MODUL PELATIHAN MEMBUAT APLIKASI BERBASIS CRUD MENGGUNAKAN LARAVEL

BY : MILLAN

Linkedin : https://www.linkedin.com/in/awal-millan-syahid-48841634b

Instagram : @awalmillansyahid

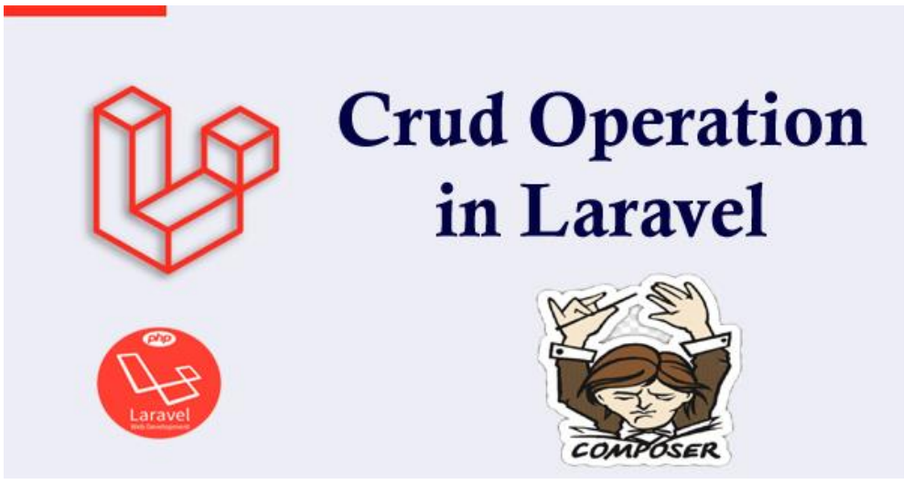
DAFTAR ISI

INTRODUCTION INTO BACKEND FRAMEWORK CRUD OPERATION, SECURITY AND SCALABILITY

Modul Praktikum by Millan

BACKEND ROLE (FULL STACK LARAVEL)

Dalam tutorial ini, kita akan mempelajari operasi mentah yang sangat mendasar dengan Laravel. Kita akan ditunjukkan langkah demi langkah dari awal sampai jadi, tujuan dari tutorial ini adalah kita akan lebih memahami cara kerja laravel.



CRUDApp Laravel adalah aplikasi berbasis Laravel yang mengimplementasikan operasi **CRUD** (Create, Read, Update, Delete). CRUD merupakan konsep dasar dalam pengelolaan data dalam database yang digunakan dalam hampir semua aplikasi web.**Penjelasan CRUD dalam Laravel**

1. **Create** → Menambahkan data baru ke database
2. **Read** → Menampilkan atau mengambil data dari database
3. **Update** → Memperbarui data yang sudah ada
4. **Delete** → Menghapus data dari database

Komponen dalam CRUDApp Laravel

Untuk membuat aplikasi CRUD di Laravel, biasanya melibatkan beberapa komponen berikut:

1. **Routing (routes/web.php)** → Menentukan URL endpoint untuk setiap operasi CRUD.
2. **Controller** → Menangani logika CRUD (mengelola request dan response).
3. **Model** → Berinteraksi dengan database menggunakan Eloquent ORM.
4. **View (Blade Template)** → Menampilkan data dalam bentuk antarmuka pengguna.
5. **Migration & Seeder** → Membantu dalam pembuatan dan pengisian database awal.

Contoh Implementasi CRUD Sederhana di Laravel

1. Membuat Model & Migration

```
bash
php artisan make:model Post -m
```

Lalu, edit file migration di database/migrations/xxxx_xx_xx_xxxxxx_create_posts_table.php

```php
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->string('title');
        $table->text('content');
        $table->timestamps();
    });
}
```

Jalankan migration:

```bash
php artisan migrate
```

2. Membuat Controller

```bash
php artisan make:controller PostController --resource
```

Isi controller app/Http/Controllers/PostController.php:

```php
use App\Models\Post;
use Illuminate\Http\Request;

class PostController extends Controller
{
    public function index()
    {
        $posts = Post::all();
        return view('posts.index', compact('posts'));
    }

    public function create()
    {
        return view('posts.create');
    }

    public function store(Request $request)
    {
        Post::create($request->validate([
            'title' => 'required',
            'content' => 'required',
        ]));
```

```
            return redirect()->route('posts.index')-
     >with('success', 'Post berhasil ditambahkan');
        }

        public function edit(Post $post)
        {
            return view('posts.edit', compact('post'));
        }

        public function update(Request $request, Post $post)
        {
            $post->update($request->validate([
                'title' => 'required',
                'content' => 'required',
            ]));

            return redirect()->route('posts.index')-
     >with('success', 'Post berhasil diperbarui');
        }

        public function destroy(Post $post)
        {
            $post->delete();
            return redirect()->route('posts.index')-
     >with('success', 'Post berhasil dihapus');
        }
}
```

3. Menambahkan Routing di routes/web.php

```
php
use App\Http\Controllers\PostController;

Route::resource('posts', PostController::class);
```

4. Membuat Blade Templates (resources/views/posts/)
   - **index.blade.php** → Menampilkan daftar post
   - **create.blade.php** → Form untuk menambah post
   - **edit.blade.php** → Form untuk mengedit post

Setelah itu, jalankan server Laravel:

```
bash
php artisan serve
```

Lalu akses http://127.0.0.1:8000/posts.

Langkah 1 : Install Laravel versi terbaru

Langkah awal yang kita lakukan adalah menginstall XAMPP, PHP, dan Composer.

1. **XAMPP**

   XAMPP adalah paket perangkat lunak yang menyediakan **Apache, MySQL (atau MariaDB), PHP, dan Perl** dalam satu instalasi. Fungsinya:

   - **Apache** → Server web yang digunakan untuk menjalankan aplikasi berbasis PHP secara lokal.
   - **MySQL/MariaDB** → Database yang digunakan untuk menyimpan data aplikasi Laravel.
   - **PHP** → Bahasa pemrograman yang digunakan Laravel.
   - **phpMyAdmin** → Alat berbasis web untuk mengelola database MySQL/MariaDB.

   Untuk link bisa didownload pada Alamat berikut :

   https://www.apachefriends.org/download.html

2. **PHP**

   PHP (Hypertext Preprocessor) adalah bahasa pemrograman yang digunakan oleh Laravel. Laravel sendiri adalah framework PHP, jadi kita harus menginstal PHP agar Laravel dapat berjalan.

   Untuk PHP nya sendiri kitab isa instal divisual studio code dibagian extensions

3. **Composer**

   Composer adalah manajer paket untuk PHP yang digunakan untuk mengelola dependensi dalam proyek Laravel. Dengan Composer, kita bisa:

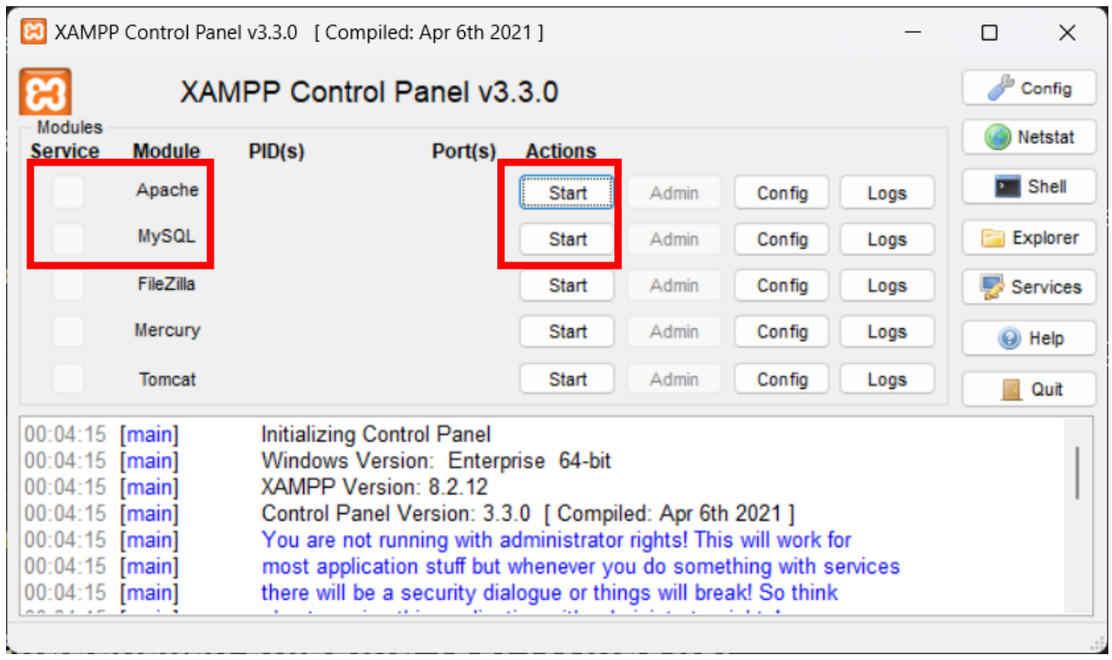
   - Menginstal Laravel dan pustaka lain dengan mudah.
   - Mengupdate dependensi proyek Laravel tanpa perlu mengunduh secara manual.
   - Mengelola autoloading sehingga kode lebih efisien.

