Pemrograman Web Lanjut

SIB245007 | D-IV Sistem Informasi Bisnis

Pertemuan 2: **Protokol HTTP dan Pola MVC**

Siklus Request/Response & Arsitektur Aplikasi Web

Yang Akan Kamu Pelajari

1. Menjelaskan siklus **request/response** HTTP (method, status code, header) dan memetakannya ke potongan kode routing
2. Membandingkan pola **MVC**, **MVVM**, dan **Clean Architecture**, serta menjelaskan bagaimana Laravel mengimplementasikan MVC
3. Mengenali peran **route**, **controller**, dan **middleware**, termasuk pola routing modern: **route parameter**, **named route**, **route group**, dan **resource routing**
4. Menjelaskan cara mengorganisasi controller: **resource controller**, **single-action controller**, dan prinsip *thin controller*

> Slide ini membahas konsep. Langkah menulis route, controller, dan middleware secara praktis dibahas terpisah di luar slide ini.

Bagian 1

Analogi Kantor Pos

Sepucuk surat

- Masuk lewat loket penerimaan
- Petugas loket membaca alamat & jenis surat
- Diteruskan ke bagian yang tepat
- Petugas loket tidak membuka amplop atau memutuskan isi balasan

Request ke Aplikasi Web

- Masuk lewat satu pintu
- Disortir berdasarkan alamat & jenisnya
- Diteruskan ke bagian yang tepat untuk diproses
- Penyortir tidak memutuskan isi jawaban

Kalau penyortiran keliru (mis. permintaan hapus data diteruskan tanpa memeriksa apakah pengirimnya admin), aplikasi kehilangan kendali atas siapa yang boleh mengubah apa.

Anatomi Sebuah Request

> **Request:** permintaan yang dikirim browser ke server, berisi method, alamat (URL), header, dan kadang data (form, JSON).

> **Route:** aturan yang memetakan satu kombinasi method dan alamat URL ke kode yang akan menanganinya.

- Setiap request selalu membawa sebuah **method** yang menyatakan maksud permintaan
- Method inilah yang menentukan route mana yang cocok, bukan alamat URL saja

Method HTTP

Method	Maksud	Contoh Penerapan
`GET`	Meminta data, **tanpa mengubah** apa pun di server	Menampilkan daftar data
`POST`	Mengirim data baru	Menyimpan data baru
`PATCH`	Mengubah sebagian data yang sudah ada	Memperbarui sebagian data
`DELETE`	Menghapus data	Menghapus data dari sistem

Method lain yang perlu kamu kenal: **`PUT`** (mengganti seluruh data sekaligus), **`HEAD`** (seperti `GET` tapi hanya meminta header, tanpa isi), **`OPTIONS`** (menanyakan method apa saja yang diizinkan server).

> Daftar lengkap method: `developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods`

Aturan Main Method: Aman & Idempoten

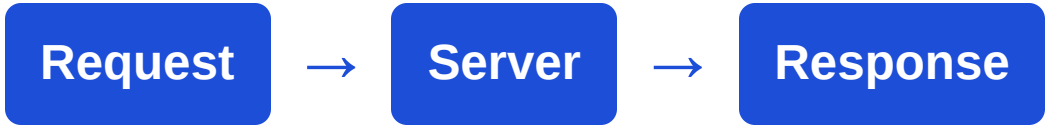
> **Aman (safe):** method yang tidak mengubah apa pun di server: `GET`, `HEAD`. **Idempoten:** method yang hasilnya sama meski dikirim berulang: `GET`, `PUT`, `DELETE`; sedangkan `POST` tidak.

- Browser & server berasumsi aturan ini dipatuhi: refresh, tombol back, dan cache semuanya bergantung padanya
- `POST` tidak idempoten, inilah alasan pola kirim-lalu-redirect di slide berikutnya diperlukan

> Aturan ketat: `GET` tidak boleh mengubah data di server. Melanggar aturan ini membuat perilaku aplikasi sulit ditebak, mis. me-refresh halaman tanpa sengaja menghapus data.

Anatomi Sebuah Response

Response: jawaban yang dikirim server kembali ke browser, berisi status code, header, dan isi (HTML, JSON, redirect).



