerja. Buku: Subbab 13.1 dan 13.2.1.

Tiga Lapisan, Satu Arah



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.

*Gambar 13.1: panggilan **hanya ke bawah**. Controller memanggil service; service memanggil repository dan `LaundryRules`; tidak ada lapisan bawah yang tahu apa pun tentang lapisan di atasnya. Satu arah itulah yang membuat dua jenis controller bisa berbagi satu himpunan service. Buku: Subbab 13.2.2.*

Apa yang Boleh Diketahui Setiap Lapisan

Tabel 13.1 Apa yang boleh diketahui setiap lapisan, dan apa yang tidak boleh.

Lapisan	Tahu tentang	Tidak boleh tahu tentang	Contoh
Controller	URL, parameter, JSON, formulir, pengalihan, templat	SQL, transaksi, harga	`OrderController.advance`
Service	operasi dan aturan toko, transaksi	HTML, URL, kode status	`OrderService.advance`
Repository	tabel, kueri	aturan, HTTP	`OrderRepository.billedBetween`
Aturan	angka, status	segala yang lain	`LaundryRules.advance`

*Tabel 13.1: uji cepat untuk baris kode mana pun: di kolom mana ia berada? Harga di controller, kode status HTTP di service, aturan di repository: masing-masing berada di tempat yang salah. **Transaksi** milik service: `@Transactional` mengelompokkan operasi satu metode service agar berhasil atau gagal bersama, terpisah dari transaksi lain: dua kasir yang menerima Rp 34.000 terakhir untuk satu pesanan pada saat yang sama tidak boleh sama-sama berhasil. Buku: Subbab 13.2.3 dan 13.2.4.*

Tempat Kodenya

`domain` — entitas dan `LaundryRules`; `repository` — antarmuka Spring Data; `service` — *service* dan pengecualiannya.

`api` — controller JSON Bab 12, dipindahkan dan dirampingkan; `web` — controller halaman yang baru dan *record* formulir.

`web.OrderController` dan `api.OrderController` sama-sama akan menjadi *bean* `orderController` dan aplikasi tidak mau menyala: yang API bernama `OrderApiController`.

Package membuat lapisan terlihat di pohon berkas, dan memungkinkan setiap penasihat galat dibatasi pada package-nya sendiri, seperti yang segera kita lihat. Buku: Subbab 13.3.

Lapisan *Service*: Tiga Pengecualian

`RuleException` — masukan yang melanggar aturan, dengan pesan per isian: 422 di API, pesan di samping isian di halaman.

`NotFoundException` — catatan yang tidak ada: 404.

`ConflictException` — permintaan yang tidak diizinkan keadaan catatan saat ini, misalnya membayar pesanan yang sudah diambil: 409.

Service mengatakan apa yang salah dengan kata-kata toko; setiap jenis controller memutuskan bagaimana menyampaikannya kepada pendengarnya. Dengan begitulah service tetap bebas dari HTTP. Pola yang sama muncul di ASP.NET Core pada Bab 16 dan, dalam bentuk yang lebih sederhana, di Laravel lewat `abort(404)` dan pengecualian validasi. Tiga pengecualian bernama itu adalah kosakata bersama antara service dan setiap jenis controller yang memanggilnya. Buku: Subbab 13.4.1.

`OrderService`: Setiap Aturan di Satu Tempat

Kode 13.1 Dua metode `OrderService.java`: menerima uang, dan menolak pesanan yang sudah ditutup.

```
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;
}

/** Laundry that has left the shop takes no more steps or money. */
private static void ensureOpen(Order order) {
    if (order.getStatus() == OrderStatus.DEPARTED) {
        throw new ConflictException("Order #" + order.getId()
                + " has left the shop and is closed.");
    }
}
```

Kode 13.1: kelas ini memakai `@Transactional`, sehingga setiap metode publiknya berjalan dalam satu transaksi. `create` lebih dulu memanggil `check`, yang mengembalikan setiap aturan basis data sebagai peta agar API dan halaman bisa melaporkan semua masalah sekaligus, lalu memeriksa lagi dan menyimpan. `advance` menanyakan langkah berikutnya kepada `LaundryRules`. `pay` menolak jumlah di

atas sisa tagihan dan langsung menyimpan pembayaran, pelajaran Soal 12.6. `now()` *membaca bean* `Clock` *yang diatur ke Waktu Indonesia Barat. Buku: Subbab 13.4.3.*

Aturan Uang, Tidak Berubah

$$S = \left\lfloor \frac{p \cdot b + 50}{100} \right\rfloor, \qquad T = S + \begin{cases} 10000 & \text{pickup and } S < 100000 \\ 0 & \text{otherwise} \end{cases} \tag{13.1}$$

$$B = T - P \tag{13.2}$$

Persamaan (13.1) adalah total Bab 8: subtotal S dari harga p dan perseratus tertagih b, ditambah biaya antar-jemput di bawah Rp 100.000. Persamaan (13.2) adalah sisa tagihan: total T dikurangi pembayaran P. Lapisan-lapisan memindahkan kode; tidak ada satu rupiah pun yang berubah. `ReportService` *menghitung angka-angka satu hari untuk API maupun halaman. Buku: Subbab 13.4.3 dan 13.4.4.*

Controller API yang Ramping

`OrderApiController`: baca JSON ke *record* permintaan, panggil `OrderService`, tulis jawabannya sebagai *record*. Tiga baris per metode.

`ApiExceptionHandler` mengubah ketiga pengecualian dan galat validasi ke bentuk kontrak: 422 dengan `errors`, 404 dan 409 dengan `message`.

Ia adalah `@RestControllerAdvice` yang dibatasi dengan `basePackages` ke `api`; tanpa batas itu, ia akan menjawab galat halaman dengan JSON.

Kontrak API dari Bab 12 tidak berubah; yang berubah hanya tempat kodenya, dan pemeriksa yang sama bisa memastikannya di komputer Anda. Buku: Subbab 13.5.

Memvalidasi Masukan Formulir

Batasan **Jakarta Bean Validation** pada ***record* formulir**: `@NotBlank`, `@Size`, `@Pattern`, `@NotNull`, `@DecimalMin`, `@Digits`, `@Min`.

`CustomerForm` punya batas yang sama dengan permintaan API; `OrderForm` tahu cara menjadi `NewOrder` untuk *service*; `PaymentForm` berisi jumlah dan metode.

Record formulir hanya memuat isian yang boleh diubah formulir itu, sehingga permintaan rekaan tidak bisa mengisi `id` atau `createdAt`: tanpa *mass assignment*.

`Forms.reject` mengubah pesan `RuleException` menjadi galat pada isian formulir yang disebut namanya, sehingga "nomor telepon ini sudah terdaftar" muncul tepat di bawah isian telepon. Buku: Subbab 13.6.

Halaman Pelanggan

Kode 13.2 Metode `create` di `web/CustomerController.java`: pelanggan baru dari sebuah 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";
    }
}
```

*Kode 13.2: `@Validated` memeriksa record formulir; `BindingResult` harus berada tepat sesudahnya, atau Spring menampilkan halaman galat alih-alih formulir. Galat batasan menampilkan formulir lagi; galat aturan dari service ditambahkan ke `BindingResult` yang sama; keberhasilan dialihkan dengan **flash attribute** yang bertahan tepat satu pengalihan. Controller halaman tidak pernah berisi aturan. Buku: Subbab 13.7.1.*

Post/Redirect/Get



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.

Tabel 13.2 URL halaman-halaman pelanggan.

URL	Mengerjakan	Menjawab
`GET /customers?q=...`	daftar dan pencarian	halaman daftar
`GET /customers/new`	menampilkan formulir kosong	halaman formulir
`POST /customers`	membuat	pengalihan, atau formulir dengan galat
`GET /customers/{id}/edit`	menampilkan formulir terisi	halaman formulir
`POST /customers/{id}`	mengubah	pengalihan, atau formulir dengan galat
`POST /customers/{id}/delete`	menghapus	pengalihan dengan pesan

Gambar 13.2 dan Tabel 13.2: formulir HTML hanya mengirim `GET` dan `POST`, sehingga ubah dan hapus adalah `POST` ke URL-nya masing-masing. Setelah `POST` yang berhasil, peramban dialihkan; menyegarkan halaman mengulang `GET` yang tidak berbahaya, tidak pernah pesanannya. Buku: Subbab 13.7.2.

Templat Thymeleaf

`fragments.html` menyimpan *head* halaman, navigasi, dan pesan *flash*; halaman menyertakannya dengan `th:replace="~{fragments :: nav}"`.

`th:object="${form}"` mengikat formulir; `th:field="*{phone}"` menulis nama, id, dan nilainya; `th:errors="*{phone}"` menampilkan pesannya; `th:errorclass="invalid"` menandai isiannya.

`th:text` meng-*escape* secara bawaan, seperti `{{ }}` di Blade dan `@` di Razor.

Halaman pesanan mengikuti pola yang sama: antrean yang disaring menurut status dengan halaman-halaman, formulir untuk menerima kantong, dan halaman pesanan dengan total, pembayaran, sisa tagihan, dan tombol langkah berikutnya. Menyembunyikan tombol yang tidak berlaku hanyalah kerapian; service tetap memeriksa aturannya. Buku: Subbab 13.7.3 dan 13.7.4.

Satu Formulir untuk Menambah dan Mengubah

Kode 13.3 Sebagian `customers/form.html`: satu formulir untuk menambah dan mengubah, dengan pesan di samping setiap isian.