   Untuk Composer bisa kita download disini : https://getcomposer.org/download/

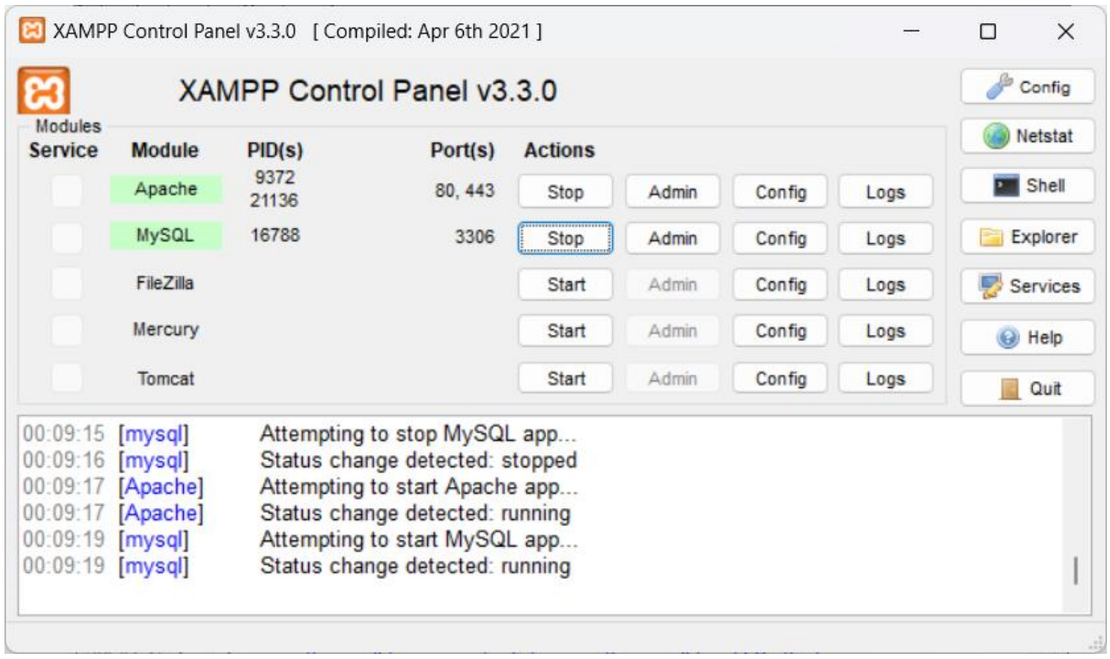
sebelum melakukan instalasi pastikan kita telah membuat folder untuk menyimpan projeknya dengan nama pilihan kita sendiri seperti contoh berikut :

D:\millan\documents\e-learning myskill\web development\Back-End Development Introduction\millan

Setelah semuanya selesai maka selanjutnya kita akan membuka XAMPP dan menyalakan server MySQL dan Apache nya dengan cara mengklik tombol start seperti berikut :



Jika berhasil dinyalakan maka tampilannya akan seperti berikut :



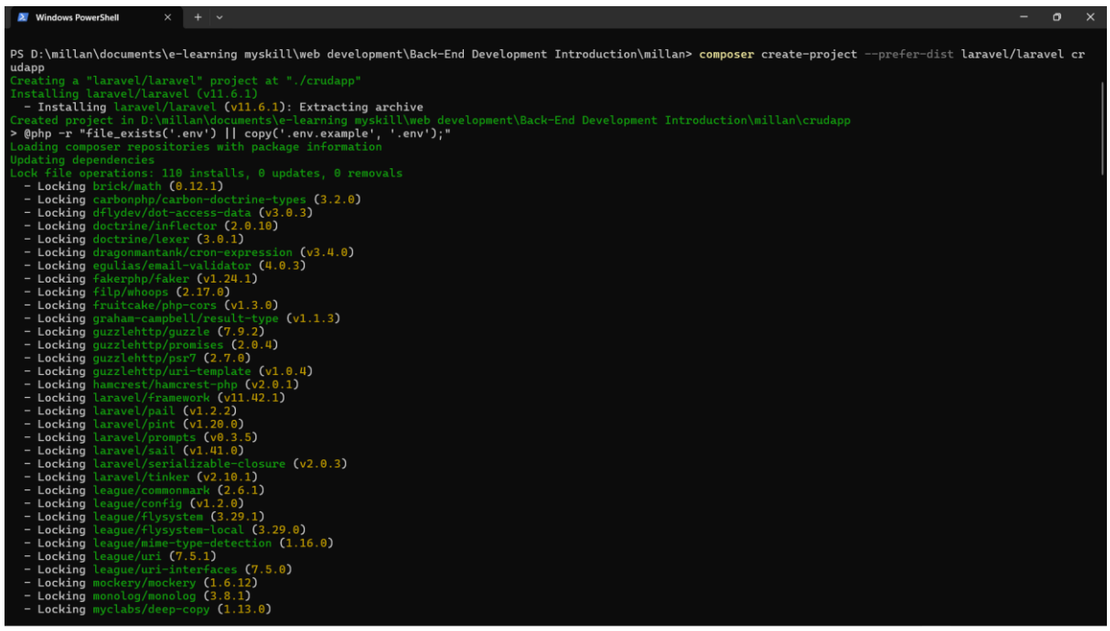
Lalu kita jalan perintah berikut dibawah ini untuk memasang Laravel pada aplikasi kita.

```
Windows PowerShell
composer create-project --prefer-dist laravel/laravel crudapp
```

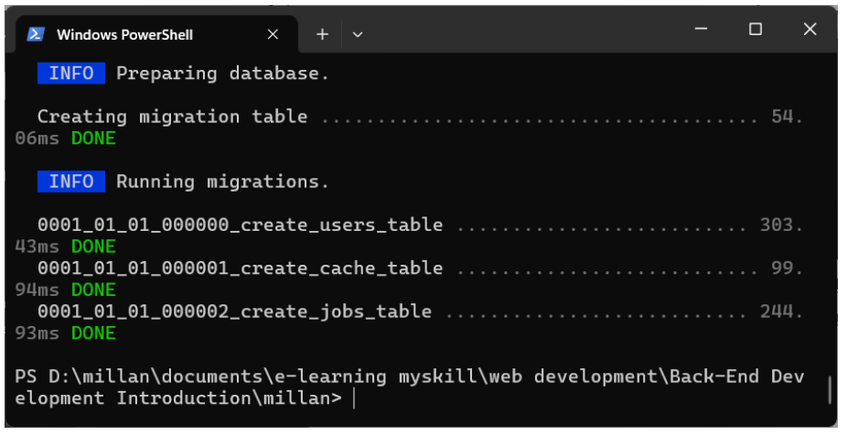
Untuk menjalankan perintah diatas kita bisa langsung membuka difolder tempat kita akan menyimpan projek CRUD kita seperti berikut :

D:\millan\documents\e-learning myskill\web development\Back-End Development Introduction\millan

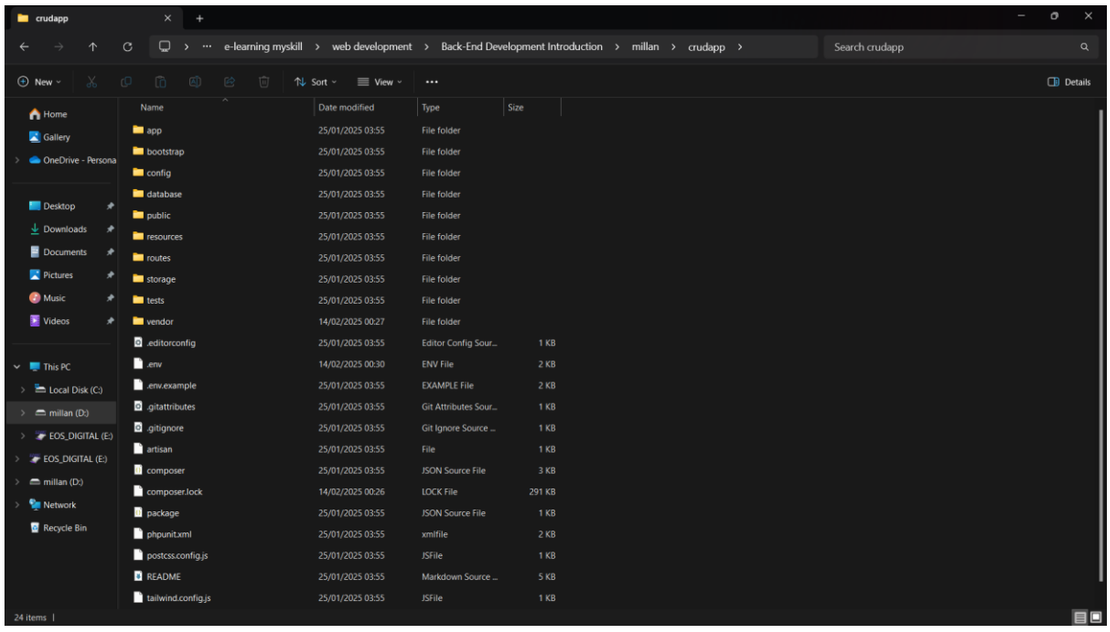
Setelah kita berada didalam folder projek kita, Langkah selanjutnya kita **klik kanan → Open in Terminal**, lalu masukan perintah seperti berikut pada terminalnya.



Lalu tekan enter dan biarkan programnya berjalan seperti diatas, tunggu beberapa menit sampai proses instalasinya selesai.Jika selesai tampilan terminalnya seperti ini :



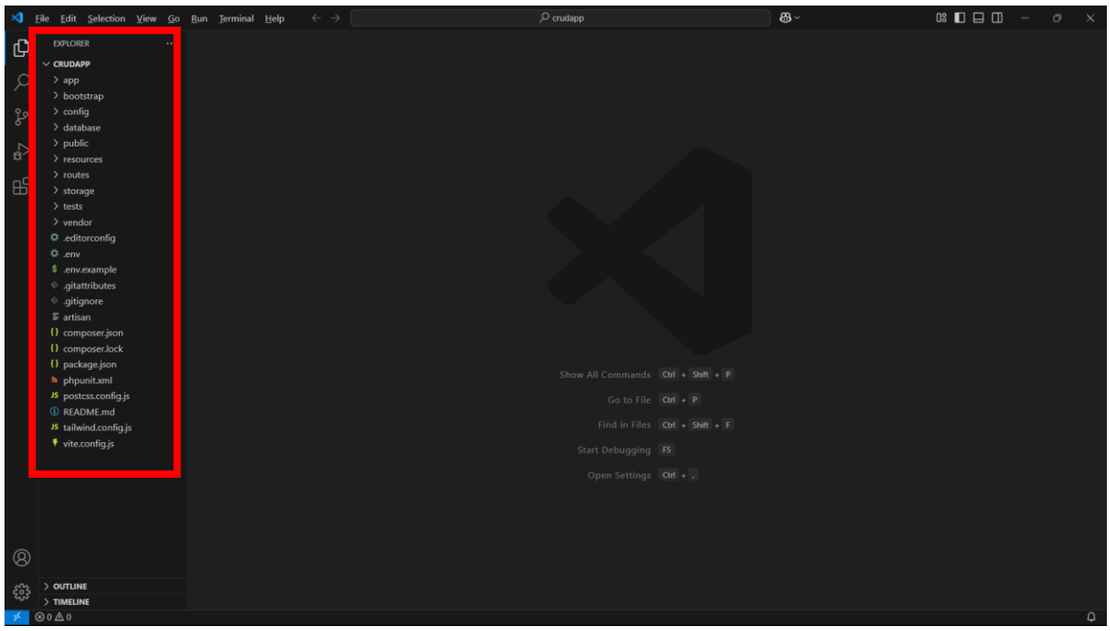
Selanjutnya kita cek ditempat folder kita menyimpan projek untuk memastikan apakah Laravel sudah terinstall atau belum, jika Laravel sudah terinstal dalam folder projek kita maka akan ada folder Bernama crudapp dan berisi seperti ini :



Maka untuk selanjutnya kita membuka visual studio code, bagi yang belum menginstall visual studio code bisa diinstall terlebih dahulu, perlu diingat kita menggunakan Visual Studio Code dengan warna logo biru bukan ungu.

