MENULIS KODE PROGRAM DENGAN PRINSIP SESUAI GUIDELINES DAN BEST PRACTICE (LARAGON)

Disusun Oleh

ABDULLAH AZZAM RABBANI

10240038

PROGRAM STUDI REKAYASA PERANGKAT LUNAK

UNIVERSITAS BINA SARANA INFORMATIKA

MARGONDA

2024/202

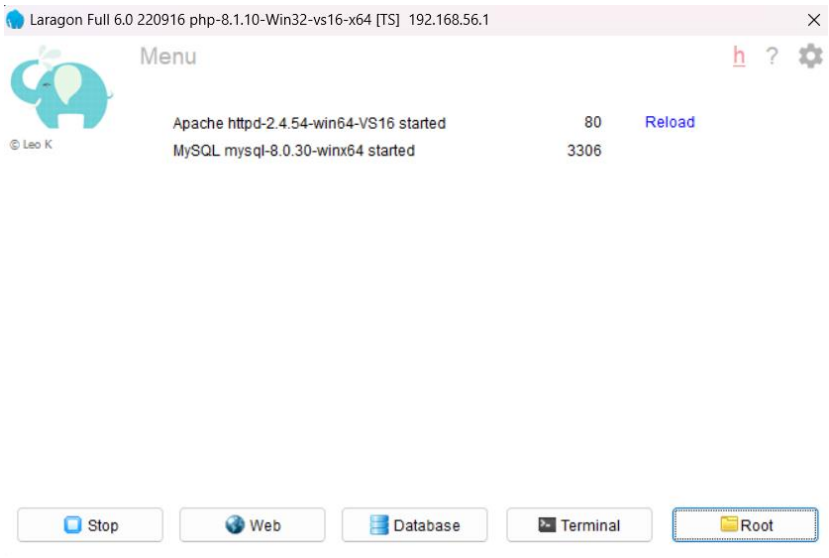
DAFTAR ISI

BAB I

INSTALASI DAN KONFIGURASI LINGKUNGAN (LARAGON)

1.1. Memastikan Laragon Berfungsi

1. Start Laragon: Pastikan Web Server dan Database telah berjalan.
2. Verifikasi: Cek status Apache/Nginx dan MySQL/MariaDB.

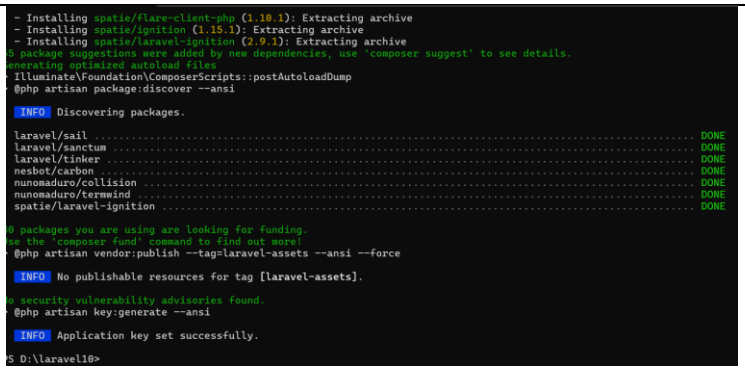


Gambar 1.1: Tangkapan layar aplikasi **Laragon** yang sedang **Running**. Soroti tombol "Start All" dan pastikan status Server & Database terlihat jelas.

1.2. Instalasi Proyek dan Struktur Direktori

1. Jalankan Perintah Instalasi: Gunakan Laragon Terminal untuk membuat proyek baru (Contoh: myapp).

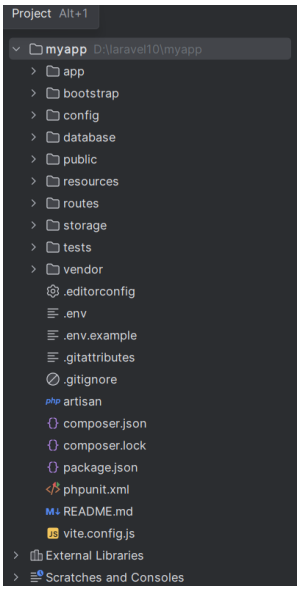
composer create-project laravel/laravel:^10.0 myapp



Gambar 1.2: Tangkapan layar **Terminal Laragon** yang menunjukkan hasil eksekusi perintah composer create-project....