```
<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>
```

Kode 13.3: templat yang sama melayani kedua halaman. Bila controller menaruh `customerId` di model, formulir dikirim ke `/customers/{id}` dan judulnya "Edit customer"; bila tidak, dikirim ke `/customers`. `@{...}` membangun URL relatif terhadap aplikasi, sehingga situs tetap berjalan bila kelak dipasang di bawah sub-jalur. Pesan yang bukan milik satu isian, seperti aturan tentang dua isian, muncul di atas lewat `#fields.globalErrors()`. Bandingkan dengan `asp-for` di Razor dan `old()` bersama `@error` di Blade: tiga ejaan, satu gagasan. Buku: Subbab 13.7.3.

Satu Tempat untuk Galat, SQLite, dan Pengujian

`WebErrorAdvice`, dibatasi ke `web`, mengubah setiap `NotFoundException` di halaman menjadi `error/404.html` dengan status 404.

Data kini tinggal di **SQLite**, berkas di samping proyek yang bertahan di antara penyalaan; `DemoData` mengisinya hanya pada penyalaan pertama. Pengujian memakai basis data H2 baru di memori.

`OrderServiceTest` menguji lapisan *service* dengan *repository* tiruan dari **Mockito** dan jam yang dibekukan: tanpa basis data, tanpa web, cepat.

`Clock` yang dibekukan membuat "diterima hari ini" dan laporan harian bisa diramalkan dalam pengujian, padanan Java untuk jam beku Laravel di Bab 12. Buku: Subbab 13.8–13.10.

Menjadikannya Lebih Baik dan Memakainya Sungguhan

Terus tambahkan fitur dengan cara yang sama: metode *service* lebih dulu, lalu controller halaman dan metode API yang memanggilnya.

Untuk usaha Anda sendiri: halaman meja layanan toko, klinik, atau bengkel mana pun mengikuti bentuk ini; hanya entitas dan aturannya yang berubah.

Bila seorang rekan bertanya di mana sebuah aturan tinggal, jawabannya selalu “di sebuah service”, dan jawaban itu lebih berharga daripada fitur tunggal mana pun. Buku: Subbab 13.11–13.13.

Kesalahan yang Sering Terjadi

Aturan di controller; transaksi di controller; dua *bean* dengan nama yang sama; satu penasihat galat untuk segalanya.

Mengikat formulir ke entitas; `BindingResult` di tempat yang salah; halaman sebagai jawaban atas `POST` yang berhasil.

`LazyInitializationException` di templat; `@Transactional` pada metode privat atau yang dipanggil dari kelasnya sendiri (*proxy* Spring tidak pernah melihatnya)**; memercayai tombol yang disembunyikan.**

Kesalahan proxy mengejutkan bahkan pengembang berpengalaman: `@Transactional` bekerja karena Spring membungkus bean dengan proxy, dan panggilan dari satu metode kelas ke metode lain di kelas yang sama tidak pernah melewati proxy itu. Jadikan metode bertransaksi publik dan panggil dari bean lain, biasanya controller. Buku: Subbab 13.14.

Ringkasan: Lima Hal untuk Diingat

1. Controller berbicara HTTP, *service* bekerja di dalam transaksi, *repository* berbicara SQL; panggilan hanya ke bawah (Gambar 13.1, Tabel 13.1).

2. Controller API dan halaman memanggil *service* yang sama; *service* melempar `RuleException`, `NotFoundException`, `ConflictException`.

3. Aturan uang tidak berubah (Persamaan (13.1), Persamaan (13.2)).

4. *Record* formulir dengan Bean Validation, `BindingResult`, `th:object` dan `th:field` Thymeleaf, serta Post/Redirect/Get (Gambar 13.2).

5. Satu penasihat galat per lapisan; SQLite untuk data, H2 untuk pengujian; *service* diuji dengan Mockito dan jam yang dibekukan.

Bandingkan dengan dua kerangka kerja lainnya: ASP.NET Core menaruh aturan harga di `PriceCalculator` sejak Bab 3, dan Laravel menumbuhkan kelas aksi di Bab 10. Lapisan service Spring adalah gagasan yang sama yang dijadikan tulang punggung aplikasi. Buku: Subbab 13.15.

Pertemuan Berikutnya

Bab 14 — Spring Boot III. Spring Security, rancangan REST API, dan pengujian otomatis.

Sebelum itu: jalankan `RulesCheck` lagi; di komputer Anda jalankan aplikasinya, tambahkan pelanggan dua kali dengan satu nomor telepon, dan ikuti sebuah pesanan dari formulir sampai diambil.

Saat mengikuti sebuah pesanan dari formulir sampai diambil, perhatikan bahwa setiap langkah melewati `OrderService` yang sama dengan yang dipakai API: coba lakukan sebagian langkah lewat halaman dan sebagian lewat `curl`, dan pastikan hasilnya sama. Sejauh ini siapa pun yang bisa menjangkau aplikasi Spring bisa memakai setiap halaman dan setiap endpoint, persis pintu terbuka di Bab 6 dan 10. Bab 14 menutupnya. Buku: Subbab 13.16 dan 13.17.

Kuliah 14 — Spring Boot III: Spring Security, Desain REST API, dan Pengujian Otomatis

Mata kuliah: Pemrograman Berbasis Kerangka Kerja

Buku: Irfan Subakti, *Pemrograman Berbasis Kerangka Kerja: ASP.NET Core, Laravel, dan Spring Boot*, Bab 14

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini aplikasi Spring Boot mendapat pintu-pintunya: akun staf dengan kata sandi ter-hash, token bergaya Sanctum untuk API, dua rantai filter keamanan (satu untuk API, satu untuk halaman), lapis pertahanan kedua di service, tinjauan atas desain REST API-nya, dan pengujian otomatis di empat tingkat. Buku: Bab 14, Subbab 14.1–14.12.

Cakupan Pertemuan Ini

1. **Pintu yang belum ada** — apa yang sudah ditolak API Laravel, dan dua keinginan Bu Wati.
2. **Akun dan kata sandi** — jangan pernah menyimpan kata sandi; staf dan peran; cara Spring Security melihat pengguna.
3. **Token untuk API** — tabel token, *service* token, sebuah filter, masuk dan keluar.
4. **Dua rantai filter** — API tanpa status, sesi dan CSRF untuk halaman, ditelusuri permintaan demi permintaan.
5. **Desain REST dan pengujian** — metode, kode status, amplop; piramida pengujian dalam empat tingkat.

Seperti di Bab 11 dan 13, pustaka Spring tidak bisa diunduh di tempat buku ini ditulis, sehingga kodenya ditulis berdasarkan dokumentasi Spring Security dan Spring Boot. Di komputer Anda, `mvn test` *menjalankan pengujian bab ini. Buku: Bab 14.*

Posisi Kita Sekarang: Pintu yang Belum Ada

Bab 12 membuat API Spring berbicara JSON Laravel; Bab 13 menaruh *service* di bawahnya dan halaman di atasnya. Satu bagian sengaja ditinggalkan: pintu-pintunya.

API Laravel menjawab permintaan tanpa token yang sah dengan `401 {"message": "Unauthenticated."}`; aplikasi para sopir masuk dengan `POST /api/tokens`.

Keinginan Bu Wati: kasir login dengan akun masing-masing, agar toko kelak bisa melihat siapa menerima pesanan mana; menghapus pelanggan, yang menghapus data seseorang, hanya untuk dirinya.

Semua hal lain yang dikerjakan kasir hari ini tetap boleh dikerjakannya. Keamanan yang baik berangkat dari apa yang perlu dikerjakan orang, bukan dari mengunci segalanya. Buku: Subbab 14.1.

Jangan Pernah Menyimpan Kata Sandi

$$N = 2^c \tag{14.1}$$

Simpan ***hash* bergaram yang sengaja dibuat lambat**. bcrypt dengan biaya c melakukan N putaran penyiapan kuncinya yang mahal, Persamaan (14.1), sehingga setiap kenaikan c menggandakan kerja bagi server maupun bagi penyerang.

`BCryptPasswordEncoder` milik Spring memakai $c = 10$ secara bawaan; bawaan Laravel 12; ASP.NET Core Identity memakai PBKDF2 dengan HMAC-SHA512 dan 100.000 iterasi.

`PasswordEncoderFactories.createDelegatingPasswordEncoder()` menuliskan nama algoritmanya ke dalam nilai yang disimpan, sehingga toko kelak bisa pindah ke algoritma yang lebih kuat.

Biaya yang membuat satu login memakan beberapa puluh milidetik tidak terasa oleh kasir dan membuat penebakan menjadi mahal. Hash cepat seperti SHA-256 atas kata sandi manusia bisa dibalik dengan menebak; jangan pernah memakainya untuk kata sandi. Buku: Subbab 14.3.1.

Staf dan Peran

`Role`: `OWNER` dan `CASHIER`, dua peran sudah cukup bagi toko. `Staff`: orang yang boleh login, dengan surel, *hash* kata sandi, dan peran.

`StaffDetailsService` mengubah surel menjadi pandangan Spring Security tentang pengguna: nama, *hash*, dan peran dalam ejaan Spring, seperti `ROLE_CASHIER`.

`DemoStaffSeeder` membuat pemilik dan satu kasir, dengan kata sandi ter-*hash* dan alamat berakhiran `.example`, domain yang dicadangkan untuk dokumentasi.

Kata sandi demo tercetak di buku dan README: hal pertama yang dilakukan toko sungguhan adalah menggantinya. Perhatikan bahwa peran hanya dua. Setiap peran tambahan adalah satu kolom lagi di tabel siapa-boleh-apa, dan satu tempat lagi untuk kesalahan; mulailah sesedikit mungkin dan tambah hanya bila toko benar-benar membutuhkannya. Buku: Subbab 14.3.2 dan 14.3.3.

Token untuk API

$$H = 30 \times 8 = 240 \text{ bits} \tag{14.2}$$

Token bergaya Sanctum berbentuk `id|secret`. Rahasianya 30 byte acak, Persamaan (14.2) bit: jauh di luar jangkauan tebakan.

`TokenService` menerbitkan, mencari, dan mencabut token; tabel `ApiToken` hanya menyimpan **SHA-256** dari rahasianya, dibandingkan dalam waktu konstan.

`TokenApiController`: `POST /api/tokens` memberi token (ditampilkan sekali), `DELETE /api/tokens/current` mengambilnya kembali.

Mengapa hash cepat boleh di sini tetapi tidak untuk kata sandi? Karena rahasianya acak dengan 240 bit, bukan kata yang dipilih manusia: tidak ada yang bisa ditebak. Bagian `id` langsung menemukan barisnya, lalu hash rahasianya dibandingkan. Rancangannya sama dengan Laravel Sanctum, sehingga aplikasi ponsel masuk dengan cara yang sama ke keduanya. Buku: Subbab 14.4.1, 14.4.2, dan 14.4.4.

Filternya

`BearerTokenFilter` membaca `Authorization: Bearer id|secret`, menanyakan stafnya kepada `TokenService`, dan menaruh identitas yang terautentikasi ke dalam konteks keamanan.

Tanpa *header* atau dengan token yang salah: ia sekadar meneruskan permintaan tanpa identitas, dan aturan rantai menjawab 401.

Ia dibuat dengan `new` di dalam rantai API, tidak didaftarkan sebagai `@Component`; filter yang menjadi *component* akan berjalan untuk setiap permintaan, di luar rantainya.

Filter adalah middleware Spring Security: bawang Bab 3 dan Bab 9 lagi, kini dengan identitas sebagai urusannya. Buku: Subbab 14.4.3.

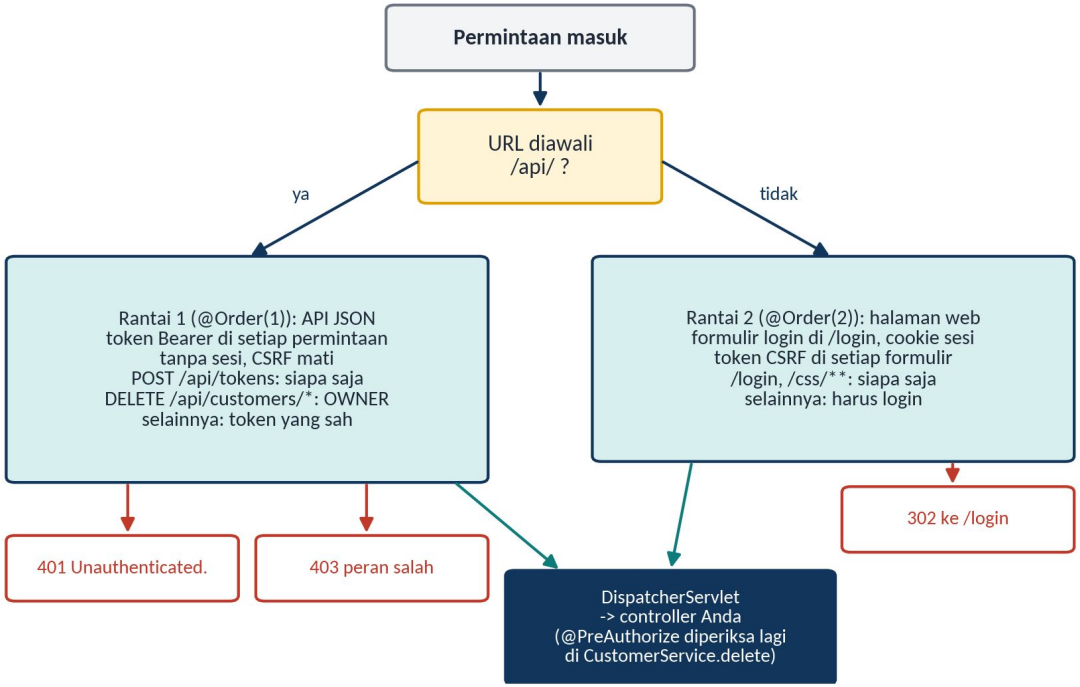
Dua Rantai Filter

Kode 14.1 Awal rantai API di `SecurityConfig.java`: satu rantai untuk API, satu untuk halaman.