Jika sudah membuka Visual Studio Code kita akan membuka folder CRUDApp nya dengan cara Klik → Open Folder → cari folder Dimana kita menyimpan projek CRUDApp seperti dibawah ini :

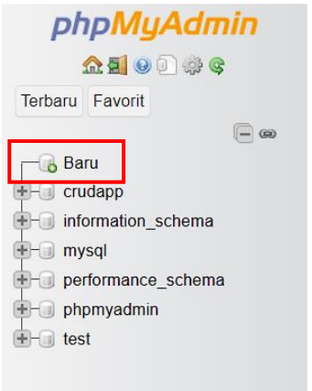
D:\millan\documents\e-learning myskill\web development\Back-End Development Introduction\millan



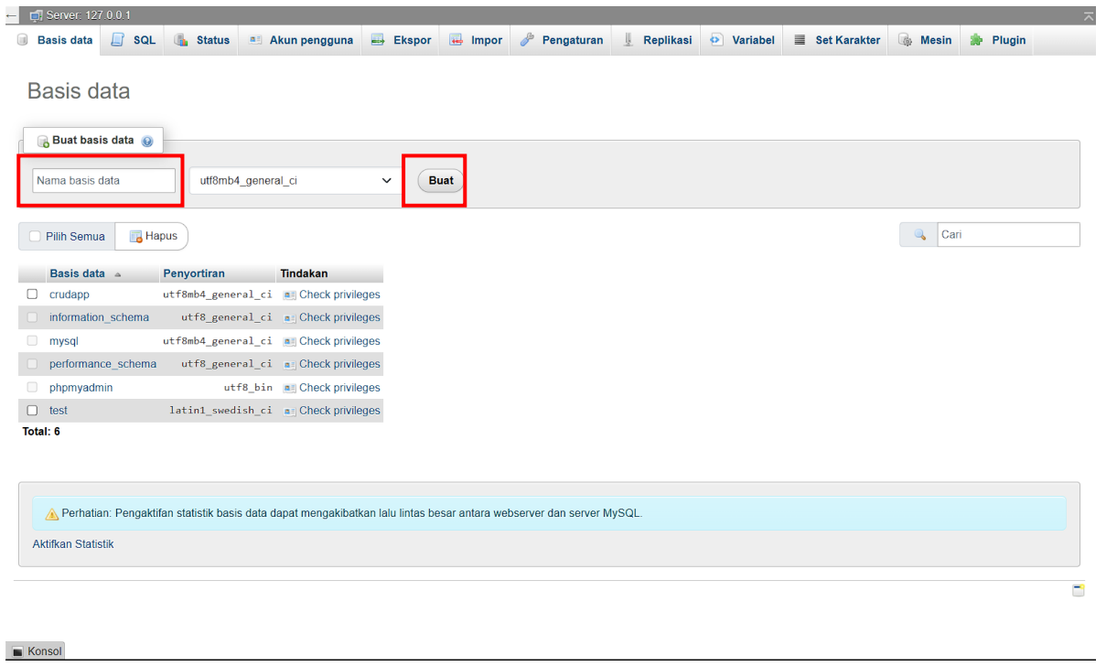
Jika kita sudah membuka foldernya maka tampilannya akan seperti ini pada Visual Studio Code, bis akita lihat di tab samping kiri EXPLORE itu sudah ada isi dari semua folder projek CRUDApp.

Langkah 2 : Konfigurasi Basis Data

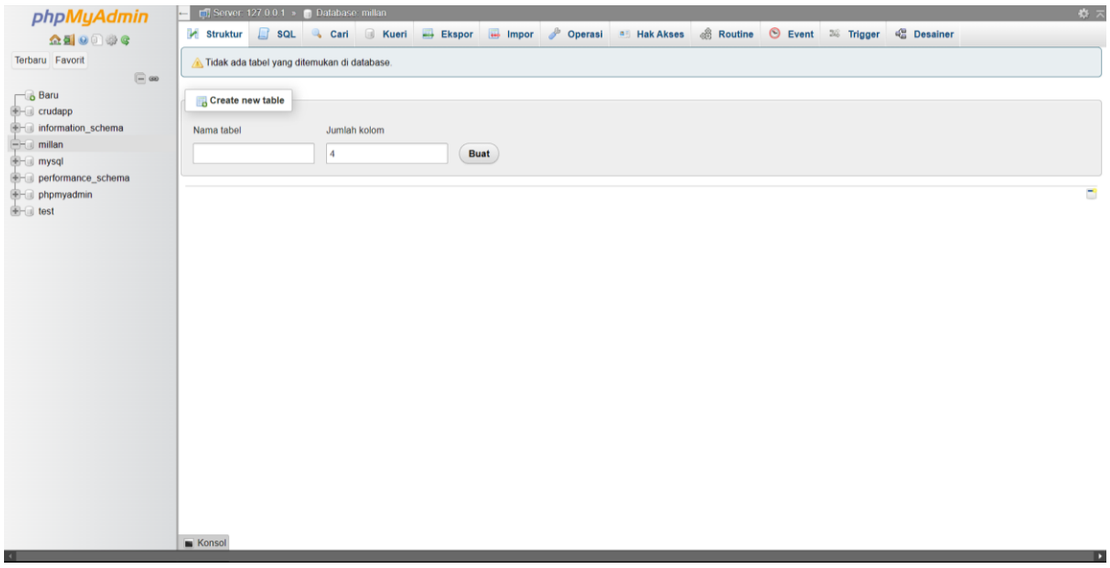
Selanjutnya kita hapus file .env, lalu kita akan membuat databasenya terlebih dahulu dengan cara membuka browser lalu masukan alamat berikut kekolom pencarian http://localhost/phpmyadmin/ selanjutnya kita klik tombol baru pada samping kiri menu tab phpMyAdmin.



Maka tampilannya akan seperti ini, selanjutnya kita masukkan nama database yang akan kita buat dikolom nama basis data setelah itu kita klik tombol buat.



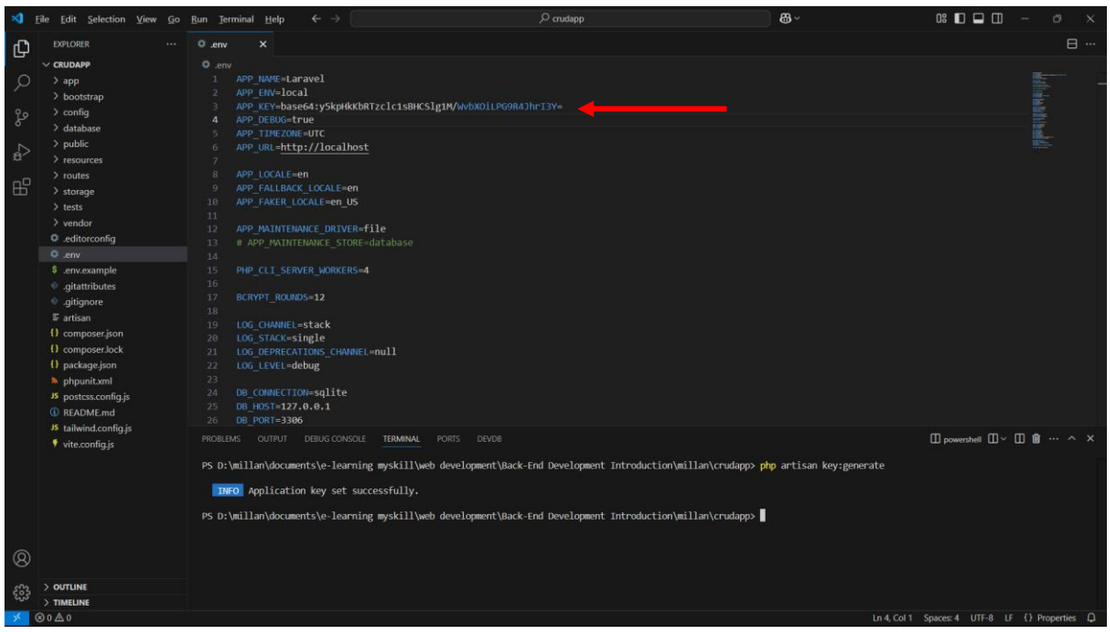
Setelah selesai dibuat maka tampilannya akan seperti ini.



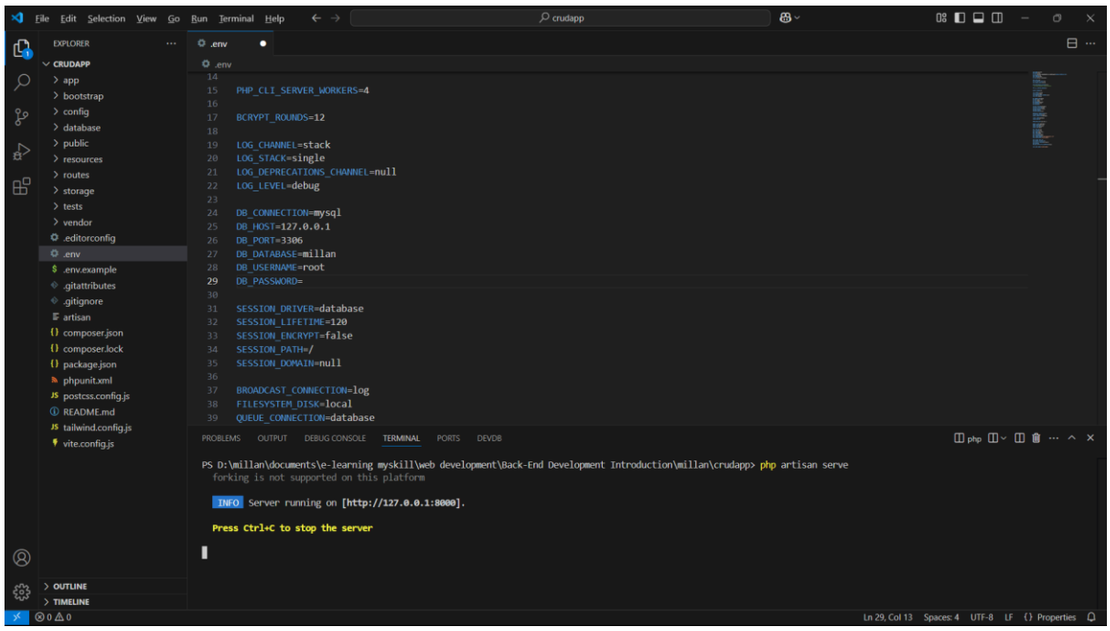
Selanjutnya kita akan mengcopy file .env.example didalam folder rootnya dengan cara memasukkan kode programnya diterminal cp .env.example .env seperti dibawah ini.