1. **Struktur Proyek:** Periksa struktur direktori yang dihasilkan

Menggunakan *framework* dan *dependency manager* (Composer) memastikan Struktur program yang sesuai dengan konsep paradigmanya. Struktur MVC yang terbagi (*app*, *public*, *routes*) adalah *best practice* yang menjamin **Separation of Concerns**, membuat kode lebih modular dan mudah di-debug.



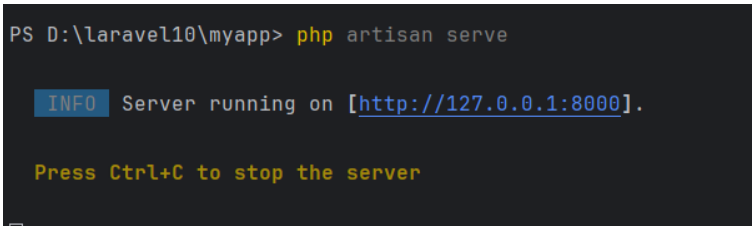
Gambar 1.3: Tangkapan layar **struktur folder proyek** (myapp).

BAB II
MENJALANKAN PROGRAM DAN VERIFIKASI AWAL

Langkah ini membuktikan bahwa proyek fungsional dan siap diakses, yang merupakan bagian dari Kemudahan interaksi

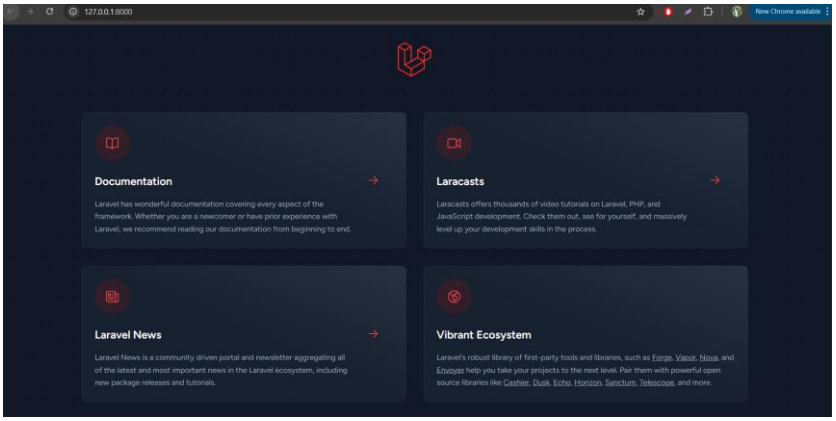
2.1. Eksekusi Program Melalui Virtual Host

1. Untuk menjalankan program cukup pergi ke ternimal di dalam projek dan eksekusi perintah **php artisan serve**



Gambar 1.4: Tangkapan layar ketika server sudah running

2. Akses browser dengan cara ketik url virtualhost http://127.0.0.1:8000 atau http://localhost:8000 dan pastikan halaman sama dengan gambar berikut



Gambar 1.5: halaman laravel yang sudah running dan berhasil

BAB III
IMPLEMENTASI ROUTE DAN CONTROLLER (PENERAPAN PRINSIP MVC)