```
@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())      // no cookies, so no CSRF risk
```

Kode 14.1: rantai API memakai `@Order(1)` dan mencocokkan `/api/**`; masuk bersifat terbuka, menghapus pelanggan butuh `OWNER`, selebihnya butuh identitas apa pun. Tanpa sesi, tanpa CSRF. Penolakannya berupa JSON: 401 "Unauthenticated." dan 403 "This action is unauthorized.", kata-kata Laravel. Rantai web, `@Order(2)`, memakai halaman login di `/login` dengan isian `email`, sesi, dan perlindungan CSRF. `hasRole("OWNER")` menambahkan awalan `ROLE_` sendiri. Buku: Subbab 14.5.1.

Menelusuri Permintaan dengan Tangan



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.

Tabel 14.1 Enam permintaan ditelusuri melalui kedua rantai.

Permintaan	Rantai	Identitas	Jawaban
`GET /api/orders`, tanpa *header*	API	tidak ada	401 JSON
`GET /api/orders`, token kasir	API	`ROLE_CASHIER`	200
`DELETE /api/customers/2`, token kasir	API	`ROLE_CASHIER`	403 JSON
`DELETE /api/customers/2`, token pemilik	API	`ROLE_OWNER`	204, atau 409 bila punya pesanan
`GET /orders`, tanpa *cookie*	web	tidak ada	302 ke `/login`
`POST /customers`, tanpa isian CSRF	web	seorang kasir	403

Gambar 14.1 dan Tabel 14.1: minta kelas menebak setiap jawaban sebelum kolomnya dibuka. Baris terakhir mengejutkan banyak orang: filter CSRF berjalan sebelum aturan apa pun, sehingga kasir yang sudah login tetap mendapat 403 untuk formulir tanpa tokennya. Buku: Subbab 14.5.2.

CSRF: Mengapa Halaman Membutuhkannya dan API Tidak

Peramban mengirim *cookie* sesi pada *setiap* permintaan ke situs, bahkan yang dipicu halaman situs lain. Formulir tersembunyi di tempat lain bisa mengirim `POST` ke `/customers` atas nama kasir yang sedang login.

Token CSRF, isian rahasia yang ada di halaman itu sendiri, membuktikan bahwa formulir berasal dari halaman kita. `th:action` menambahkannya otomatis; `action=` biasa tidak, dan formulirnya gagal dengan 403.

API tidak memakai *cookie*: token *bearer* tidak pernah dikirim otomatis, jadi hanya rantai yang tidak memakai *cookie* yang boleh tanpa CSRF.

Alasan yang sama membuat rute web Laravel butuh `@csrf` sedangkan rute API-nya tidak, dan formulir ASP.NET Core membawa token anti-pemalsuan. Logout adalah formulir dengan `POST`, tidak pernah sebuah tautan. Buku: Subbab 14.5.3 dan 14.5.4.

Lapis Pertahanan Kedua di *Service*

`CustomerService.delete` memakai `@PreAuthorize("hasRole('OWNER')")`: menghapus hanya untuk pemilik, siapa pun pemanggil metodenya.

Aturan URL melindungi rute API; aturan *service* melindungi setiap pemanggil di masa depan: sebuah halaman, tugas terjadwal, *endpoint* baru yang ditambahkan seseorang tahun depan.

Inilah **pertahanan berlapis** (*defense in depth*): dua pemeriksaan yang berdiri sendiri, sehingga satu kesalahan tidak membuka pintu.

`DELETE /api/customers/{id}` *yang baru menjawab 204 bila berhasil dan 409 bila pelanggan itu punya pesanan, karena pesanan adalah sejarah toko. Buku: Subbab 14.5.5.*

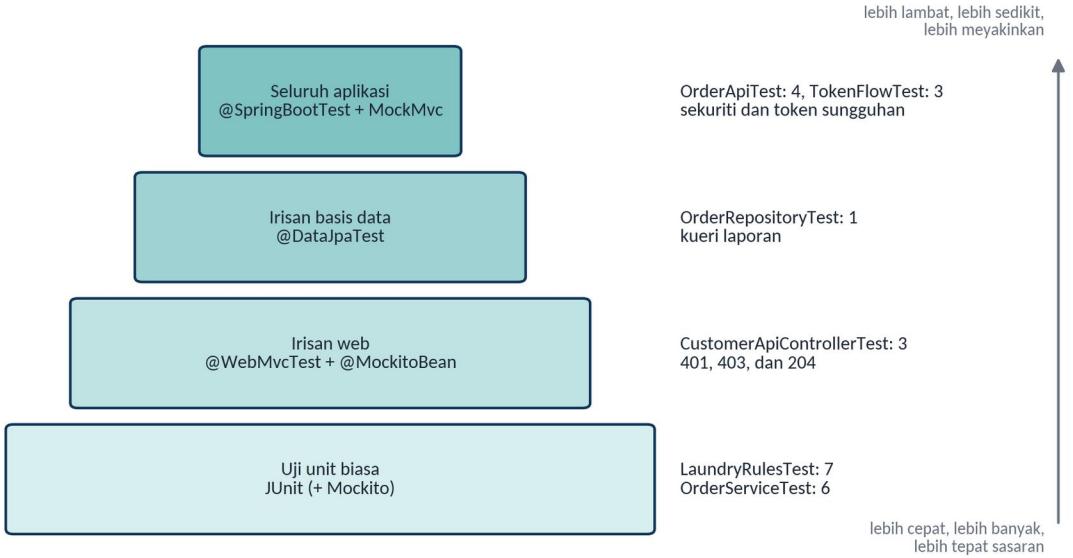
Merancang REST API

Tabel 14.2 Kode status API dan kapan masing-masing dipakai.

Kode	Arti di API ini
200 OK	pembacaan atau perubahan status berhasil
201 Created	token, pelanggan, pesanan, atau pembayaran dibuat
204 No Content	logout atau penghapusan berhasil
401 Unauthorized	tidak ada token yang sah
403 Forbidden	token sah, tetapi perannya tidak boleh melakukan ini
404 Not Found	catatannya tidak ada
409 Conflict	keadaan catatan tidak mengizinkan permintaan itu
422 Unprocessable Content	masukan melanggar aturan

*Tabel 14.2: kata benda di URL (`/api/orders/7/payments`), metode dengan makna yang jelas: `GET` **aman** (tidak mengubah apa pun), `PUT` dan `DELETE` **idempoten** (mengulangnya tidak mengubah apa-apa lagi), `POST` tidak keduanya. Sedikit kode status, amplop untuk data dan halaman. Yang masih kurang adalah versi. Selama aplikasi dan server dimutakhirkan bersama, itu masih bisa diterima; begitu sebagian sopir tidak memutakhirkan aplikasinya, setiap perubahan harus kompatibel ke belakang: isian boleh ditambah, tidak boleh diganti nama atau dihapus. Buku: Subbab 14.6.*

Pengujian Otomatis di Empat Tingkat



Gambar 14.2 Empat tingkat tes dalam proyek Bab 14, dari tes unit biasa di bawah sampai tes seluruh aplikasi di atas.

*Gambar 14.2: **piramida pengujian**. Banyak pengujian cepat di bawah, sedikit pengujian lambat di atas. Setiap tingkat hanya menyalakan apa yang dibutuhkannya, menangkap jenis kesalahannya sendiri, dan melewatkan jenis lain. Buku: Subbab 14.7.1.*

Apa yang Ditangkap Setiap Tingkat

Tabel 14.3 Empat tingkat pengujian, dengan 24 eksekusi di bab ini.

Tingkat	Menyalakan	Eksekusi	Menangkap	Melewatkan
Unit	hanya kelasnya	13	harga dan aturan yang salah	*wiring*, SQL, HTTP
Web slice	controller, JSON, keamanan	3	kode status, JSON, aturan akses	*service*, SQL
JPA slice	entitas, *repository*, H2	1	kueri dan pemetaan yang salah	HTTP, aturan
Seluruh aplikasi	semuanya, dengan H2	7	*wiring*, alur ujung ke ujung	sedikit, tetapi lambat

Tabel 14.3: ***web slice*** *(`@WebMvcTest`) menyalakan controller dengan aturan keamanan sungguhan dan service tiruan, sehingga memeriksa bahwa kasir mendapat 403 saat menghapus.* ***JPA slice*** *(`@DataJpaTest`) memeriksa kueri laporan terhadap H2. `TokenFlowTest` masuk, memakai token, keluar, lalu melihat tokennya ditolak. `mvn test` menjalankan semuanya dan menulis laporan ke `target/surefire-reports`. Buku: Subbab 14.7.2–14.7.4.*

Sebuah Pengujian *Web Slice*

Kode 14.2 Satu pengujian di `CustomerApiControllerTest.java`: lapisan web dan aturan keamanan, 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);
}
```