Maka file .env.example berhasil tersalin dengan nama .env, fungsi .env ini adalah untuk menyambungkan Laravel kita ke Database yang kita buat sebelumnya. selanjutnya kita buka file .env yang disalin tadi. Jika sudah terbuka cari (DB_DATABASE=Laravel) dan ganti dengan (DB_DATABASE=millan), nama millan adalah nama yang kita berikan pada pembuatan database sebelumnya.

Langkah selanjutnya kita akan membuat (APP_KEY=) untuk keamanan Laravel kita, untuk membuatnya kita memasukan kode program berikut keterminal php artisan key:generate jika berhasil tampilannya akan seperti ini.



(APP_KEY=base64:y5kpHkKbRTzclc1sBHCSlg1M/WvbXOiLPG9R4JhrI3Y=) APP Key telah terisi. Selanjutnya pastikan (DB_CONNECTION=mysql) terisi dengan "mysql" dan (DB_PORT=3306) terisi angka port 3306 sesuai dengan nomor port database kita. Selanjutnya kita jalankan Laravel kita dengan memasukkan kode program php artisan serve



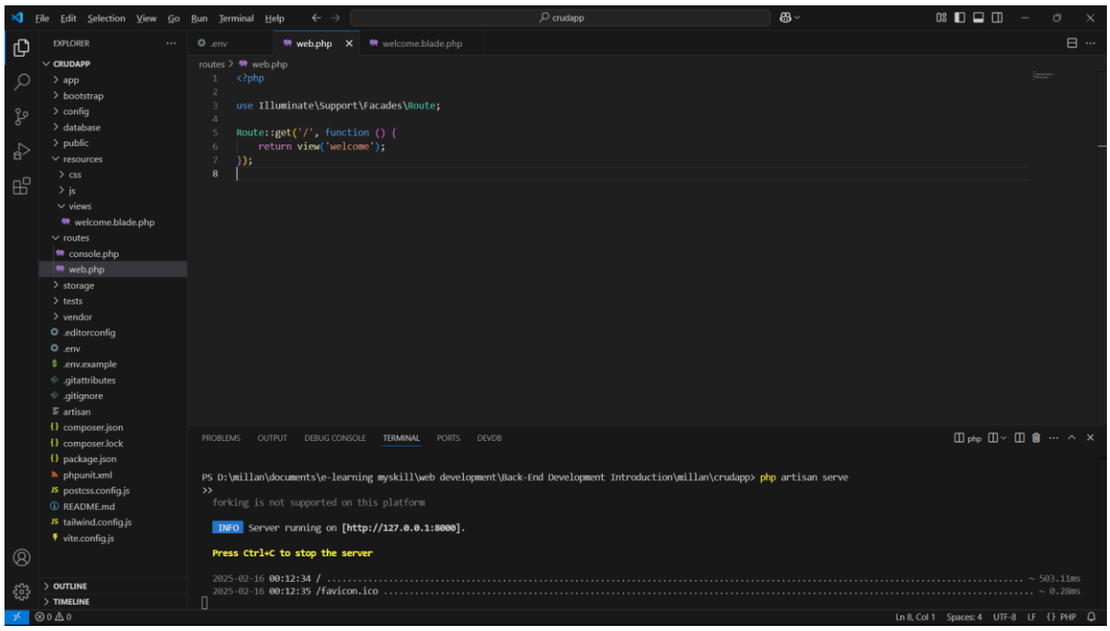
Jika berhasil akan ada indikasi INFO Server running on [http://127.0.0.1:8000]. Sebelum itu pastikan pada file .env tidak ada tanda # awal disetiap nama value dalam .env.

Langkah 3 : Buat Migrasi dan Tambahkan Rute Sumber Daya

setelah itu jalankan perintah migrasi pada terminal php artisan migrate setelah berhasil dijalankan kita buka file web.php di folder routes lalu masukan kode program ini pada bagian konsolnya.

```
php
<?php

use Illuminate\Support\Facades\Route;

Route::get('/', function () {
```

```
    return view('welcome');
});
```



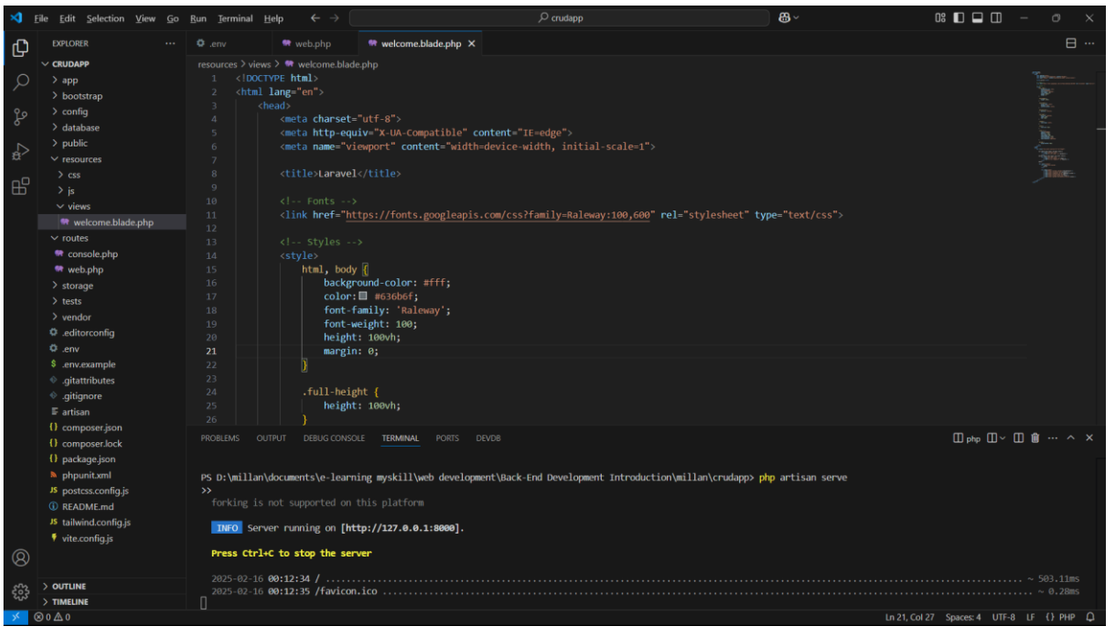
Maka tampilannya seperti gambar berikut. Jika sudah kita buka file berikutnya pada folder resources → views → welcome.blade.php, lalu masukkan kode berikut kekonsol.

```
php
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-
scale=1">

        <title>Laravel</title>

        <!-- Fonts -->
                                                                <link
href="https://fonts.googleapis.com/css?family=Raleway:100,600"
rel="stylesheet" type="text/css">

        <!-- Styles -->
        <style>
            html, body {
                background-color: #fff;
                color: #636b6f;
                font-family: 'Raleway';
```

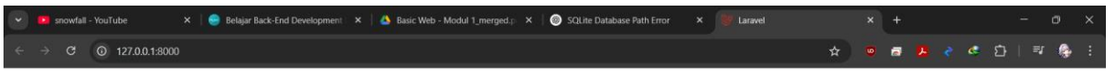
```
            font-weight: 100;
            height: 100vh;
            margin: 0;
        }

        .full-height {
            height: 100vh;
        }

        .flex-center {
            align-items: center;
            display: flex;
            justify-content: center;
        }

        .position-ref {
            position: relative;
        }

        .top-right {
            position: absolute;
            right: 10px;
            top: 18px;
        }

        .content {
            text-align: center;
        }

        .title {
            font-size: 84px;
        }

        .links > a {
            color: #636b6f;
            padding: 0 25px;
            font-size: 12px;
            font-weight: 600;
            letter-spacing: .1rem;
            text-decoration: none;
            text-transform: uppercase;
        }
```

```
            .m-b-md {
                margin-bottom: 30px;
            }
        </style>
    </head>
    <body>
        <div class="flex-center position-ref full-height">

            @if (Route::has('login') && Auth::check())
                <div class="top-right links">
                    <a href="{{ url('/home') }}">Dashboard</a>
                </div>
            @elseif (Route::has('login') && !Auth::check())
                <div class="top-right links">
                    <a href="{{ url('/login') }}">Login</a>
                    <a href="{{ url('/register') }}">Register</a>
                </div>
            @endif

            <div class="content">
                <div class="title m-b-md">
                    Laravel
                </div>

                <div class="links">
                                                                    <a
href="https://laravel.com/docs">Documentation</a>
                    <a href="https://laracasts.com">Laracasts</a>
                      <a href="https://codecasts.com.br">CODECASTS
[pt-BR]</a>
                    <a href="https://laravel-news.com">News</a>
                    <a href="https://forge.laravel.com">Forge</a>
                                                                    <a
href="https://github.com/codecasts/laravel">GitHub</a>
                </div>
            </div>
        </div>
    </body>
</html>
```

Jika sudah diinput kodenya maka tampilannya akan seperti dibawah ini.



Selanjutnya kita klik alamat link pada indikasi diterminal dengan cara tahan tombol ctr+klik kiri pada bagian alamat link tersebut INFO Server running on [http://127.0.0.1:8000]. Jika berhasil tampilannya akan seperti dibawah ini. Pastikan tidak ada error yang terjadi pada tampilannya.