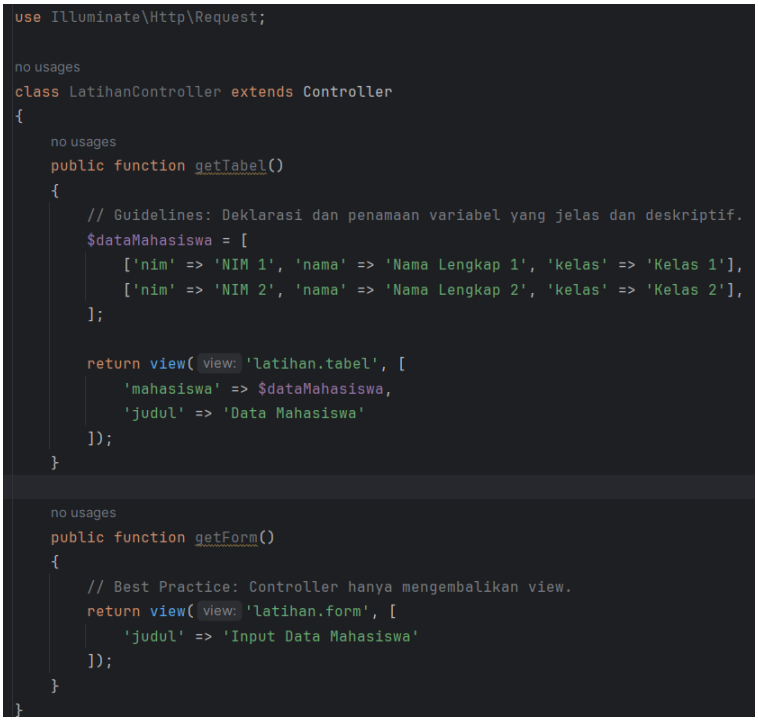
Langkah ini adalah inti dari demonstrasi kompetensi penulisan kode. Kita akan menerapkan Guidelines dan Best Practices untuk memastikan kode tidak hanya berfungsi, tetapi juga mudah dirawat (*maintainable*), mudah diuji (*testable*), dan memenuhi standar industri.

3.1. Implementasi Controller: Menerapkan SRP dan Dependency Injection

Controller harus memiliki tanggung jawab tunggal (SRP) dan tidak boleh mengandung logika bisnis yang kompleks. Sebaliknya, ia harus menggunakan layanan (*Service*) atau model lain melalui *Dependency Injection*.

a) Buatlah controller: Gunakan dengan perintah **php artisan make:controller LatihanController.** Controller ini akan berisi dua *method* (getTabel dan getForm) yang langsung menyiapkan data dan mengarahkan ke *View*.

b) **Tulis Kode:** Tulis *method* `getTabel()` dan `getForm()` di dalam *Controller*.



```php
use Illuminate\Http\Request;

no usages
class LatihanController extends Controller
{
    no usages
    public function getTabel()
    {
        // Guidelines: Deklarasi dan penamaan variabel yang jelas dan deskriptif.
        $dataMahasiswa = [
            ['nim' => 'NIM 1', 'nama' => 'Nama Lengkap 1', 'kelas' => 'Kelas 1'],
            ['nim' => 'NIM 2', 'nama' => 'Nama Lengkap 2', 'kelas' => 'Kelas 2'],
        ];

        return view( view: 'latihan.tabel', [
            'mahasiswa' => $dataMahasiswa,
            'judul' => 'Data Mahasiswa'
        ]);
    }

    no usages
    public function getForm()
    {
        // Best Practice: Controller hanya mengembalikan view.
        return view( view: 'latihan.form', [
            'judul' => 'Input Data Mahasiswa'
        ]);
    }
}
```

Gambar 1.6: Tangkapan layar Kode Sumber LatihanController.php di *Code Editor* Anda

Penjelasan (Coding Guidelines & Best Practices):

- **Penamaan & Indentasi:** Nama *Class* PascalCase (LatihanController) dan *method* camelCase (getTabel, getForm) dengan indentasi konsisten menjamin kode mudah dibaca. Penamaan variabel ($dataMahasiswa) yang jelas menghindari ambiguitas.
- **Pemisahan Tugas:** Meskipun data dideklarasikan di *Controller* (untuk studi kasus sederhana), tanggung jawab *Controller* tetap dipisahkan dari *View*—ia hanya bertugas menyiapkan data dan mengarahkan ke *view* (return view(...)).
- **Struktur Data:** Data diorganisir dalam *array* terstruktur, yang merupakan *best practice* dalam pengiriman data ke lapisan presentasi (*View*).