Kode 14.2: kelasnya adalah `@WebMvcTest(CustomerApiController.class)` dengan `@Import(SecurityConfig.class)`, sehingga rantai sungguhan ikut berjalan; `CustomerService` adalah `@MockitoBean`. `@WithMockUser(roles = "CASHIER")` memberi identitas tanpa token. Pengujian memeriksa status, pesan JSON yang persis, dan, dengan `verifyNoInteractions`, bahwa service bahkan tidak pernah dipanggil: penolakan terjadi di pintu. Pengujian saudaranya memeriksa 401 tanpa token dan 204 untuk pemilik. Buku: Subbab 14.7.2.

Menjadikannya Lebih Baik dan Memakainya Sungguhan

Idempotency key untuk `POST`, agar ponsel yang mengulang pembayaran tidak membayar dua kali; **token yang kedaluwarsa** dan daftar perangkat; **membatasi percobaan login**.

Mencatat siapa mengerjakan apa pada pesanan dan pembayaran, kini setelah akun ada; **HTTPS** di belakang *reverse proxy*, dengan `server.forward-headers-strategy=native`.

Perbaikan pertama paling penting bagi sopir dengan sinyal seluler yang lemah: permintaan yang kehabisan waktu sering diulang, dan tanpa kunci, server tidak bisa membedakan pengulangan dari pembayaran kedua. Buku: Subbab 14.8–14.10.

Kesalahan yang Sering Terjadi

Kata sandi dengan SHA-256 atau teks polos; **filter khusus sebagai** `@Component`; **rantai dalam urutan yang salah**; **CSRF dimatikan di mana-mana**.

`action=` **biasa di formulir Thymeleaf**; **jawaban berbeda untuk "surel tidak ada" dan "kata sandi salah"**; **peran hanya diperiksa di templat**; `hasRole("ROLE_OWNER")`.

Pengujian yang rusak setelah keamanan ditambahkan (beri `@WithMockUser` atau token sungguhan); **hanya pengujian seluruh aplikasi**.

Dua di antaranya layak didemonstrasikan. Taruh rantai web lebih dulu, tanpa pencocok, maka setiap permintaan API tiba-tiba dijawab dengan pengalihan ke halaman login yang tidak bisa dipakai ponsel mana pun. Tulis formulir Thymeleaf dengan `action=` *biasa, maka kasir yang sudah login mendapat 403 tanpa alasan yang terlihat. Keduanya hanya butuh semenit untuk diperlihatkan dan menghemat satu sore kelak. Buku: Subbab 14.11.*

Ringkasan: Lima Hal untuk Diingat

1. Identitas yang tidak ada adalah 401, penolakan adalah 403. Kata sandi berupa *hash* bergaram yang lambat (Persamaan (14.1)); token hanya menyimpan SHA-256 dari rahasia 240 bit (Persamaan (14.2)).
2. Dua rantai (Gambar 14.1): rantai API tanpa status dengan token *bearer* dan penolakan JSON, rantai web dengan login, sesi, dan CSRF (Tabel 14.1).
3. Menghapus pelanggan diperiksa di URL dan dengan `@PreAuthorize` di *service*: pertahanan berlapis.
4. REST: kata benda di URL, metode yang aman dan idempoten, sedikit kode status (Tabel 14.2).
5. Empat tingkat pengujian (Gambar 14.2, Tabel 14.3), 24 eksekusi, sebagian besar di tingkat bawah.

Dengan bab ini ketiga versi toko sejajar: masing-masing punya aturan di satu tempat, basis data dengan migrasi atau jalan menuju migrasi, halaman, API, login, peran, dan pengujian. Buku: Subbab 14.12.

Pertemuan Berikutnya

Bab 15 — Membandingkan Tiga Kerangka Kerja. Kinerja, produktivitas, ekosistem, dan cara memilih.

Sebelum itu: di komputer Anda jalankan `mvn test`, lalu rusak satu aturan dengan sengaja (biarkan kasir menghapus) dan lihat tingkat mana yang menangkapnya; kerjakan soal latihan bab ini.

Saat merusak aturan dengan sengaja, perhatikan tingkat mana yang gagal lebih dulu: pengujian web slice menangkapnya dalam hitungan detik, jauh sebelum pengujian seluruh aplikasi selesai. Itulah alasan piramida: kesalahan ditangkap di tingkat termurah yang bisa melihatnya. Buku: Subbab 14.13 dan 14.14.

Kuliah 15 — Membandingkan Tiga Kerangka Kerja: Kinerja, Produktivitas, Ekosistem, Memilih

Mata kuliah: Pemrograman Berbasis Kerangka Kerja

Buku: Irfan Subakti, *Pemrograman Berbasis Kerangka Kerja: ASP.NET Core, Laravel, dan Spring Boot*, Bab 15

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini kita mundur selangkah dari laundry dan membandingkan tiga kerangka kerja tempat kita membangunnya. Kita menjajarkan ketiganya konsep demi konsep, melihat bukti yang biasa dikutip orang (benchmark, ukuran kode, survei, kalender dukungan, bawaan keamanan), mengukur satu endpoint sendiri, dan menutup dengan prosedur singkat untuk memilih. Buku: Bab 15, Subbab 15.1–15.13.

Cakupan Pertemuan Ini

1. **Pilihan Dimas** — usaha katering, satu pengembang, tiga pendapat.
2. **Satu aplikasi, tiga kosakata** — konsep yang sama dengan tiga nama.
3. **Kinerja** — *benchmark* publik, pengukuran kita sendiri, dan hukum Little.
4. **Produktivitas, ekosistem, kalender, keamanan** — masing-masing dengan sumber dan tanggalnya.
5. **Memilih** — enam langkah, empat situasi.

Aturan yang mengalir sepanjang bab ini: sebutkan dari mana setiap angka berasal, kapan dikumpulkan, dan apa yang diukurnya. Pengukuran ASP.NET Core dijalankan untuk buku ini pada 25 September 2026; angka lainnya berasal dari sumber publik yang disebut beserta tanggalnya. Buku: Bab 15.

Posisi Kita Sekarang: Pilihan Dimas

Dimas lulus diploma informatika tahun ini dan melihat ketiga versi program kasir berjalan di komputer yang sama.

Keluarga seorang temannya menjalankan usaha katering kecil dengan dua belas pegawai dan memintanya membangun sistem pesanan. Ia harus memilih satu kerangka kerja.

Teman-teman sekelasnya masing-masing punya pendapat, masing-masing didukung grafik, survei, atau cerita.

Kejutan terbesar setelah membangun aplikasi yang sama tiga kali adalah betapa miripnya kerangka-kerangka kerja itu. Karena itulah pilihannya lebih bergantung pada orang dan tempatnya daripada pada kerangka kerjanya. Buku: Subbab 15.1.

Apa yang Dibandingkan, dan Bukti yang Dibutuhkan

Tabel 15.1 Apa yang dibandingkan, dan jenis bukti yang dibutuhkan setiap pertanyaan.

Pertanyaan	Jenis bukti
Cukup cepatkah?	*benchmark*, dan pengukuran sendiri atas beban kerja sendiri
Seberapa besar kerja sebuah fitur?	ukuran kode fitur yang sama, perkakas, kurva belajar
Bisakah kita merekrut dan mendapat bantuan?	survei, registri paket, pasar kerja dan *hosting* setempat
Berapa lama sebuah versi didukung?	kalender rilis dan dukungan resmi
Amankah secara bawaan?	bawaan untuk kata sandi, XSS, CSRF, SQL
Cocokkah dengan tim dan server kita?	apa yang dikuasai tim, di mana aplikasi harus berjalan

Tabel 15.1: setiap jenis bukti punya jebakannya. Benchmark mengukur satu beban kerja di satu mesin; survei mengukur siapa yang menjawabnya; kalender bisa berubah; pengukuran sendiri hanya sebaik metodenya. Buku: Subbab 15.2.

Satu Aplikasi, Tiga Kosakata

Tabel 15.2 Konsep yang sama dengan tiga nama (sebagian).