Langkah 4 : Tambahkan Pengontrol dan Model

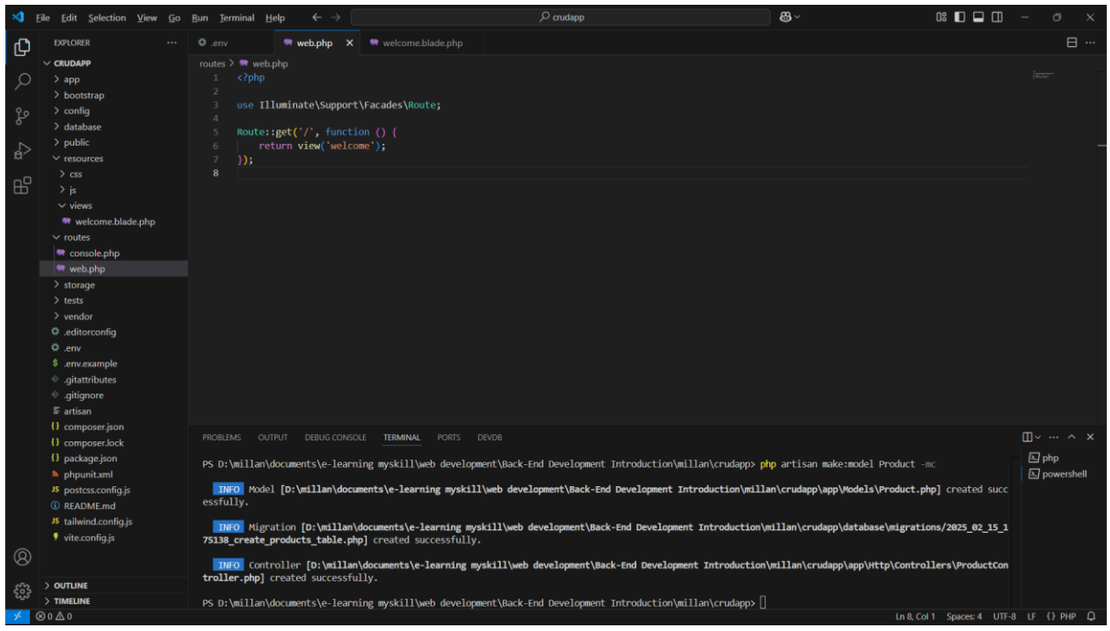
1. **Menambahkan Product Table**

   Selanjutnya kita tambahkan terminal baru supaya tidak mengganggu terminal antar satu sama lain, dengan cara sebagai berikut klik tombol + pada tab terminal disamping kanan bawah seperti gambar dibawah ini.

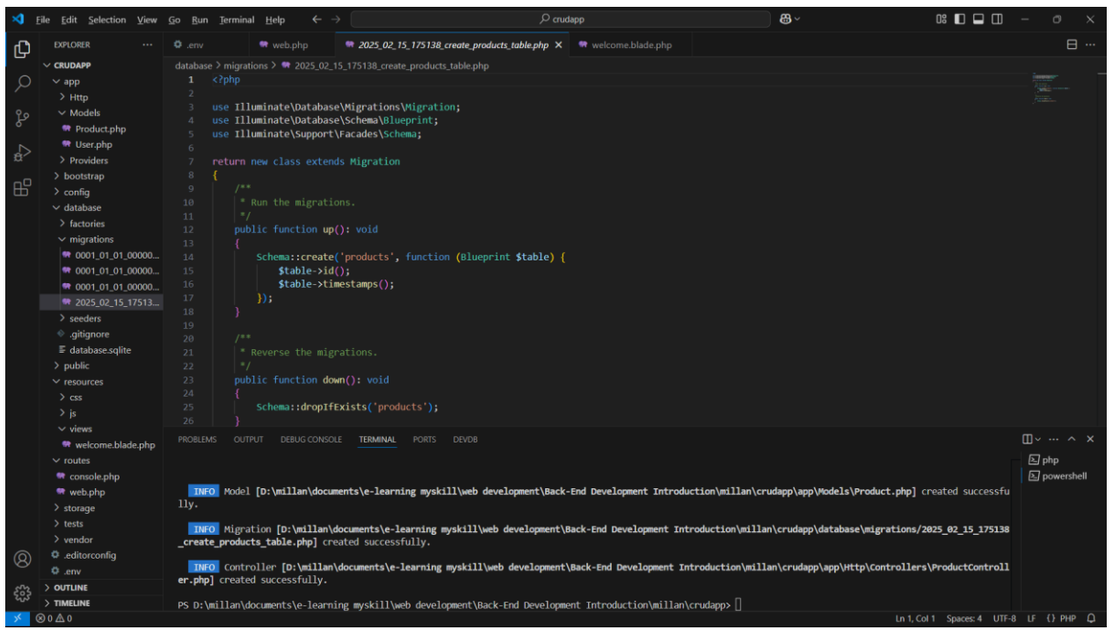
   Selanjutnya kita akan membuat file controller untuk Laravelnya, **Controller** berfungsi sebagai penghubung antara **model (database)** dan **view (tampilan)**. Dengan menggunakan

controller, kita dapat mengorganisir logika aplikasi dalam satu tempat, sehingga kode lebih rapi dan mudah dikelola.

Langsung saja kita masukkan kode perintahnya diterminal seperti ini php artisan make:model Product -mc lalu tunggu beberapa saat, jika berhasil hasilnya seperti dibawah ini.



Maka folder productnya sudah kita buat, Langkah selanjutnya kita buka file 2025_02_15_175138_create_products_table.php di folder database → migrations → 2025_02_15_175138_create_products_table.php. Setelah berhasil dibuka maka tampilannya seperti berikut.



Setelah itu kita masukkan kode program berikut pada bagian konsol file 2025_02_15_175138_create_products_table.php

```
php
<?php


use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
```

```
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('products', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('description');
            $table->text('image');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('products');
    }
};
```

Kode di atas adalah **migration** di Laravel yang digunakan untuk membuat tabel products di database. Berikut adalah penjelasan rinci dari setiap bagian:

1. **Struktur Dasar Migration**

```
php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
```

- **Migration** → Class dasar untuk migrasi database.
- **Blueprint** → Digunakan untuk mendefinisikan struktur tabel.
- **Schema** → Fasilitas untuk membuat dan mengubah tabel di database

2. **Fungsi up() → Membuat Tabel**

```
php
public function up(): void
```

```
    {
        Schema::create('products', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('description');
            $table->text('image');
            $table->timestamps();
        });
    }
```

- **Schema::create('products', function (Blueprint $table) {...})** → Membuat tabel products jika belum ada.
- **$table->id();** → Membuat kolom id sebagai primary key (auto-increment).
- **$table->string('name');** → Kolom name, tipe VARCHAR(255).
- **$table->string('description');** → Kolom description, sebaiknya pakai text() untuk dekodesi panjang.
- **$table->text('image');** → Kolom image, tipe TEXT, untuk menyimpan URL atau path gambar.
- **$table->timestamps();** → Menambahkan created_at dan updated_at otomatis.

3. **Fungsi down() → Menghapus Tabel**

```
php
public function down(): void
{
    Schema::dropIfExists('products');
}
```

- **Schema::dropIfExists('products');** → Menghapus tabel products jika ada (untuk rollback migration).

Lalu setelah itu kita upload data product tablenya ke database dengan cara memasukkan perintah berikut keterminal “php artisan migrate”.

```
php
php artisan migrate
```

2. **Menambahkan Product Controller**

Selanjutnya kita akan membuat model baru untuk table costumer dengan cara memasukan perintah ini keterminal “php artisan make:migration create_costumers_table”

```
php
php artisan make:migration create_costumers_table
```

Setelah Langkah-langkah migrasi telah kita lakukan selanjutnya kita akan menambahkan product controller untuk digunakan menangani logika bisnis terkait produk dalam aplikasi. Ini adalah bagian dari **MVC (Model-View-Controller)**, di mana **controller** bertanggung jawab untuk mengelola request dari user dan menghubungkannya dengan model serta view.

Kita bisa salin kode program dibawah berikut lalu dimasukkan kedalam file Bernama "ProductController.php" yang berlokasikan di app → Http/Controllers → ProductController.php.

```
php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Product;

class ProductController extends Controller
{
    public function index()
    {
        $products = Product::all();
        return view('products.index', compact('products'));
    }

    public function add()
    {
        return view('products.add');
    }

    public function save(Request $request)
    {
        $request->validate([
            'name' => 'required|string',
            'description' => 'required|string',
        ]);

        $product = new Product();
        $product->name = $request->input('name');
        $product->description = $request->input('description');
        $product->image = $request->input('image');
        $product->save();

       return redirect()->route('products.index')->with('success',
'Produk berhasil disimpan.');
    }

    public function edit($id)
```

```
    {
        $product = Product::findOrFail($id);
        return view('products.edit', compact('product'));
    }

    public function update(Request $request, $id)
    {
        $request->validate([
            'name' => 'required|string',
            'description' => 'required|string'
        ]);

        $product = Product::findOrFail($id);
        $product->name = $request->input('name');
        $product->description = $request->input('description');
        $product->image = $request->input('image');
        $product->save(); // Simpan perubahan

       return redirect()->route('products.index')->with('success',
'Produk berhasil diedit.');
    }

    public function delete($id)
    {
        $product = Product::findOrFail($id);
        $product->delete(); // Hapus produk

       return redirect()->route('products.index')->with('success',
'Produk berhasil dihapus.');
    }
}
```

Kode **ProductController** dalam Laravel yang digunakan untuk mengelola produk dalam aplikasi berbasis web. Berikut penjelasan kode secara rinci :

1. **Namespace dan Penggunaan Kelas**

```
php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Models\Product;
```

- namespace App\Http\Controllers; → Menandakan bahwa controller ini berada dalam folder app/Http/Controllers.

- use Illuminate\Http\Request; → Mengimpor kelas Request yang digunakan untuk menangani request dari pengguna.
- use App\Models\Product; → Mengimpor model Product yang berhubungan dengan database.

2. Deklarasi Kelas

```
php
class ProductController extends Controller
```

- ProductController adalah kelas yang merupakan turunan dari Controller, yang memungkinkan penggunaan fitur bawaan Laravel.