Response selalu membawa **status code**, angka tiga digit yang menyatakan hasil permintaan secara ringkas, sebelum satu byte pun dari isinya dibaca.

Lima Kelas Status Code

Ratusan status code dikelompokkan lewat digit pertamanya: kamu cukup hafal lima kelasnya, bukan setiap kodenya.

Kelas	Arti	Intinya
`1xx`	Informational	"Diterima, masih diproses", jarang kamu temui langsung
`2xx`	Success	Permintaan berhasil diproses
`3xx`	Redirection	Browser diminta pergi ke alamat lain
`4xx`	Client Error	Kesalahan di sisi pengirim (alamat salah, data tidak valid)
`5xx`	Server Error	Kesalahan di sisi server

> Daftar lengkap: `developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status`

Status Code pada Aplikasi Web

Kode	Arti	Contoh Penerapan
200	OK	Halaman daftar data berhasil ditampilkan
302	Redirect	Setelah form disimpan, diarahkan ke halaman detail
404	Not Found	Membuka halaman detail untuk ID yang tidak ada
422	Unprocessable Entity	Form dikirim dengan data tidak valid
500	Server Error	Kesalahan tak tertangani di sisi server

- Perhatikan polanya: 2xx = sukses, 3xx = pindah alamat, 4xx = salah di sisi pengirim, 5xx = salah di sisi server

Pola 302: Kirim-lalu-Redirect

POST /articles → **302 + Location** → **GET /articles/{id}**

> Setelah `POST /articles` berhasil menyimpan data, server tidak langsung mengirim HTML sebagai jawaban: ia mengirim response 302 berisi header `Location` yang memerintahkan browser meminta ulang ke alamat lain.

- Pola ini berulang di hampir setiap fitur tulis-data
- Mencegah pengguna menekan **refresh** dan tanpa sadar mengirim data yang sama dua kali

Header: Metadata di Luar Isi

> **Header:** metadata yang menyertai request maupun response, di luar isi utamanya.

Header request (browser → server)

- `Content-Type` : format data yang dikirim
- `Accept` : format jawaban yang diinginkan
- `Authorization` : token identitas pengirim
- `Cookie` : data sesi yang dikirim balik

Header response (server → browser)

- `Content-Type` : format isi jawaban
- `Location` : alamat tujuan redirect
- `Set-Cookie` : server menitipkan data sesi
- `Cache-Control` : boleh-tidaknya jawaban disimpan

Daftar lengkap header: `developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers`

Melihat HTTP Secara Langsung

- Browser modern menyediakan **DevTools** → **tab Network** untuk melihat setiap request-response yang terjadi
- Setiap baris menampilkan: method, alamat, status code, dan waktu respons
- Cara paling langsung membuktikan bahwa "di balik setiap klik ada request HTTP yang nyata"

> Konsep ini akan langsung kamu praktikkan: membuka DevTools dan mengamati request sungguhan dari route yang kamu buat sendiri.

Bagian 2

Masalah yang Dijawab MVC

- Menerima request dan mengirim response saja **belum cukup** untuk menjaga kode tetap rapi begitu aplikasi tumbuh
- Tanpa pemisahan tanggung jawab yang tegas, kode "menyortir permintaan", "mengambil data", dan "menyusun tampilan" bercampur dalam satu berkas
- Pola **MVC** (Model-View-Controller) menjawabnya dengan membagi tiga tanggung jawab secara tegas

Model, View, dan Controller

MVC: Model mengurus data, View mengurus tampilan, Controller mengurus alur permintaan di antara keduanya.

Kembali ke analogi kantor pos

- Controller = petugas loket yang menyortir
- Model = arsip berisi data sesungguhnya
- View = formulir balasan yang diserahkan ke pengirim

Aturan praktis menaruh kode

- Menyentuh data → Model
- Mengatur alur permintaan → Controller
- Menampilkan → View

Framework tidak memaksa aturan ini: query di dalam view tetap bisa jalan. Pola ini adalah disiplin yang dijaga penulis kodenya sendiri.