Konsep	ASP.NET Core 10	Laravel 13	Spring Boot 4.1
Perutean	`[HttpGet("...")]` atau `MapGet`	`routes/web.php`, `routes/api.php`	`@GetMapping`
Validasi masukan	anotasi data	Form Request, `validate()`	Jakarta Bean Validation
Gaya ORM	EF Core, *unit of work*	Eloquent, *active record*	JPA/Hibernate, *repository*
Perubahan skema	migrasi EF Core	migrasi Laravel	Flyway atau Liquibase
Templat	Razor	Blade	Thymeleaf
Login dan peran	Identity, `[Authorize(Roles=...)]`	*guard*, Gate, `can`	rantai filter, `@PreAuthorize`
Pengujian	xUnit, `WebApplicationFactory`	PHPUnit atau Pest	JUnit 6, `@WebMvcTest`

*Tabel 15.2: konsep demi konsep, ketiganya berpadanan. Perbedaan rancangan yang sesungguhnya ada dua: **active record** (Eloquent: objek menyimpan dirinya sendiri) lawan **unit of work** (EF Core, JPA: sebuah konteks melacak perubahan dan menyimpannya bersama), dan seberapa banyak yang diatur lewat **konvensi**. Riwayat satu permintaan adalah delapan langkah Bab 1 di ketiganya. Buku: Subbab 15.3.1 dan 15.3.2.*

Endpoint yang Sama Tiga Kali

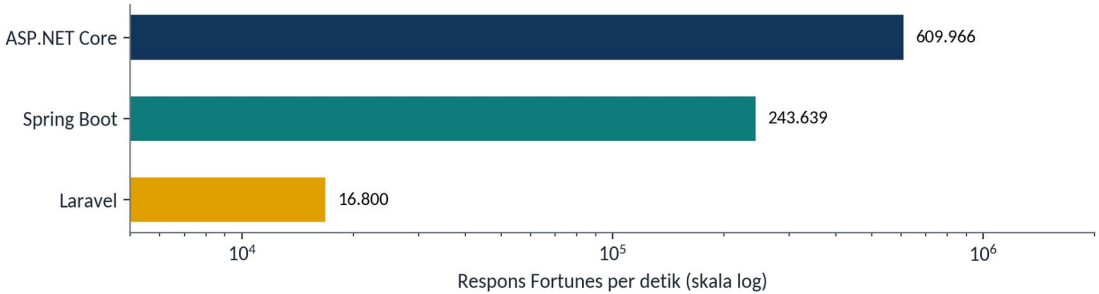
Kode 15.1 *Endpoint* penawaran di `PerfLab/Program.cs`, sebuah *minimal API* ASP.NET Core.

```
app.MapGet("/api/quote",
    (long price, string unit, decimal quantity, bool pickup) =>
{
    // The rule of Chapters 8 and 11, in hundredths of a unit.
    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);
});
```

$$T = \left\lfloor \frac{p \cdot b + 50}{100} \right\rfloor + \begin{cases} 10000 & \text{pickup and subtotal} < 100000 \\ 0 & \text{otherwise} \end{cases} \tag{15.1}$$

Kode 15.1 menghitung Persamaan (15.1), aturan Bab 8 dan 11: `GET /api/quote?price=8000&unit=kg&quantity=2.5&pickup=true` *menjawab total 34000. Buku ini menampilkan endpoint yang sama sebagai rute Laravel dan controller Spring: masing-masing sekitar selusin baris.* `CreateSlimBuilder`*, tanpa log, dan JSON yang dibangkitkan saat kompilasi menjaga pengukuran tetap tentang pekerjaannya, bukan tambahannya. Buku: Subbab 15.3.4.*

Apa yang Diukur *Benchmark* Publik

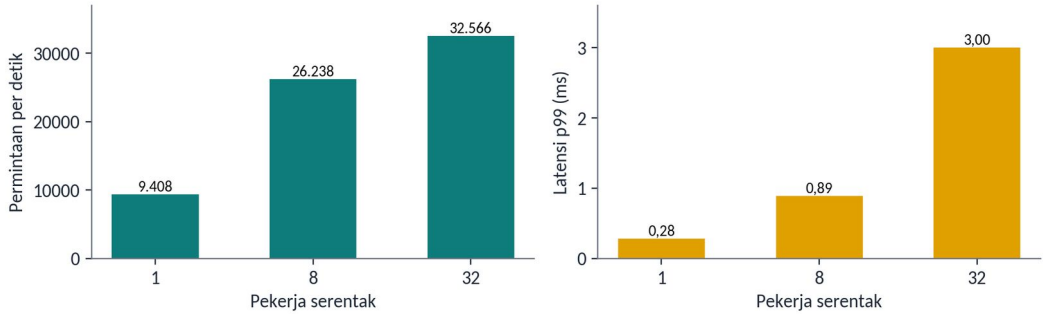


Gambar 15.1 TechEmpower Round 23 (Februari 2025), uji Fortunes: respons per detik dari implementasi ASP.NET Core, Spring, dan Laravel terbaik, dalam skala logaritmik.

Gambar 15.1: Round 23 adalah putaran terakhir benchmark TechEmpower, diarsipkan pada Maret 2026. Uji Fortunes-nya mengurutkan ASP.NET Core, Spring, dan Laravel dalam urutan itu, tetapi yang ditampilkan adalah entri terbaik yang sudah disetel dari setiap kerangka kerja di perangkat keras yang sangat besar. Perhatikan skala logaritmiknya. Untuk laundry, katering, atau klinik, ketiganya jauh lebih cepat daripada yang pernah dibutuhkan menit tersibuknya. Buku: Subbab 15.4.1.