3. Fungsi dalam Controller

Controller ini memiliki beberapa fungsi utama:

a. index() - Menampilkan Semua Produk

```
php
public function index()
{
    $products = Product::all(); // Mengambil semua data
produk dari database
    return view('products.index', compact('products'));
}
```

- Product::all(); → Mengambil semua data dari tabel products.
- return view('products.index', compact('products')); → Menampilkan tampilan products.index dengan data products yang bisa digunakan di dalam view.

b. add() - Menampilkan Form Tambah Produk

```
php
public function add()
{
    return view('products.add');
}
```

- Fungsi ini hanya mengembalikan tampilan form untuk menambahkan produk baru (products.add).

c. save(Request $request) - Menyimpan Produk Baru

```
php
public function save(Request $request)
{
    $request->validate([
        'name' => 'required|string',
        'description' => 'required|string',
```

```
    ]);
```

- Request $request → Menerima input dari user.
- $request->validate([...]) → Melakukan validasi input agar tidak kosong dan sesuai tipe data.

```
php
    return redirect()->route('products.index')-
>with('success', 'Produk berhasil disimpan.');
}
```

- redirect()->route('products.index') → Mengarahkan kembali ke halaman daftar produk setelah menyimpan.
- with('success', 'Produk berhasil disimpan.') → Menyimpan pesan sukses ke session.

d. edit($id) - Menampilkan Form Edit Produk

```
php
public function edit($id)
{
    $product = Product::findOrFail($id);
    return view('products.edit', compact('product'));
}
```

- Product::findOrFail($id); → Mencari produk berdasarkan id, jika tidak ditemukan akan mengembalikan error 404.
- return view('products.edit', compact('product')); → Mengirim data produk ke view products.edit.

e. update(Request $request, $id) - Menyimpan Perubahan Produk

```
php
public function update(Request $request, $id)
{
    $request->validate([
        'name' => 'required|string',
        'description' => 'required|string'
    ]);
```

- Validasi input agar data sesuai dengan format yang diharapkan.

```
php
    $product = Product::findOrFail($id);
    $product->name = $request->input('name');
    $product->description = $request-
>input('description');
    $product->image = $request->input('image');
    $product->save(); // Simpan perubahan
```

- Mengambil data produk berdasarkan id dan memperbarui dengan data yang baru.

```
php
        return redirect()->route('products.index')-
>with('success', 'Produk berhasil diedit.');
}
```

- Mengarahkan kembali ke daftar produk dengan pesan sukses.

f. delete($id) - Menghapus Produk

```
php
public function delete($id)
{
    $product = Product::findOrFail($id);
    $product->delete(); // Hapus produk
```

- Mencari produk berdasarkan id dan menghapusnya dari database.

```
php
    return redirect()->route('products.index')-
>with('success', 'Produk berhasil dihapus.');
}
```

- Mengarahkan kembali ke daftar produk dengan pesan sukses.

Kesimpulan

Fungsi	Keterangan
index()	Menampilkan daftar produk
add()	Menampilkan form tambah produk
save(Request $request)	Menyimpan produk baru
edit($id)	Menampilkan form edit produk
update(Request $request, $id)	Memperbarui data produk
delete($id)	Menghapus produk

Controller ini berguna untuk mengelola CRUD produk di Laravel dengan routing seperti :

```
php
    Route::get('/products', [ProductController::class, 'index'])-
>name('products.index');
Route::get('/products/add', [ProductController::class, 'add'])-
>name('products.add');
Route::post('/products/save', [ProductController::class, 'save'])-
>name('products.save');
Route::get('/products/edit/{id}', [ProductController::class,
'edit'])->name('products.edit');
Route::post('/products/update/{id}', [ProductController::class,
'update'])->name('products.update');
```

```
Route::delete('/products/delete/{id}', [ProductController::class,
'delete'])->name('products.delete');
```

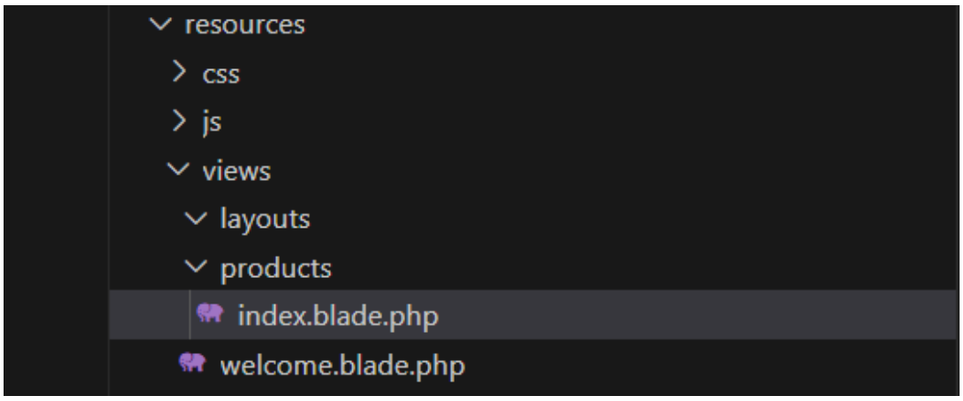
Ini adalah implementasi sederhana dari **CRUD (Create, Read, Update, Delete)** di Laravel menggunakan controller.

Langkah 5 : Menambahkan File Blade

Selanjutnya Langkah terakhir adalah menambahkan file blade pada framework kita. File **Blade** di Laravel adalah **template engine** yang digunakan untuk merancang tampilan (view) dalam framework Laravel. Fungsinya adalah untuk memudahkan pengelolaan tampilan dengan sintaks yang lebih sederhana dibandingkan PHP murni, serta mendukung fitur seperti inheritance, component, dan directive.

1. **Menambahkan index.blade.php**

   Sebelum memulai Langkah ini kita diharuskan untuk membuat folder khusus untuk menyimpan file blade ini secara terpisah dari yang lain. File blade yang pertama kita buat adalah file index.blade.php yang akan kita simpan difolder products yang berlokasi pada resources → views → products → index.blade.php. Kita tambahkan saja file dengan cara klik kanan pada folder yang baru saja kita buat yaitu products lalu klik new file dan beri nama index.blade.php. Maka hasilnya akan seperti dibawah ini.



   Kemudian masukkan kode program ini ke dalam console index.blade.php.

```
blade
@extends('layouts.master')

@section('content')
<section class="container mt-4">
    <div class="d-flex justify-content-between align-items-
center mb-3">
        <h2>Daftar Produk</h2>
        <a href="{{ route('product.add') }}" class="btn btn-
primary">
            + Tambah Produk
        </a>
    </div>

    <div class="table-responsive">
```

```
        <table class="table table-bordered table-striped">
            <caption>Daftar Produk</caption>
            <thead class="table-dark">
                <tr>
                    <th>No</th>
                    <th>Nama Produk</th>
                    <th>Dekodesi Produk</th>
                    <th>Gambar</th>
                    <th>Aksi</th>
                </tr>
            </thead>
            <tbody>
                @forelse ($products as $key => $item) {{-- Ganti
$product dengan $products --}}
                    <tr>
                        <td>{{ $key + 1 }}</td>
                        <td>{{ $item->name }}</td> {{--
Sesuaikan dengan kolom database --}}
                        <td>{{ $item->description }}</td> {{--
Sesuaikan dengan kolom database --}}
                        <td>
                            <img src="{{$item->image}}"
class="img-responsive" width="100" height="100"/>
                        </td>
                        <td>
                            <div class="d-flex gap-2">
                                <a href="{{
route('product.edit', $item->id) }}" class="btn btn-warning btn-
sm">
                                    Edit
                                </a>
                                <form action="{{
route('product.delete', $item->id) }}" method="POST"
onsubmit="return confirm('Apakah Anda yakin ingin menghapus
produk ini?')">
                                    @csrf
                                    @method('DELETE')
                                    <button type="submit"
class="btn btn-danger btn-sm">
                                        Hapus
                                    </button>
                                </form>
```

```
                            </div>
                        </td>
                    </tr>
                @empty
                    <tr>
                        <td colspan="5" class="text-center">Data
kosong</td>
                    </tr>
                @endforelse
            </tbody>
        </table>
    </div>
</section>
@stop
```

Jika sudah memasukkan kode program tersebut jangan lupa untuk menyimpannya. Kode di atas adalah file **Blade** di Laravel yang digunakan sebagai tampilan untuk halaman **Daftar Produk**. Berikut adalah penjelasan fungsi dan kegunaannya secara detail:

1. **Menggunakan Template Utama**

```blade
@extends('layouts.master')
```

- **@extends('layouts.master')** berarti file ini mewarisi atau menggunakan layout utama dari **layouts/master.blade.php**.
- Ini berguna agar struktur HTML utama seperti header, footer, dan styling tetap konsisten di seluruh halaman.

2. **Mendefinisikan Bagian content**

```blade
@section('content')
```

- **@section('content') ... @stop** digunakan untuk mengisi bagian content yang ada di **layouts.master**.

3. **Membuat Container untuk Halaman**

```blade
<section class="container mt-4">
```

- Menggunakan **Bootstrap** (container mt-4) untuk membuat tata letak rapi dengan margin atas (mt-4).

4. **Header dan Tombol Tambah Produk**

```
blade
<div class="d-flex justify-content-between align-items-center
mb-3">
    <h2>Daftar Produk</h2>
    <a href="{{ route('product.add') }}" class="btn btn-
primary">
        + Tambah Produk
    </a>
</div>
```

- **Menampilkan Judul "Daftar Produk"**.
- **Tombol "Tambah Produk"** akan mengarah ke **route product.add**, yang merupakan halaman untuk menambahkan produk baru.

5. **Membuat Tabel Daftar Produk**

```
blade
<table class="table table-bordered table-striped">
    <caption>Daftar Produk</caption>
    <thead class="table-dark">
        <tr>
            <th>No</th>
            <th>Nama Produk</th>
            <th>Dekodesi Produk</th>
            <th>Gambar</th>
            <th>Aksi</th>
        </tr>
    </thead>
```

- **Membuat tabel Bootstrap** dengan style:
  - **table-bordered** → Memberikan border pada tabel.
  - **table-striped** → Memberikan warna bergantian pada baris tabel.
  - **table-dark** → Header tabel diberi warna gelap.
- **Menampilkan kolom**:
  - No (Nomor urut)
  - Nama Produk
  - Dekodesi Produk
  - Gambar
  - Aksi (Tombol Edit & Hapus)

6. **Menampilkan Data Produk Menggunakan @forelse**

```
blade
@forelse ($products as $key => $item)
    <tr>
```

```
        <td>{{ $key + 1 }}</td>
        <td>{{ $item->name }}</td>
        <td>{{ $item->description }}</td>
        <td>
            <img src="{{$item->image}}" class="img-responsive"
width="100" height="100"/>
        </td>
        <td>
            <div class="d-flex gap-2">
                <a href="{{ route('product.edit', $item->id)
}}" class="btn btn-warning btn-sm">
                    Edit
                </a>
                <form action="{{ route('product.delete',
$item->id) }}" method="POST" onsubmit="return confirm('Apakah
Anda yakin ingin menghapus produk ini?')">
                    @csrf
                    @method('DELETE')
                    <button type="submit" class="btn btn-
danger btn-sm">
                        Hapus
                    </button>
                </form>
            </div>
        </td>
    </tr>
@empty
    <tr>
        <td colspan="5" class="text-center">Data kosong</td>
    </tr>
@endforelse
```

- **Menggunakan @forelse** untuk menampilkan data dari variabel **$products**.
  - Jika **ada data produk**, maka akan ditampilkan dalam tabel.
  - Jika **tidak ada data produk**, akan menampilkan pesan "Data kosong".

