Jobsheet Praktikum: Pertemuan 2

Route, Controller, dan Mengamati HTTP secara Langsung

Mata Kuliah	Pemrograman Web Lanjut (SIB245007)
Pertemuan	2 (Minggu 2)
Durasi	2 sesi × 170 menit
Sub-CPMK	Sub-CPMK 1: Mahasiswa mampu memahami konsep dasar web framework serta menerapkan routing, controller, dan pengelolaan basis data dalam pengembangan aplikasi web.
Kode Awal	branch `chapter-01` di `github.com/se-polinema/simple-pos`
Kode Akhir	branch `chapter-02` di `github.com/se-polinema/simple-pos`

A. Capaian Praktikum

Setelah menyelesaikan jobsheet ini, kamu mampu:

1. Mengamati siklus request/response HTTP secara langsung lewat browser DevTools dan `curl`.
2. Membuat controller dengan Artisan dan mendaftarkannya lewat route.
3. Membedakan dua route dengan alamat sama tapi method berbeda.
4. Membungkus sekelompok route dengan middleware dan menjelaskan efeknya.

B. Persiapan dan Prasyarat

- **Alat**: sama seperti Pertemuan 1 (PHP 8.2+, Composer, Node.js, Git), ditambah `curl` (sudah tersedia bawaan di macOS/Linux; di Windows tersedia lewat PowerShell modern atau Git Bash).
- **Kelanjutan kode**: lanjutkan proyek `simple-pos` milikmu dari Pertemuan 1. Kalau tertinggal atau proyekmu bermasalah, mulai dari kode awal pertemuan ini:

  ```
  git clone -b chapter-01 https://github.com/se-polinema/simple-pos.git
  cd simple-pos
  composer install
  npm install
  cp .env.example .env
  php artisan key:generate
  touch database/database.sqlite
  php artisan migrate:fresh --seed
  ```

- **Verifikasi cepat** sebelum mulai:

  ```
  git log --oneline
  ```

  Harus menampilkan minimal satu baris `increment 1: proyek Laravel kosong`. Jalankan juga `php artisan serve` di satu jendela terminal dan biarkan tetap berjalan sepanjang praktikum ini.

C. Langkah Kerja

Langkah 1: Route pertama dengan closure

Sebelum memakai controller, tulis dulu route paling sederhana: alamat yang langsung ditangani oleh sepotong kode (closure), tanpa class terpisah. Ini membuktikan bahwa route hanyalah pemetaan alamat → kode, sebelum controller ditambahkan sebagai lapisan pengorganisasian.

Buka `routes/web.php` dan tambahkan baris berikut (isi lengkap berkas setelah diedit):

```php
<?php
// routes/web.php

use Illuminate\Support\Facades\Route;

Route::get('/', function () {
    return view('welcome');
});

Route::get('/halo', function () {
    return 'Halo dari Simple POS';
});
```

Buka `http://127.0.0.1:8000/halo` di browser.

> ✅ **Checkpoint:** halaman menampilkan teks polos `Halo dari Simple POS`.

Langkah 2: Mengamati HTTP di DevTools

Buka DevTools browser (klik kanan → Inspect, atau F12), pindah ke tab **Network**, aktifkan **Preserve log**, lalu muat ulang `http://127.0.0.1:8000/halo`.

> ✅ **Checkpoint (isi tabel ini):**

Alamat	Method	Status Code	Content-Type
`/halo`			

Sekarang buka alamat yang sengaja tidak terdaftar, mis. `http://127.0.0.1:8000/tidak-ada`, dan catat baris barunya di tabel yang sama.

Alamat	Method	Status Code	Content-Type
`/tidak-ada`			

> ⚠ **Jika gagal:** kalau tab Network kosong setelah reload, pastikan **Preserve log** aktif dan tab DevTools sudah terbuka **sebelum** halaman dimuat ulang.

Langkah 3: Mengamati HTTP dengan curl

DevTools menampilkan HTTP lewat antarmuka visual; `curl` menampilkannya sebagai teks mentah, cara yang sama dipakai untuk menguji API nantinya.

```
curl -i http://127.0.0.1:8000/halo
```

> ✅ **Checkpoint:** baris pertama output adalah `HTTP/1.1 200 OK`, diikuti header-header lain (`Content-Type`, dll.), lalu baris kosong, lalu isi `Halo dari Simple POS`.