Mengukurnya Sendiri

```
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 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.

Gambar 15.2: keluaran sungguhan `LoadGen`, pembangkit beban kecil, terhadap `PerfLab`. Throughput mendatar di batas kemampuan mesin sementara latensi terus naik: pekerja tambahan hanya menunggu lebih lama. Server dan klien berbagi dua inti, yang cukup untuk memperlihatkan metodenya tetapi tidak untuk dibandingkan dengan benchmark pusat data. Panaskan dulu, dan ukur dalam mode Release. Buku: Subbab 15.4.2.

Memeriksa Angka dengan Hukum Little

$$L = \lambda W \tag{15.2}$$

Dalam uji beban tertutup, setiap pekerja selalu punya tepat satu permintaan yang sedang berjalan: L adalah jumlah pekerja, λ adalah *throughput*, sehingga latensi rata-rata $W = L/\lambda$.

1 pekerja: $W = 1/9408$ s ≈ 0,106 ms. 8 pekerja: $8/26238$ s ≈ 0,305 ms. 32 pekerja: $32/32566$ s ≈ 0,983 ms.

Setiap rata-rata sedikit di atas median (0,09; 0,28; 0,84 ms) dan jauh di bawah p99: persis seperti latensi berekor panjang.

Persamaan (15.2) juga perkakas untuk langkah 3 dalam memilih: perkirakan menit tersibuk. Untuk usaha katering Dimas, menit tersibuk adalah pesanan telepon saat makan siang, beberapa permintaan per detik, dan kerangka kerja mana pun melayaninya di server kecil. Buku: Subbab 15.4.3.

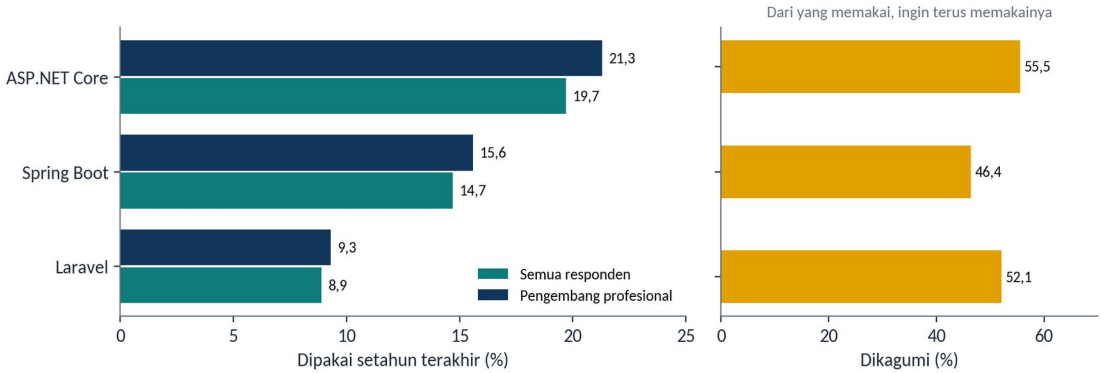
Produktivitas: Baris untuk Fitur yang Sama

Tabel 15.3 Jumlah baris untuk dua fitur yang sama di *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	sekitar 12	sekitar 17
Papan publik, controller dan view	18 + 42	17 + 37, ditambah 1 rute	21 + 35, ditambah 1 aturan

Tabel 15.3: jumlah baris yang mirip di ketiganya; Laravel paling pendek untuk fitur kecil berkat konvensinya (sebuah gate hanya satu baris). Baris hanyalah ukuran kasar: kurva belajar, perkakas, dan keakraban tim sama pentingnya. Waktu menyala dan memori juga berbeda, yang penting di server kecil atau bila membayar per kontainer. Buku: Subbab 15.4.4 dan 15.5.

Ekosistem dan Komunitas



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”).

Gambar 15.3: ASP.NET Core dipakai oleh 19,7% dari semua responden, Spring Boot oleh 14,7%, dan Laravel oleh 8,9%; sekitar separuh pengguna masing-masing ingin terus memakainya. (Bab 1 mengutip angka yang sedikit lebih tinggi khusus untuk pengembang profesional; selalu sebutkan populasi mana yang digambarkan sebuah angka.) Di luar survei: NuGet, Packagist, dan Maven Central, hosting PHP murah di mana-mana di Indonesia, dan pasar kerja setempat. Buku: Subbab 15.6.

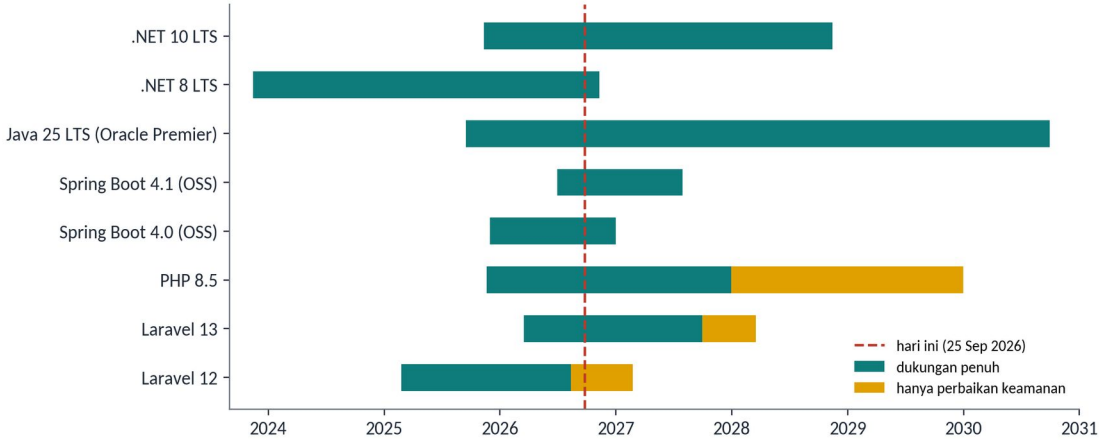
Kalender Dukungan

Tabel 15.4 Tanggal rilis dan dukungan (sebagian), dikumpulkan dari endoflife.date pada 25 September 2026.

Versi	Rilis	Perbaikan keamanan sampai
.NET 10 (LTS)	11 Nov 2025	14 Nov 2028
Java 25 (LTS, Oracle Premier)	16 Sep 2025	30 Sep 2030
Spring Boot 4.1 (sumber terbuka)	30 Jun 2026	31 Jul 2027
PHP 8.5	20 Nov 2025	31 Des 2029
Laravel 13	17 Mar 2026	17 Mar 2028

Tabel 15.4: versi LTS .NET mendapat tiga tahun; Laravel merilis versi mayor setiap tahun dengan dua tahun perbaikan keamanan; Spring Boot merilis versi minor setiap enam bulan dengan sekitar tiga belas bulan dukungan sumber terbuka. Proyek yang dimulai pada versi yang kehilangan dukungan dalam tiga bulan butuh pemutakhiran di siklus rilis pertamanya. Buku: Subbab 15.7.

Kalender Sekilas



Gambar 15.4 Jendela dukungan versi-versi yang dipakai buku ini; garis putus-putus adalah 25 September 2026.

Gambar 15.4: rencanakan pemutakhiran sebagai bagian dari anggaran sejak awal. Usaha kecil yang tidak sanggup memutakhirkan setiap tahun sebaiknya memilih jendela terpanjang yang bisa didapat, atau mitra hosting yang mengerjakan pemutakhirannya. Buku: Subbab 15.7.

Keamanan secara Bawaan

Tabel 15.5 Bawaan keamanan dibandingkan (sebagian).

Perhatian	ASP.NET Core	Laravel	Spring Boot
Injeksi SQL	parameter EF Core	nilai diikat	JPA mengikat nilai
XSS di templat	Razor meng-*encode* `@value`	Blade `{{ }}` meng-*escape*	`th:text` meng-*escape*
CSRF pada formulir	token antiforgery	token `@csrf`	token oleh Thymeleaf
Kata sandi	PBKDF2-HMAC-SHA512, 100.000	bcrypt, biaya 12	bcrypt, biaya 10

Tabel 15.5: bawaannya kuat dan mirip di ketiganya. Risikonya ada di pintu darurat: `Html.Raw`, `{!! !!}`, `th:utext`, SQL mentah, mematikan CSRF. Tim yang tahu di mana pintu darurat kerangka kerjanya lebih aman daripada tim yang memilih kerangka kerja "paling aman". Buku: Subbab 15.8.

Memilih: Enam Langkah

1. **Batasan yang tidak bisa bergeser:** di mana harus berjalan, dengan apa harus terhubung. **2. Apa yang sudah dikuasai tim:** dengan satu pengembang, pilih yang paling dikuasainya.
3. **Bebannya:** perkirakan menit tersibuk dengan hukum Little; biasanya kecepatan gugur sebagai pertimbangan. **4. Kalendernya:** sisa dukungan setidaknya satu tahun.
5. **Sebuah *spike*:** dua hari membangun satu fitur nyata di kandidat terdepan mengungkap lebih banyak daripada dua minggu membaca. **6. Tuliskan keputusannya**, beserta alasannya.

Sebuah batasan sering memutuskan sendirian: klien dengan server Windows dan SQL Server, atau bank dengan platform Java. Menuliskan keputusan memberi tahu orang berikutnya mengapa, dan kapan perlu ditinjau ulang. Buku: Subbab 15.9.

Empat Situasi

Tabel 15.6 Empat situasi dan kerangka kerja yang disarankan prosedurnya.

Situasi	Pilihan yang mungkin	Alasan utama
Usaha katering, satu pengembang PHP, *hosting* murah	Laravel	pengetahuan tim dan *hosting*
Grup rumah sakit dengan server Windows dan SQL Server	ASP.NET Core	platform dan staf yang sudah ada
Bank koperasi dengan sistem berbasis Spring	Spring Boot	cocok dengan platform dan aturan keamanannya
Start-up yang mengharapkan lalu lintas API berat, tim menguasai C# dan Java	ASP.NET Core atau Spring Boot, setelah pengukuran	kinerja per server, tipe yang ketat

Tabel 15.6: perhatikan betapa jarangnya kecepatan menjadi alasan utama. Untuk usaha katering Dimas, prosedurnya menunjuk ke apa yang sudah dimiliki dirinya dan hosting-nya. Buku: Subbab 15.9–15.11.

Kesalahan yang Sering Terjadi

Memilih berdasarkan grafik *benchmark*; angka tanpa sumber dan tanggal; mengukur aplikasi yang masih dingin; pembangkit beban di mesin jenuh, tanpa disebut.

Mengabaikan kalender dukungan; menganggap perbedaan kerangka kerja sebagai risiko utama; berganti kerangka kerja untuk memperbaiki masalah rancangan.

Yang terakhir umum dan mahal: controller gemuk dan aturan yang tersebar di banyak berkas tetap gemuk dan tersebar di kerangka kerja yang baru. Perbaiki lapisannya lebih dulu; Bab 3, 10, dan 13 memperlihatkan caranya di setiap kerangka kerja. Buku: Subbab 15.12.

Ringkasan: Lima Hal untuk Diingat

1. Ketiga kerangka kerja berpadanan konsep demi konsep (Tabel 15.2); perbedaan sesungguhnya adalah *active record* lawan *unit of work*, dan konvensi.

2. *Benchmark* memperlihatkan entri terbaik di perangkat keras besar (Gambar 15.1); ukur beban kerja Anda sendiri (Gambar 15.2) dan periksa dengan hukum Little (Persamaan (15.2)).

3. Ukuran kode mirip, Laravel paling pendek untuk fitur kecil (Tabel 15.3); ketiganya banyak dipakai dan dikagumi (Gambar 15.3).

4. Kalender berbeda (Tabel 15.4, Gambar 15.4); bawaan keamanan mirip, risikonya ada di pintu darurat (Tabel 15.5).

5. Pilih menurut batasan, tim, beban, kalender, dan sebuah *spike*, lalu tuliskan keputusannya (Tabel 15.1, Tabel 15.6).

Bagi Dimas prosedurnya singkat: satu pengembang yang paling menguasai PHP, hosting PHP yang sudah disewa, menit tersibuk berupa pesanan telepon saat makan siang dengan beberapa permintaan per detik. Laravel pilihannya, dicatat dalam architecture decision record satu halaman (konteks, pilihan, keputusan, konsekuensi), dengan catatan untuk meninjaunya ulang bila usaha itu kelak butuh aplikasi ponsel dengan lalu lintas berat. Buku: Subbab 15.9 dan 15.13.