Penjelasan Bagian-bagian Kode:

1. **@forelse ($products as $key => $item)**
   - **$products** adalah daftar produk dari controller.
   - **$key + 1** digunakan untuk nomor urut.
   - **$item->name dan $item->description** → Menampilkan nama dan dekodesi produk.
   - **Menampilkan gambar produk** dari URL yang ada di **$item->image**.
2. **Tombol Edit & Hapus Produk**

- **Edit** → Mengarah ke **route('product.edit', $item->id)** untuk mengedit produk.
- **Hapus**:
  - Menggunakan form POST dengan method DELETE.
  - **@csrf** → Token keamanan Laravel.
  - **@method('DELETE')** → Laravel mengubah request ini menjadi HTTP DELETE.
  - **Konfirmasi JavaScript** saat menghapus produk.

7. **Menutup @section**

```blade
@stop
```

- **Menutup bagian @section('content')**.

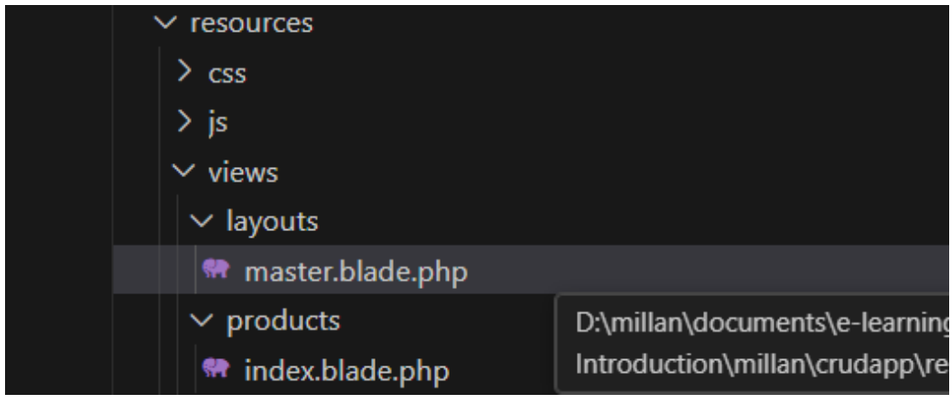
Kesimpulan

Kode ini adalah tampilan untuk **halaman daftar produk** dalam Laravel. Kegunaannya:

- **Menampilkan daftar produk dalam tabel**.
- **Memiliki fitur tambah produk, edit, dan hapus produk**.
- **Menggunakan Blade Template Engine** untuk membuat kode lebih rapi dan efisien.
- **Menggunakan Bootstrap** untuk tampilan yang lebih menarik dan responsif.

2. **Menambahkan master.blade.php**

Setelah manambahkan file index.blade.php Langkah selanjutnya kita menambahkan file bernama master.blade.php didalam folder layouts yang berlokasikan di resources → views → layouts. Caranya sama seperti menambahkan file index.blade.php hanya saja difolder yang beda.



Kemudian masukkan kode bootstrapnya kedalam console master.blade.php seperti dibawah ini.

```blade
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

```
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <meta name="csrf-token" content="{{ csrf_token() }}">
    <title>Tokoku - Tutorial CRUD Laravel</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/boot
strap.min.css">
    <link
href="https://fonts.googleapis.com/css2?family=Nunito:wght@400;60
0;700&display=swap" rel="stylesheet">
    <style>
        body {
            font-family: 'Nunito', sans-serif;
        }
    </style>
</head>
<body>

<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container">
        <a class="navbar-brand" href="#">Tokoku</a>
        <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-
label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse"
id="navbarSupportedContent">
            <ul class="navbar-nav ml-auto">
                <li class="nav-item active">
                    <a class="nav-link" href="{{ route('home')
}}">Home <span class="sr-only">(current)</span></a>
                </li>
            </ul>
        </div>
    </div>
</nav>

<main class="container mt-4">
    @if ($errors->any())
        <div class="alert alert-danger alert-dismissible fade
show" role="alert">
```

```
            <strong>Oops! Something went wrong:</strong>
            <ul>
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
            <button type="button" class="close" data-
dismiss="alert" aria-label="Close">
                <span aria-hidden="true">&times;</span>
            </button>
        </div>
    @endif

    @if (session('success'))
        <div class="alert alert-success alert-dismissible fade
show" role="alert">
            {{ session('success') }}
            <button type="button" class="close" data-
dismiss="alert" aria-label="Close">
                <span aria-hidden="true">&times;</span>
            </button>
        </div>
    @endif

    @if (session('error'))
        <div class="alert alert-danger alert-dismissible fade
show" role="alert">
            {{ session('error') }}
            <button type="button" class="close" data-
dismiss="alert" aria-label="Close">
                <span aria-hidden="true">&times;</span>
            </button>
        </div>
    @endif

    @yield('content')
</main>

<script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.m
in.js"></script>
```

```
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootst
rap.min.js"></script>
</body>
</html>
```

Kemudian simpan. Kode di atas adalah file **Blade layout utama** untuk aplikasi Laravel. File ini berfungsi sebagai **template dasar** yang digunakan oleh halaman-halaman lain dengan menggunakan @extends. Berikut adalah penjelasan setiap bagian dan kegunaannya:

1. **Struktur HTML Dasar**

```
blade
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
```

- **Menentukan tipe dokumen HTML5**.
- **Menentukan bahasa halaman** berdasarkan app()->getLocale(), yang diambil dari konfigurasi Laravel.

2. **Bagian <head>: Meta, CSS, dan Font**

```
blade
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <meta name="csrf-token" content="{{ csrf_token() }}">
    <title>Tokoku - Tutorial CRUD Laravel</title>
```

- **<meta charset="UTF-8">** → Menentukan encoding karakter.
- **<meta name="viewport">** → Agar responsif di perangkat mobile.
- **<meta http-equiv="X-UA-Compatible" content="ie=edge">** → Untuk kompatibilitas dengan Internet Explorer.
- **<meta name="csrf-token" content="{{ csrf_token() }}">**
  - Menyediakan **CSRF token** agar Laravel bisa memverifikasi request dari formulir.
- **<title>Tokoku - Tutorial CRUD Laravel</title>**
  - Menampilkan judul halaman.

```
blade
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bo
otstrap.min.css">
    <link
href="https://fonts.googleapis.com/css2?family=Nunito:wght@400;
600;700&display=swap" rel="stylesheet">
```

- **Menggunakan Bootstrap 4.5.2** untuk styling.
- **Menggunakan font Nunito** dari Google Fonts.

```
blade
      <style>
        body {
            font-family: 'Nunito', sans-serif;
        }
    </style>
```

- Mengubah font default menjadi **Nunito**.

3. **Navbar (Navigasi)**

```
blade
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container">
        <a class="navbar-brand" href="#">Tokoku</a>
        <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-
label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse"
id="navbarSupportedContent">
            <ul class="navbar-nav ml-auto">
                <li class="nav-item active">
                    <a class="nav-link" href="{{ route('home')
}}">Home <span class="sr-only">(current)</span></a>
                </li>
            </ul>
        </div>
    </div>
</nav>
```

- **Navbar responsif** menggunakan Bootstrap.
- **Link navigasi ke route('home')**, yang akan mengarahkan ke halaman utama.
- **Menggunakan ml-auto** agar menu ada di sebelah kanan.

4. **Menampilkan Pesan Flash dan Error**

Pesan Error Validasi

```
blade
@if ($errors->any())
    <div class="alert alert-danger alert-dismissible fade show"
role="alert">
```

```
        <strong>Oops! Something went wrong:</strong>
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
        <button type="button" class="close" data-
dismiss="alert" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
@endif
```

- **Menampilkan daftar error validasi** jika ada input yang salah saat submit form.

Pesan Sukses

```
blade
@if (session('success'))
    <div class="alert alert-success alert-dismissible fade
show" role="alert">
        {{ session('success') }}
        <button type="button" class="close" data-
dismiss="alert" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
@endif
```

- Jika ada session **success**, maka pesan sukses akan ditampilkan.

Pesan Error Umum

```
blade
@if (session('error'))
    <div class="alert alert-danger alert-dismissible fade show"
role="alert">
        {{ session('error') }}
        <button type="button" class="close" data-
dismiss="alert" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
@endif
```

- Jika ada session **error**, maka pesan error akan ditampilkan.

5. **Menampilkan Konten Halaman**

```blade
@yield('content')
```

- **Digunakan sebagai tempat untuk menampilkan konten dari halaman lain**.
- Halaman lain yang menggunakan layout ini harus memiliki @section('content').

6. **Menyertakan Script JavaScript**

```blade
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
```

Menggunakan jQuery dan Bootstrap JS agar navbar dan komponen Bootstrap lainnya bisa berfungsi.

Kesimpulan

Fungsi utama :

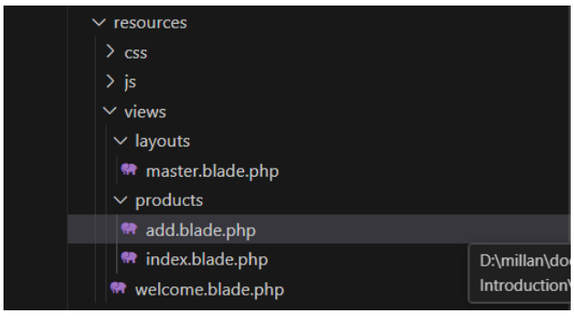
1. Menyediakan struktur HTML standar untuk halaman lain.
2. Memuat Bootstrap dan jQuery untuk tampilan yang lebih baik.
3. Menampilkan navbar dengan link ke halaman utama.
4. Menampilkan pesan flash (sukses/gagal) dan error validasi.
5. Memiliki @yield('content') agar halaman lain bisa memasukkan kontennya.

Jadi, halaman lain bisa menggunakan layout ini dengan:

```blade
@extends('layouts.master')

@section('content')
    <h1>Halo, ini halaman utama!</h1>
@endsection
```

3. **Menambahkan add.blade.php**

Selanjutnya kita akan menambahkan file add.blade.php dalam framework **Laravel**, file add.blade.php bis akita buat dengan cara yang sama sebelumnya seperti index dan master. Untuk Lokasi penyimpanannya sama dengan di index yaitu folder products untuk lokasinya sendiri berada di resources → views → products → add.blade.php.



Jika sudah dibuat maka Langkah selanjutnya dengan menyalin kode berikut ke console file add.blade.php.