MVC di Struktur Folder Laravel

Folder	Peran MVC
`routes/web.php`	Controller: pendaftaran alamat URL
`app/Http/Controllers/`	Controller: kelas pemroses request
`app/Models/`	Model: representasi data & aturan bisnis
`resources/views/`	View: tampilan yang dilihat pengguna

> Struktur folder ini bukan kebetulan: ia mewujudkan pola MVC secara konsisten sejak proyek pertama kali dibuat.

MVVM

> **MVVM (Model-View-ViewModel):** menyisipkan ViewModel di antara Model dan View; ViewModel menyimpan state tampilan dan otomatis menyinkronkannya ke View lewat binding dua arah.

- Populer pada aplikasi antarmuka **reaktif**: tampilan berubah terus-menerus tanpa reload halaman
- Lebih relevan untuk framework frontend (mis. Vue) dibanding aplikasi server-rendered pada umumnya

Clean Architecture

> **Clean Architecture:** mengisolasi aturan bisnis inti (use case) sepenuhnya dari framework yang dipakai, sehingga bisa diuji dan dipindah ke framework lain tanpa disentuh.

- Manfaat: aturan bisnis bisa diuji dan dipindah tanpa bergantung pada framework
- Ongkos: lapisan abstraksi tambahan yang perlu dirawat

> Untuk aplikasi skala kecil-menengah, isolasi seketat ini menambah abstraksi yang belum sepadan manfaatnya. MVC bawaan Laravel sudah cukup rapi.

Perbandingan Tiga Pola

Pola	Pemisahan Utama	Contoh Pemakaian
MVC	Model, View, Controller	Laravel (bawaan)
MVVM	Model, View, ViewModel	Aplikasi dengan binding dua arah antara tampilan & state
Clean Architecture	Lapisan use case terisolasi dari framework	Sistem berskala besar dengan banyak aturan bisnis

Bagian 3

Dari alamat URL sampai controller yang rapi

Route, Controller, dan Middleware

> **Controller:** class berisi method-method penanganan request, dipanggil oleh route yang cocok.

> **Middleware:** lapisan yang memeriksa atau mengubah request sebelum sampai ke controller, misalnya memeriksa apakah pengguna sudah login.

Request → **Route** → **Middleware** → **Controller**

- Bagian praktikum akan menerapkan ketiganya pada studi kasus Simple POS

Route Parameter: Alamat yang Dinamis

> **Route parameter:** bagian alamat yang ditulis dalam kurung kurawal, mis. `{id}`, yang nilainya diisi dari URL sesungguhnya dan diteruskan ke controller.

```
Route::get('/articles/{id}', [ArticleController::class, 'show']);
```

- `/articles/1`, `/articles/2`, `/articles/9999`: satu route melayani semuanya
- Nilai `{id}` diterima method `show` sebagai argumen
- ID yang tidak ada → **404** (lihat kembali tabel status code di Bagian 1)

Named Route: Alamat Boleh Berubah, Nama Tidak

> **Named route:** label unik yang ditempelkan ke sebuah route lewat `->name()`, sehingga bagian lain aplikasi merujuk nama itu, bukan alamat mentahnya.

```
Route::get('/login', [LoginController::class, 'create'])
    ->name('login.create');
Route::post('/login', [LoginController::class, 'store'])
    ->name('login.store');
```

- `GET /login` dan `POST /login` adalah **dua route berbeda** meski alamatnya sama, dibedakan oleh method-nya
- Link/redirect ditulis `route('login.create')`. Alamat berubah dari `/login` ke `/masuk`? Tidak ada pemanggil yang perlu diedit
- Konvensi penamaan: `sumber.aksi`, contoh `articles.index`, `articles.store`, dst.

Route Group: Aturan yang Dibagikan Bersama

Route group: membungkus beberapa route agar berbagi middleware, prefix alamat, atau prefix nama yang sama, ditulis sekali, berlaku untuk semuanya.

```
Route::middleware('auth')->group(function () {
    Route::get('/dashboard', [DashboardController::class, 'index'])
        ->name('dashboard');
    Route::get('/settings', [SettingController::class, 'edit'])
        ->name('settings.edit');
});
```

Route baru yang lupa dibungkus group middleware yang benar adalah lubang keamanan yang mudah luput: route hapus data yang seharusnya khusus admin bisa diakses siapa pun yang tahu alamatnya. Selalu periksa posisi route baru sebelum menganggapnya selesai.