Langkah 4: Membuat TransactionController

Closure cukup untuk satu route sederhana, tapi tidak praktis begitu logikanya bertambah. Controller mengelompokkan method-method penanganan request yang saling berhubungan dalam satu class.

```
php artisan make:controller TransactionController
```

> ✅ **Checkpoint:** berkas baru muncul di `app/Http/Controllers/TransactionController.php`, isinya berupa class kosong seperti berikut:

```php
<?php

namespace App\Http\Controllers;

class TransactionController extends Controller
{
```

```
    //
}
```

Langkah 5: Mengisi method stub

Simple POS pada akhirnya akan punya empat method di controller ini: `create` (menampilkan halaman kasir), `store` (menyimpan transaksi baru), `index` (menampilkan daftar transaksi), dan `show` (menampilkan detail satu transaksi). Tampilan sungguhan (Blade) baru dibahas Pertemuan 3, jadi untuk sekarang setiap method cukup mengembalikan teks biasa sebagai bukti bahwa alurnya benar.

Ganti isi `app/Http/Controllers/TransactionController.php` menjadi:

```php
<?php

namespace App\Http\Controllers;

class TransactionController extends Controller
{
    public function create()
    {
        return 'Halaman kasir (belum ada tampilan)';
    }

    public function store()
    {
        return 'Transaksi disimpan (belum ada logika penyimpanan)';
    }

    public function index()
    {
        return 'Daftar transaksi';
    }

    public function show(string $id)
    {
        return "Detail transaksi #{$id}";
    }
}
```

Langkah 6: Mendaftarkan route ke controller

Ganti isi `routes/web.php` menjadi (perhatikan baris `use` yang mengimpor controller):

```php
<?php
// routes/web.php

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

Route::get('/', function () {
    return view('welcome');
});

Route::get('/halo', function () {
    return 'Halo dari Simple POS';
});

Route::get('/pos', [TransactionController::class, 'create'])
    ->name('pos.create');
Route::post('/pos', [TransactionController::class, 'store'])
    ->name('transactions.store');
Route::get('/transactions', [TransactionController::class, 'index'])
    ->name('transactions.index');
```

Buka `http://127.0.0.1:8000/pos` dan `http://127.0.0.1:8000/transactions` di browser.

> ✓ **Checkpoint:** `/pos` menampilkan teks `Halaman kasir (belum ada tampilan)`; `/transactions` menampilkan teks `Daftar transaksi`.

> ⚠ **Jika gagal:** pesan error `Target class [TransactionController] does not exist` berarti baris `use App\Http\Controllers\TransactionController;` terlupa atau salah ketik. Periksa kembali baris paling atas berkas.

Langkah 7: Memeriksa daftar route

Alih-alih membuka `routes/web.php` satu per satu untuk memastikan route terdaftar, Artisan menyediakan perintah yang menampilkan tabel ringkas.

```
php artisan route:list
```

> ✓ **Checkpoint:** tabel menampilkan `GET /pos` dan `POST /pos` sebagai **dua baris terpisah**, meski alamatnya sama persis: keduanya dibedakan oleh method HTTP-nya, satu menampilkan formulir kasir, satu lagi memprosesnya.

Langkah 8: Membungkus route dengan middleware `auth`

Route yang seharusnya hanya bisa diakses pengguna yang sudah login perlu dibungkus middleware. Ubah bagian route Simple POS di `routes/web.php` menjadi:

```php
<?php
// routes/web.php

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

Route::get('/', function () {
    return view('welcome');
});

Route::get('/halo', function () {
    return 'Halo dari Simple POS';
});

Route::middleware('auth')->group(function () {
    Route::get('/pos', [TransactionController::class, 'create'])
        ->name('pos.create');
    Route::post('/pos', [TransactionController::class, 'store'])
        ->name('transactions.store');
    Route::get('/transactions', [TransactionController::class, 'index'])
        ->name('transactions.index');
});
```

Muat ulang `http://127.0.0.1:8000/pos`.