Pertemuan Berikutnya

Bab 16 — Evaluasi dan Pembahasan IV. *Capstone*: Profitina Laundry lengkap di ketiga kerangka kerja.

Sebelum itu: jalankan `PerfLab` dan `LoadGen` di komputer Anda sendiri, periksa angkanya dengan hukum Little, dan tulis keputusan satu halaman untuk usaha yang Anda kenal.

Keputusan satu halaman itu sebaiknya memuat empat judul: konteks, pilihan yang dipertimbangkan, keputusan, dan konsekuensinya, termasuk apa yang akan membuat Anda meninjau ulang. Simpan di repositori proyek, di samping kodenya, agar orang berikutnya menemukannya. Buku: Subbab 15.14 dan 15.15.

Kuliah 16 — Evaluasi dan Pembahasan IV: *Capstone*

Mata kuliah: Pemrograman Berbasis Kerangka Kerja

Buku: Irfan Subakti, *Pemrograman Berbasis Kerangka Kerja: ASP.NET Core, Laravel, dan Spring Boot*, Bab 16

Dosen: Irfan Subakti — yifana@gmail.com

Hari ini seluruh mata kuliah bertemu. Profitina Laundry selesai di ketiga kerangka kerja: empat entitas yang sama, aturan yang sama, API JSON yang sama dengan login dan peran, dan papan konter publik yang sama. Kita membahas lima soal, satu kontrak beserta pemeriksanya, aturan dalam tiga bahasa, dan tiga capstone, lalu menutup dengan keputusan Bu Wati. Buku: Bab 16, Subbab 16.1–16.13.

Cakupan Pertemuan Ini

1. **Permintaan terakhir dan keputusannya** — papan di samping meja kasir, dan versi mana yang dijalankan gerai.
2. **Soal 16.1** — kontraknya, ditulis sebagai pemeriksa 25 permintaan.
3. **Soal 16.2** — aturan dalam tiga bahasa, dikerjakan dengan tangan.
4. **Soal 16.3–16.5** — *capstone* ASP.NET Core, Laravel, dan Spring Boot.
5. **Membandingkan dan memutuskan** — ketiga penyelesaian berdampingan, dan keputusan Bu Wati.

Bab ini menyatakan persis apa yang dijalankan, dalam empat tingkat bukti: diperiksa sintaksnya, dikompilasi, dijalankan, dan diperiksa terhadap kontrak. Capstone ASP.NET Core (versi memori) dijalankan di sini dan lolos pemeriksa; kode Laravel diperiksa sintaksnya; kode Spring dikompilasi terhadap pengganti dan aturannya dijalankan. Buku: Subbab 16.2.

Posisi Kita Sekarang

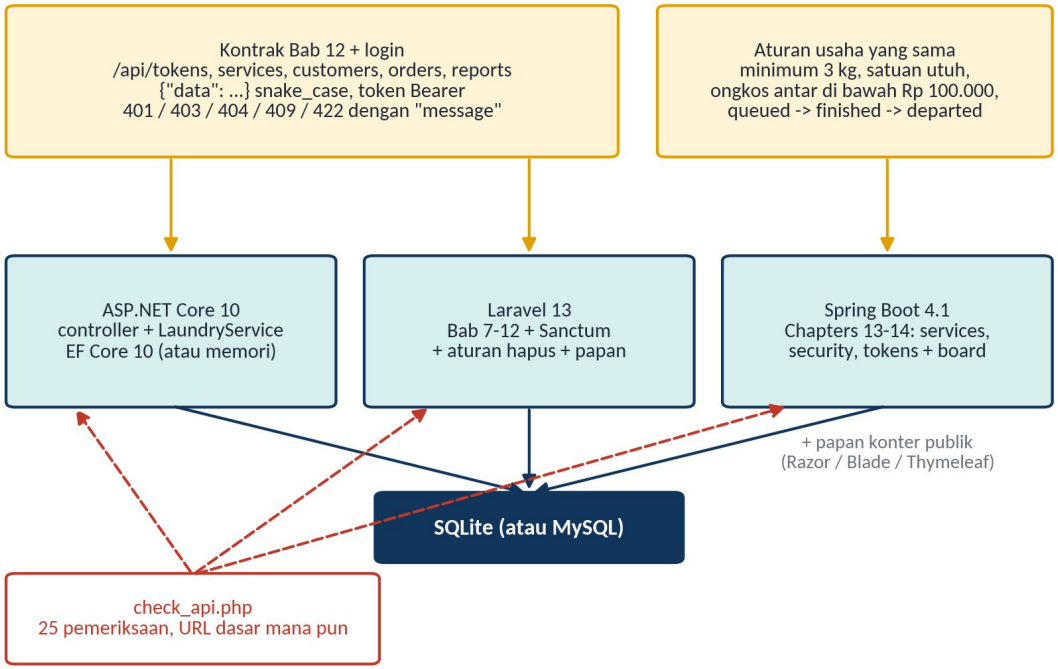
Bab 3–6 membangun *laundry* di ASP.NET Core, 7–10 di Laravel, 11–14 di Spring Boot; Bab 12 menulis API-nya sebagai kontrak; Bab 15 membandingkan kerangka-kerangka kerjanya.

Permintaannya: televisi di samping meja kasir di setiap gerai, menampilkan pesanan mana yang antre dan mana yang siap diambil, agar pelanggan berhenti bertanya “punya saya sudah siap?”.

Keputusannya: versi mana dari ketiganya yang akan dijalankan gerai. Raka tidak peduli yang mana, asalkan aplikasinya tetap bekerja.

Cara paling bersih menjaga aplikasi Raka tetap bekerja, versi mana pun yang berjalan, adalah membuat kontraknya begitu tepat sehingga sebuah program bisa memeriksanya. Itulah Soal 16.1. Buku: Subbab 16.1.

Cara Menghadapi *Capstone*



Gambar 16.1 Satu kontrak, tiga implementasi: pemeriksa `check_api.php` mengirim 25 permintaan yang sama ke setiap server dan membandingkan jawabannya dengan kontrak.

Gambar 16.1: rutinitas yang sudah akrab: nyatakan soalnya, kerjakan dengan tangan, rancang di kertas, selesaikan, jalankan dan periksa, cari penyelesaian keliru yang menggoda. Dua aturan baru: sebutkan apa yang dijalankan, dan tempatkan setiap hasil pada salah satu dari empat tingkat bukti. Hanya tingkat terakhir, diperiksa terhadap kontrak, yang membuktikan bahwa aplikasi ponsel akan bekerja. Buku: Subbab 16.2.

Soal 16.1: Kontrak sebagai Program

Enam belas pemeriksaan Bab 12, ditambah login dengan kata sandi salah, permintaan tanpa token, penghapusan oleh kasir yang ditolak 403, dua penghapusan oleh pemilik (409 bila punya pesanan, 204 bila tidak), logout, dan token lama yang ditolak.

Pemeriksanya PHP biasa, berjalan terhadap URL dasar mana pun, membangkitkan apa yang harus unik (nomor telepon acak), dan membaca isian yang dibaca aplikasi.

Membaca eksekusi yang gagal: pelajari `FAIL` yang *pertama*; bila login gagal, dua puluh pemeriksaan berikutnya gagal dengan 401 dari satu sebab.

Dua jenis kegagalan pertama yang umum: isian dengan nama atau bentuk lain (`customerId` untuk `customer_id`, `"QUEUED"` untuk `"queued"`, tanggal dengan pecahan detik), dan status yang berbeda (400 alih-alih 422). Perbaikannya di server, tidak pernah di pemeriksa. Buku: Subbab 16.3.1–16.3.5.

Pemeriksa Keliru yang Menggoda

Membandingkan seluruh jawaban JSON karakter demi karakter: gagal karena alasan yang salah. Id, waktu, dan telepon acak berbeda di setiap eksekusi; `4.5` lawan `4.50`; urutan isian di JSON tidak bermakna.

Hanya memeriksa kode status: server yang menjawab 422 dengan isi kosong, atau 201 dengan `"id": null`, lolos dan tetap merusak aplikasi.

Pemeriksa yang benar menyebut setiap harapan: status ini, isian ini, nilai ini, bentuk ini.

Kontrak yang hanya berupa tulisan dibaca berbeda oleh orang yang berbeda; kontrak sebagai program dibaca dengan cara yang sama setiap kali. Buku: Subbab 16.3.6.

Soal 16.2: Aturan dalam Tiga Bahasa

$$T = \left\lfloor \frac{8000 \cdot \max(250, 300) + 50}{100} \right\rfloor + 10000 = 24000 + 10000 = 34000 \tag{16.1}$$

$$B = T - P, 0 \le B \le T \tag{16.2}$$

Persamaan (16.1) adalah pesanan milik pemeriksa sendiri yang dikerjakan dengan tangan: 2,5 kg cuci, kering, dan lipat adalah 250 perseratus, ditagih 300, Rp 24.000, ditambah antar-jemput Rp 10.000 karena subtotalnya di bawah Rp 100.000. Persamaan (16.2): sisa tagihan tidak pernah di bawah nol (tanpa kelebihan bayar) atau di atas total. Buku: Subbab 16.4.1 dan 16.4.2.

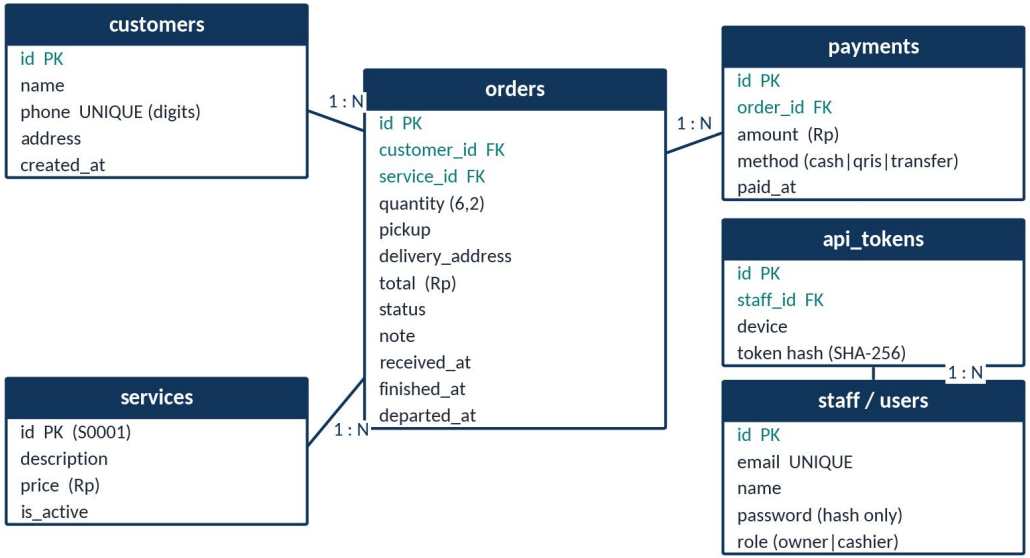
Sembilan Harga Dikerjakan dengan Tangan

Tabel 16.1 Enam dari sembilan harga yang dikerjakan dengan tangan.

Layanan, harga	Jumlah	Ditagih	Subtotal	Antar-jemput	Total
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
Gorden, 15.000	4,40 kg	440	66.000	ya	76.000
Jas, 65.000	2 potong	200	130.000	ya	130.000
Kemeja, 20.000	5 potong	500	100.000	ya	100.000
Sepatu, 45.000	1,5 pasang	200	90.000	tidak	90.000

Tabel 16.1: lihat kemejanya. Subtotal tepat Rp 100.000 tidak membayar biaya antar-jemput, karena aturannya berbunyi “di bawah”. `<=` di kode akan salah menagih Rp 10.000; hanya kasus di batas yang menangkapnya. Aturan Java dan C# menghasilkan total yang sama dengan aritmetika bilangan bulat dalam perseratus. Penyelesaian keliru yang menggoda, `Math.round(price * Math.max(quantity, 3))` dengan `double`, benar untuk sebagian besar masukan dan salah ketika 4,02 menjadi 4,0199999999999996. Buku: Subbab 16.4.3 dan 16.4.4.

Soal 16.3: *Capstone* ASP.NET Core



Gambar 16.2 Tabel-tabel *capstone*: layanan dari daftar harga (`S0001` sampai `S0012`), pelanggan, pesanan dengan antar-jemput, pembayaran, staf, dan token API.

Gambar 16.2: dibangun dari nol dengan semua yang diajarkan mata kuliah ini: `LaundryRules` dengan aritmetika bilangan bulat, antarmuka penyimpanan `ILaundryStore` dengan dua implementasi (EF Core dengan SQLite, dan memori), `LaundryService` dengan setiap aturan, tiga pengecualian, record permintaan dengan atribut validasi, `ApiErrors` untuk bentuk galat kontrak, dan penangan token bearer dengan token bergaya Sanctum yang di-hash. Sakelar `UseEf` di berkas proyek menentukan apakah EF Core ikut dikompilasi. Buku: Subbab 16.5.1–16.5.5.

Menjalankannya, dan Pemeriksaan Sungguhan

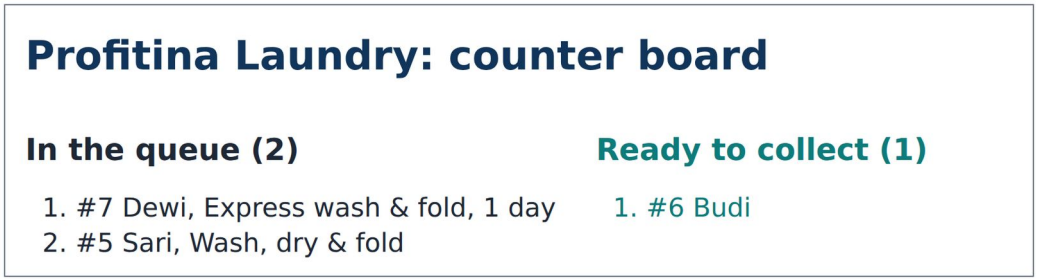
```
dotnet run -p:UseEf=false --urls http://localhost:5000
php ../../contract/check_api.php http://localhost:5000
```