Resource Controller: Tujuh Aksi Konvensi

Fitur CRUD apa pun selalu butuh tujuh aksi yang sama. Laravel membakukannya jadi konvensi nama method controller.

Aksi	Method	URL	Tugas
`index`	GET	`/articles`	Daftar semua artikel
`create`	GET	`/articles/create`	Form tambah artikel
`store`	POST	`/articles`	Simpan artikel baru
`show`	GET	`/articles/{id}`	Detail satu artikel
`edit`	GET	`/articles/{id}/edit`	Form ubah artikel
`update`	PATCH	`/articles/{id}`	Simpan perubahan
`destroy`	DELETE	`/articles/{id}`	Hapus artikel

`Route::resource`: Tujuh Route, Satu Baris

```php
Route::resource('articles', ArticleController::class);
```

- Satu baris ini mendaftarkan ke-7 route pada slide sebelumnya sekaligus, lengkap dengan named route (`articles.index`, `articles.show`, dst.)
- Butuh sebagian saja? `->only(['index', 'show'])`
- Konvensi yang sama di setiap fitur membuat siapa pun di tim langsung tahu di mana sebuah aksi berada

> Perintah `php artisan route:list` menampilkan tabel seluruh route yang terdaftar (method, URL, nama, dan middleware-nya) tanpa membuka berkas route satu per satu.

Single-Action Controller

> **Single-action controller:** controller dengan satu method `__invoke()` untuk satu tugas tunggal, dipakai ketika sebuah aksi tidak masuk akal digabung ke tujuh aksi resource.

```
Route::get('/articles/{id}/pdf', ExportArticlePdfController::class);
```

- Mengekspor artikel sebagai PDF bukan `show`, bukan `update`: ia aksi berdiri sendiri
- Route-nya menunjuk class-nya langsung, tanpa nama method
- Tanda kamu membutuhkannya: sebuah aksi terus "dipaksakan" masuk ke method resource yang tidak pas

Prinsip Thin Controller

> **Thin controller:** controller hanya mengurus alur: menerima request, memanggil pihak yang tepat, memilih response. Aturan bisnis tinggal di Model (atau lapisan service), bukan di controller.

Controller gemuk (hindari)

- Hitung total, validasi input, kirim notifikasi, format tampilan: semua di satu method
- Sulit diuji, sulit dipakai ulang

Controller tipis (tujuan)

- `store` hanya memvalidasi input, menyerahkan perhitungan ke Model, lalu redirect
- Logika bisnis bisa dipakai ulang dari mana pun

Ingat aturan dari Bagian 2: menyentuh data → Model, mengatur alur → Controller. Controller gemuk adalah pelanggaran pelan-pelan terhadap MVC: tiap baris terasa kecil, sampai suatu hari method `store`-mu 200 baris.

Rangkuman

- Request membawa **method** (`GET` / `POST` / `PATCH` / `DELETE` + `PUT` / `HEAD` / `OPTIONS`); response membawa **status code** yang terbagi lima kelas (`1xx` – `5xx`); **header** membawa metadata di kedua arah
- **MVC** memisahkan Model (data), View (tampilan), Controller (alur); **MVVM** untuk antarmuka reaktif; **Clean Architecture** untuk sistem besar. Laravel memetakan MVC langsung ke struktur foldernya
- Routing modern: **route parameter** (`{id}`), **named route** (`->name()`), **route group** (middleware/prefix bersama), dan **`Route::resource`** yang mendaftarkan tujuh aksi sekaligus
- Organisasi controller: **resource controller** untuk CRUD, **single-action controller** untuk aksi tunggal, dan prinsip **thin controller**: alur di controller, aturan bisnis di Model

Referensi & Diskusi

Dokumentasi resmi Laravel · MDN Web Docs (developer.mozilla.org) · Disertasi Fielding (2000) tentang REST

Kode lengkap: `github.com/se-polinema/simple-pos`

Pertemuan berikutnya: Frontend & Templating (Blade, Tailwind, Alpine)