3.2. Implementasi View

1. **Buat Direktori:** Buat folder **latihan** di dalam resources/views/.
2. **View tabel.blade.php** (Output untuk getTabel):

```
<!DOCTYPE html>
<html>
<head>
    <title>{{ $judul }}</title>
</head>
<body>
<h1>{{ $judul }}</h1>
<table border="1">
    <thead>
    <tr>
        <th>No</th>
        <th>NIM</th>
        <th>Nama</th>
        <th>Kelas</th>
    </tr>
    </thead>
    <tbody>
    @foreach ($mahasiswa as $index => $data)
        <tr>
            <td>{{ $index + 1 }}</td>
            <td>{{ $data['nim'] }}</td>
            <td>{{ $data['nama'] }}</td>
            <td>{{ $data['kelas'] }}</td>
        </tr>
    @endforeach
    </tbody>
</table>
</body>
</html>
```

Gambar 1.7: Tangkapan layar Kode Sumber tabel.blade.php**.**

3. **View form.blade.php** (Output untuk getForm):

```
<!DOCTYPE html>
<html>
<head>
    <title>{{ $judul }}</title>
</head>
<body>
<h1>{{ $judul }}</h1>
<form action="" method="POST">
    <label for="nim">NIM</label><br>
    <input type="text" id="nim" name="nim"><br><br>

    <label for="nama">Nama Lengkap</label><br>
    <input type="text" id="nama" name="nama"><br><br>

    <label for="kelas">Kelas</label><br>
    <input type="text" id="kelas" name="kelas"><br><br>

    <button type="submit">Simpan</button>
</form>
</body>
</html>
```

Gambar 1.8: Tangkapan layar Kode Sumber form.blade.php.

3.3. Implementasi Route: Clean Routing

Kita harus mendefinisikan *Route* agar dapat memanggil kedua *method* di atas dengan URL yang bersih.

1. **Edit File Route**: Buka routes/web.php.
2. **Definisikan Route:** Daftarkan *Route* untuk kedua fungsi tersebut.

```php
use Illuminate\Support\Facades\Route;
// Guideline: Import Class di awal file
use App\Http\Controllers\LatihanController;



Route::get( uri: '/', function () {
    return view( view: 'welcome');
});
// Route untuk menampilkan Tabel Data Mahasiswa
Route::get( uri: '/latihan/tabel', [LatihanController::class, 'getTabel'])
    ->name( name: 'latihan.tabel');
// Route untuk menampilkan Formulir Input
Route::get( uri: '/latihan/form', [LatihanController::class, 'getForm'])
    ->name( name: 'latihan.form');
```

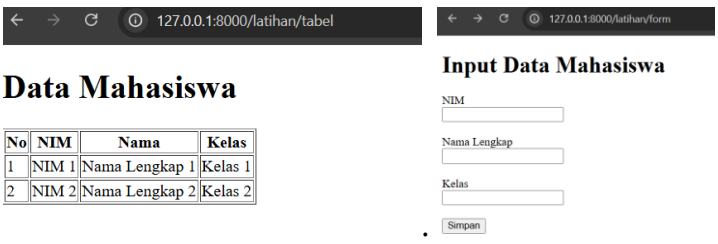
Gambar 1.9: Tangkapan layar Kode Sumber routes/web.php di *Code Editor* Anda.

Penjelasan (Best Practice - Clean Routing):

- **Eksplisit *Controller Action***: Penggunaan sintaks [LatihanController::class, 'getTabel'] adalah *best practice* yang meningkatkan **Maintainability** dan memungkinkan *IDE* mendeteksi kesalahan lebih awal.
- **Named Routes:** Setiap *route* diberi nama (latihan.tabel, latihan.form). Ini adalah *best practice* yang menjaga Kemudahan Interaksi di sisi pengembangan, karena tautan dalam aplikasi dapat direferensikan menggunakan nama, bukan URL yang kaku.

3.3. Verifikasi Akhir

Verifikasi ini membuktikan bahwa seluruh komponen (Route → Controller → View) bekerja harmonis, memastikan program berjalan tanpa *error* runtime.



Gambar 2.0: Tampilan Tabel Mahasiswa dan form

BAB IV
PENUTUP

5.1. Kesimpulan

Dokumentasi portofolio ini secara komprehensif membuktikan penguasaan terhadap Unit Kompetensi Menulis Kode Dengan Prinsip Sesuai Guidelines dan Best Practice melalui proyek pengembangan aplikasi sederhana menggunakan Laragon dan framework PHP (Laravel).