```
PASS  a wrong password -> 422, errors.email
PASS  no token -> 401 with a message
PASS  POST /api/orders -> 201, total 34000 (3 kg + pickup), queued
PASS  a bad order -> 422, all of its errors at once
PASS  paying more than the balance -> 422, errors.amount
PASS  a cashier deletes a customer -> 403 with a message
PASS  the owner deletes a customer with orders -> 409
PASS  the owner deletes a customer without orders -> 204
PASS  the old token no longer works -> 401

25 passed, 0 failed
```

Sembilan dari 25 baris eksekusi sungguhan terhadap versi memori, dengan SDK 10.0.112 dan runtime 10.0.12; enam belas pemeriksaan Bab 12 juga lolos. Versi EF Core tidak bisa dijalankan di tempat buku ini ditulis, karena paketnya tidak bisa diunduh; README meminta Anda menjalankan pemeriksa terhadapnya lebih dulu. Buku: Subbab 16.5.6.

Papan Konter



Gambar 16.3 Papan konter *capstone* ASP.NET Core saat berjalan pada 25 September 2026: dua pesanan dalam antrean, satu siap diambil.

*Gambar 16.3: halaman publik, `[AllowAnonymous]`, yang menyegarkan diri setiap 30 detik dan hanya menampilkan **nomor pesanan dan nama depan**. Televisi di samping meja kasir dilihat setiap pelanggan di toko: tanpa nomor telepon, tanpa alamat, tanpa jumlah uang. Bab ini menguji bahwa sisanya tidak muncul, karena data pribadi di layar publik adalah kesalahan yang tidak disadari siapa pun sampai seorang pelanggan menyadarinya. Buku: Subbab 16.5.7.*

Dua Kesalahan yang Hanya Ditemukan Pemeriksa

```
public sealed record OrderRequest(
    [property: Required] long? CustomerId, ...
```

Kode ini dibangun tanpa peringatan. `POST /api/orders` yang pertama gagal: ASP.NET Core menolak atribut validasi pada properti *record* posisional dan menginginkannya pada parameter konstruktor.

Setelah *record*-nya diperbaiki, galat kembali dengan kunci `CustomerId` dan `Quantity`: `ModelState` menamai isian menurut properti C#, dan opsi JSON *snake_case* tidak berlaku untuk kunci-kunci itu.

Keduanya terkompilasi bersih dan akan lolos pengamatan sekilas di peramban. Pemeriksa, yang mengharapkan `customer_id`, langsung melaporkan yang kedua, dan `ApiErrors` kini mengubah kunci-kuncinya. Inilah sebabnya "dikompilasi" adalah tingkat bukti yang lebih rendah daripada "diperiksa terhadap kontrak". Buku: Subbab 16.5.8.

Soal 16.4: *Capstone* Laravel

Laravel butuh paling sedikit: kontraknya *adalah* bentuk bawaan Laravel. Yang baru: rute `DELETE` khusus pemilik, *gate* `delete-customers`, `destroy` yang menjawab 409 untuk pelanggan yang punya pesanan, dan papan sebagai rute publik dengan view Blade.

`CapstoneTest` memeriksa penghapusan oleh kasir, dua penghapusan oleh pemilik, dan privasi papan.

Keliru yang menggoda: `if (! $request->user()->isOwner()) abort(403);` di dalam controller. 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 aturannya di tempat rute dibaca, dan berjalan sebelum binding. Composer tidak bisa mengunduh Laravel di tempat buku ini ditulis, jadi kodenya diperiksa sintaksnya dengan `php -l`; di komputer Anda: `migrate:fresh --seed`, pengujian, server, lalu pemeriksa dengan akun demo Laravel. Buku: Subbab 16.6.

Soal 16.5: *Capstone* Spring Boot

Spring butuh papan dan satu aturan keamanan: `BoardController` membaca pesanannya lewat `OrderService`, `board.html` adalah papan yang sama dalam Thymeleaf, dan `/board` adalah `permitAll()` di rantai web.

Selebihnya, galat kontrak, token, dan penghapusan khusus pemilik, sudah datang dari Bab 12 sampai 14.

Di komputer Anda: `mvn test` menjalankan 24 eksekusi pengujian Bab 14, aplikasinya mendengarkan port 8080 dengan SQLite, dan pemeriksa memakai akun bawaan. Di tempat buku ini ditulis, setiap berkas Java dikompilasi terhadap pengganti dan aturannya dijalankan. Buku: Subbab 16.7.

Membandingkan Ketiga Penyelesaian

Tabel 16.2 Ketiga *capstone* dibandingkan (sebagian).

Perhatian	ASP.NET Core	Laravel	Spring Boot
"Wajib login kecuali ditandai"	*fallback policy*, `[AllowAnonymous]`	grup `auth:sanctum`	`authenticated()`, `permitAll()`
Khusus pemilik	`[Authorize(Roles = "OWNER")]`	*gate* dan `can:`	aturan URL dan `@PreAuthorize`
422 dengan setiap galat	gabungkan `ModelState` dan aturan	Form Request, bawaan	`Validator` dan aturan digabung
JSON *snake_case*	*naming policy*, plus kunci galat	lewat konvensi	pengaturan Jackson
Papan, nama depan saja	Razor	Blade	Thymeleaf
Yang dijalankan di sini	dibangun dan dijalankan; 25/25 dan 16/16	`php -l`	dikompilasi terhadap pengganti; aturan dijalankan

Tabel 16.2: kerangka kerja yang bawaannya cocok dengan kontrak butuh penyesuaian paling sedikit. Bagian sistem yang awet bukanlah kerangka kerjanya, melainkan aturan, kontrak, dan pemeriksanya: ketiganya akan bertahan tanpa perubahan di kerangka kerja keempat. Buku: Subbab 16.8.

Keputusan Bu Wati

Batasan: server virtual kecil yang disewa dengan memori 2 GB; ketiganya bisa berjalan di sana; tidak ada yang harus terhubung ke platform Java atau Windows.

Orangnya: Dimas yang merawat sistem paling menguasai PHP dan Laravel; tidak ada anggota keluarga yang bekerja dengan C# atau Java sehari-hari.

Beban: sekitar 150 pesanan sehari di tiga gerai, jauh di bawah satu permintaan per detik pada jam tersibuk; menurut hukum Little, kurang dari 0,03 permintaan sedang berjalan. **Kalender:** setiap versi didukung cukup lama untuk tahun pertama.

*Prosedurnya menunjuk ke **Laravel**: orang yang merawatnya menguasainya, cocok dengan hosting-nya, dan butuh penyesuaian paling sedikit terhadap kontrak. Keputusannya dicatat sebagai architecture decision record, dengan syarat untuk meninjaunya ulang: banyak gerai dengan server pusat, atau pengembang yang bekerja dengan C# atau Java. Dua versi lainnya sudah siap, dan pemeriksa akan memberi tahu dalam hitungan detik apakah perpindahan merusak sesuatu. Buku: Subbab 16.9.*

Kesalahan yang Sering Terjadi

Kontrak yang hanya berupa tulisan; pemeriksa yang tidak bisa dijalankan dua kali; uang dengan *floating point*; `<=` ketika aturannya berbunyi "di bawah".

`[property:]` pada atribut validasi *record* posisional; memercayai *naming policy* untuk kunci galat; data pribadi di layar publik.

Pemeriksaan peran di dalam metode controller; mengklaim lebih dari yang dijalankan.

Kesalahan terakhir adalah yang paling keras dihindari buku ini: setiap bab menyebutkan apa yang dibangun, apa yang dijalankan, dan apa yang hanya dikompilasi. Nyatakan hal yang sama tentang pekerjaan Anda sendiri. Buku: Subbab 16.12.

Ringkasan: Lima Hal untuk Diingat

1. Satu kontrak (Tabel 16.3) dan satu pemeriksa 25 permintaan (Gambar 16.1) mendefinisikan API untuk ketiga kerangka kerja.
2. Aturannya dikerjakan dengan tangan (Persamaan (16.1), Persamaan (16.2), Tabel 16.1) dan dihitung dengan bilangan bulat di setiap bahasa.
3. *Capstone* ASP.NET Core (Gambar 16.2) lolos 25 dari 25 pemeriksaan; dua kesalahan yang terkompilasi bersih hanya ditemukan pemeriksa.
4. Papan publik hanya menampilkan nomor pesanan dan nama depan (Gambar 16.3), dalam Razor, Blade, dan Thymeleaf.
5. Kerangka kerja yang bawaannya cocok dengan kontrak butuh penyesuaian paling sedikit (Tabel 16.2); aturan, kontrak, dan pemeriksa adalah bagian yang awet.

Kontraknya sendiri menyusul, satu-satunya artefak yang wajib dihormati setiap versi toko. Buku: Subbab 16.13.

Kontrak Setelah Bab 16

Tabel 16.3 Kontrak API Profitina Laundry setelah Bab 16.

Bagian	Aturan
Login	`POST /api/tokens` dengan `email`, `password`, `device`: 201 dengan token; kata sandi salah adalah 422 pada `email`
Pintu	tanpa token: 401 “Unauthenticated.”; peran yang tidak boleh: 403 “This action is unauthorized.”
Data	isian *snake_case*; satu catatan sebagai `{"data": {...}}`; satu halaman dengan `data` dan `meta`
Nilai	status dan metode huruf kecil; rupiah utuh; `YYYY-MM-DDTHH:MM:SS` dalam waktu toko
Masukan buruk	422 dengan `message` dan `errors`, semua galat sekaligus
Tidak ada, ditutup	404 dengan `message`; 409 untuk permintaan yang dilarang keadaan catatannya
Peran	hanya pemilik yang menghapus pelanggan; pelanggan yang punya pesanan tidak bisa dihapus (409)

Tabel 16.3: di awal mata kuliah keinginan Bu Wati adalah “satu perubahan di satu tempat”. Tabel ini, dan program yang memeriksanya, adalah keinginan itu untuk seluruh toko: satu tempat tempat janji kepada setiap klien ditulis, di tiga kerangka kerja sekaligus. Buku: Subbab 16.3.3, dan 16.14–16.15 untuk soal latihan beserta penyelesaiannya.