> ✓ **Checkpoint (ini bukan kesalahan, baca sampai selesai):** halaman menampilkan error `Route [login] not defined.` Ini justru **bukti bahwa middleware bekerja**: sebelum request sampai ke `TransactionController`, middleware `auth` memeriksa apakah pengirim sudah login, mendapati belum, lalu mencoba mengarahkan ke halaman login, yang belum dibuat sampai pertemuan tentang autentikasi nanti. Jalankan `php artisan route:list` sekali lagi dan perhatikan kolom middleware kini menampilkan `auth` di baris `/pos` dan `/transactions`.

> ⚠ **Jika gagal (dalam arti sesungguhnya):** kalau error yang muncul justru `Class "auth" does not exist` atau sejenisnya, periksa penulisan `Route::middleware('auth')`: nama middleware harus persis string `'auth'`, bukan nama class.

Langkah 9: Controller memproses sebelum merespons

Controller tidak sekadar meneruskan request: ia bisa memproses data terlebih dahulu sebelum mengirim response. Ubah method `create()` di `TransactionController` untuk membuktikannya:

```php
public function create()
{
```

```
    $waktu = now()->format('H:i:s');

    return "Halaman kasir dibuka pukul {$waktu}";
}
```

Muat ulang `/pos` dua kali dengan jeda beberapa detik.

> ✓ **Checkpoint:** waktu yang ditampilkan berubah setiap reload, bukti bahwa controller menjalankan kode PHP baru setiap kali request masuk, bukan menampilkan halaman statis yang sama.

Langkah 10: Tantangan mandiri

Tambahkan satu route baru `GET /pos/riwayat` pada `routes/web.php`, di dalam group middleware `auth` yang sama seperti route `/pos` lainnya, mengarah ke method baru bernama `riwayat` pada `TransactionController`. Method-nya cukup mengembalikan teks biasa, misalnya `return "Riwayat kasir";`.

> ✓ **Checkpoint:** `php artisan route:list` menampilkan route barumu dengan alamat `pos/riwayat`, method `GET`, dan middleware `auth`, tanpa membuka `/pos/riwayat` di browser (karena akan menampilkan error `Route [login]` yang sama seperti Langkah 8, dan itu diharapkan).

Langkah 11: Commit `increment 2`

```
git add .
git commit -m "increment 2: route dan controller transaksi"
git log --oneline
```

> ✓ **Checkpoint:** `git log --oneline` menampilkan dua baris: `increment 2: ...` di atas, `increment 1: ...` di bawahnya.

D. Tugas dan Deliverable

Kumpulkan hal berikut sesuai format yang diminta asisten/dosen:

- Output `php artisan route:list` setelah Langkah 10 (menunjukkan route `/pos/riwayat`).
- Screenshot tab Network DevTools untuk `/halo` (status 200) dan `/tidak-ada` (status 404).
- Tabel isian Langkah 2.
- Output `git log --oneline` menunjukkan commit `increment 2`.
- **Tugas mandiri:** jelaskan dengan kata-katamu sendiri, dalam 3-5 kalimat: (a) mengapa `GET /pos` dan `POST /pos` dianggap dua route berbeda meski alamat URL-nya identik, dan (b) apa yang akan terjadi kalau route `DELETE /produk/{id}` lupa dibungkus middleware yang memeriksa peran admin.

E. Kriteria Penilaian

Komponen	Bobot	Kriteria Lengkap (100%)	Kriteria Minimum
Langkah kerja tuntas	40%	Seluruh Langkah 1-11 selesai dan berfungsi	Sebagian besar langkah selesai
Checkpoint terverifikasi	30%	`route:list`, screenshot DevTools, dan tabel isian lengkap dan benar	Sebagian checkpoint terbukti
Tugas mandiri	20%	Kedua penjelasan tepat dan berdiri sendiri	Jawaban ada meski belum lengkap

Komponen	Bobot	Kriteria Lengkap (100%)	Kriteria Minimum
Kerapian commit	10%	Pesan commit persis `increment 2: route dan controller transaksi`	Commit ada meski pesan kurang rapi