```
blade
@extends('layouts.master')

@section('content')
<section class="container mt-4">
    <div class="card shadow-sm p-4">
        <h4 class="mb-4">Tambah Produk</h4>
        <form action="{{ route('product.save') }}"
method="POST">
            @csrf

            <div class="mb-3">
                <label for="name" class="form-label">Nama
Produk</label>
                <input name="name" type="text" class="form-
control" id="name" placeholder="Masukkan nama produk" value="{{
old('name') }}">
                <small class="form-text text-muted">Masukkan
nama produk yang dijual</small>
            </div>

            <div class="mb-3">
                <label for="description" class="form-
label">Dekodesi Produk</label>
                <textarea name="description" class="form-
control" id="description" placeholder="Masukkan dekodesi
produk">{{ old('description') }}</textarea>
```

```
                <small class="form-text text-muted">Masukkan
keterangan produk yang dijual</small>
            </div>

            <div class="mb-3">
                <label for="image" class="form-label">URL Gambar
Produk</label>
                <input type="url" class="form-control"
id="image" name="image" placeholder="Masukkan URL gambar produk"
value="{{ old('image') }}">
                <small class="form-text text-muted">Masukkan URL
gambar produk (contoh: https://example.com/image.jpg)</small>
            </div>

            <div class="d-flex gap-2">
                <button type="submit" class="btn btn-
primary">Simpan</button>
                <a href="{{ route('home') }}" class="btn btn-
secondary">Batal</a>
            </div>
        </form>
    </div>
</section>
@endsection
```

Kode di atas adalah **view Blade (add.blade.php) dalam Laravel**, yang digunakan untuk **menampilkan form tambah produk**. Berikut adalah **penjelasan rinci fungsi setiap bagian kode**:

1. **Menggunakan Template Master**

```
blade
@extends('layouts.master')
```

- Menggunakan **template utama (layouts/master.blade.php)** agar halaman ini memiliki **header, footer, dan styling** yang sama dengan halaman lain.
- File layouts/master.blade.php biasanya memiliki struktur HTML utama, termasuk **Bootstrap atau Tailwind untuk styling**.

2. **Mendefinisikan Bagian Konten**

```
blade
@section('content')
```

- Mengisi bagian @yield('content') yang ada di layouts.master, sehingga konten ini akan muncul di tempat yang sudah disediakan di template utama.

3. **Container untuk Form Tambah Produk**

```
blade
<section class="container mt-4">
    <div class="card shadow-sm p-4">
        <h4 class="mb-4">Tambah Produk</h4>
```

- **<section class="container mt-4">** → Membungkus form dalam container Bootstrap dengan margin atas (mt-4) agar tampilan lebih rapi.
- ** ** → Menggunakan **Bootstrap Card** dengan **shadow kecil (shadow-sm)** dan padding **4 (p-4)** untuk tampilan yang lebih menarik.

4. **Form Tambah Produk**

```
blade
<form action="{{ route('product.save') }}" method="POST">
    @csrf
```

- **action="{{ route('product.save') }}"**
  - Form akan mengirim data ke **route bernama product.save**.
  - Biasanya, route ini didefinisikan di **routes/web.php** dan mengarah ke method **store() di ProductController**.

```
php
Route::post('/product/save', [ProductController::class, 'store'])->name('product.save');
```

  - **method="POST"**
    - Menggunakan metode **POST** untuk mengirim data **ke server** (bukan GET, karena GET tidak aman untuk data sensitif).
  - **@csrf**
    - Laravel menggunakan **CSRF Token** untuk **keamanan** agar permintaan POST ini hanya bisa dilakukan dari halaman yang sah.

5. **Input Nama Produk**

```
blade
<div class="mb-3">
    <label for="name" class="form-label">Nama Produk</label>
    <input name="name" type="text" class="form-control" id="name" placeholder="Masukkan nama produk" value="{{ old('name') }}">
    <small class="form-text text-muted">Masukkan nama produk yang dijual</small>
</div>
```

- Membuat **input teks** untuk **nama produk**.
- **name="name"** → Sesuai dengan field di database yang akan menyimpan nama produk.

- **id="name"** → Memudahkan penggunaan **JavaScript atau label <label for="name">**.
- **placeholder="Masukkan nama produk"** → Menampilkan teks petunjuk di input.
- **value="{{ old('name') }}"**
  - Jika form dikirim tetapi ada error, **Laravel akan mengisi kembali input dengan nilai sebelumnya** agar pengguna tidak perlu mengetik ulang.

6. **Input Deskripsi Produk**

```
blade
<div class="mb-3">
    <label for="description" class="form-label">Deskripsi Produk</label>
    <textarea name="description" class="form-control" id="description" placeholder="Masukkan deskripsi produk">{{ old('description') }}</textarea>
    <small class="form-text text-muted">Masukkan keterangan produk yang dijual</small>
</div>
```

- **Membuat input teks area** untuk deskripsi produk.
- **name="description"** → Digunakan untuk menyimpan deskripsi di database.
- **placeholder="Masukkan deskripsi produk"** → Petunjuk input.
- **value="{{ old('description') }}"** → Mengisi kembali input jika ada error.

7. **Input URL Gambar Produk**

```
blade
<div class="mb-3">
    <label for="image" class="form-label">URL Gambar Produk</label>
    <input type="url" class="form-control" id="image" name="image" placeholder="Masukkan URL gambar produk" value="{{ old('image') }}">
    <small class="form-text text-muted">Masukkan URL gambar produk (contoh: https://example.com/image.jpg)</small>
</div>
```

- Input khusus untuk **URL gambar**.
- **type="url"** → Memastikan input hanya menerima format URL.
- **name="image"** → Field database yang menyimpan URL gambar.
- **value="{{ old('image') }}"** → Agar tetap terisi jika form gagal dikirim.

8. **Tombol Simpan & Batal**

```
blade
<div class="d-flex gap-2">
```

```
    <button type="submit" class="btn btn-
primary">Simpan</button>
    <a href="{{ route('home') }}" class="btn btn-
secondary">Batal</a>
</div>
```

- **Tombol "Simpan"**
  - **type="submit"** → Mengirimkan form ke Laravel untuk disimpan.
  - **class="btn btn-primary"** → Menggunakan **tombol Bootstrap** warna biru.
- **Tombol "Batal"**
  - **Menggunakan <a> dengan href="{{ route('home') }}"** untuk kembali ke halaman utama.
  - **class="btn btn-secondary"** → Menggunakan warna abu-abu untuk tombol batal.

9. **Menutup @section**

```
blade
@endsection
```

- Menandakan **akhir dari bagian content**, agar bisa digunakan dalam template layouts.master.

Kesimpulan

Kode ini adalah form tambah produk dalam Laravel, yang memiliki fitur:

- Menggunakan Blade Template Engine
- Menggunakan Bootstrap untuk tampilan modern
- Menggunakan CSRF protection untuk keamanan
- Menampilkan input untuk nama, deskripsi, dan gambar produk
- Menggunakan validasi old() untuk mengisi kembali input jika terjadi error
- Memiliki tombol "Simpan" dan "Batal"

4. **Menambahkan edit.blade.php**

File blade terakhir yang akan ditambahkan dalam framework laravel yaitu edit.blade.php, Lokasi penyimpanannya di folder products. Setelah file berhasil dibuat salin kode berikut dibawah ini ke console file edit.blade.php.

```
blade
@extends('layouts.master')

@section('content')
<section>
    <div class="container">
        <h4>Edit Produk</h4>
        <form action="{{ route('product.update', $product->id)
}}" method="POST" accept-charset="utf-8">
```

```
            @csrf
            @method('PUT')

            <div class="mb-3">
                <label for="title" class="form-label">Nama
Produk</label>
                <input name="name" type="text" class="form-
control" id="title"
                    aria-describedby="fet1"
placeholder="Masukkan nama produk"
                    value="{{ old('name', $product->name) }}">
                <small id="fet1" class="form-text text-
muted">Masukkan nama produk yang dijual</small>
            </div>

            <div class="mb-3">
                <label for="desc" class="form-label">Deskripsi
Produk</label>
                <input name="description" type="text"
class="form-control" id="desc"
                    aria-describedby="fet2"
placeholder="Masukkan deskripsi produk"
                    value="{{ old('description', $product-
>description) }}">
                <small id="fet2" class="form-text text-
muted">Masukkan keterangan produk yang dijual</small>
            </div>

            <div class="mb-3">
                <label for="image" class="form-label">Gambar
Produk</label>
                <div class="mb-2">
                    <img src="{{ $product->image }}" alt="Gambar
Produk" class="img-fluid" width="150">
                </div>
                <input name="image" type="text" class="form-
control" id="image"
                    placeholder="Masukkan URL gambar produk"
                    value="{{ old('image', $product->image) }}">
                <small class="form-text text-muted">Masukkan URL
gambar produk (contoh: https://example.com/image.jpg)</small>
            </div>
```

```
            <div class="d-flex gap-2">
                <button type="submit" class="btn btn-
primary">Simpan</button>
                <a href="{{ route('home') }}" class="btn btn-
secondary">Batal</a>
            </div>
        </form>
    </div>
</section>
@endsection
```

Kode di atas adalah **view Blade dalam Laravel** yang digunakan untuk **menampilkan form edit produk**. Form ini memungkinkan pengguna untuk **memperbarui data produk yang sudah ada**.

1. **Menggunakan Template Master**

```blade
@extends('layouts.master')
```

- **Menggunakan template utama (layouts.master)** untuk menyusun halaman agar lebih **rapi dan konsisten**.
- File layouts/master.blade.php biasanya berisi **header, footer, dan styling utama**.

2. **Mendefinisikan Bagian Konten**

```blade
@extends('layouts.master')
```

- **Menandakan awal bagian konten utama** yang akan menggantikan @yield('content') dalam template layouts.master.

3. **Container & Judul Halaman**

```blade
<section>
    <div class="container">
        <h4>Edit Produk</h4>
```

- **Membuat section** yang berisi form edit produk.
- **Menggunakan div.container** agar tampilan lebih rapi dengan Bootstrap.
- **Judul halaman "Edit Produk"** untuk memberi tahu pengguna bahwa mereka sedang mengedit produk.

4. **Form Edit Produk**

```blade
<form action="{{ route('product.update', $product->id) }}"
method="POST" accept-charset="utf-8">
    @csrf
    @method('PUT')
```

1. **action="{{ route('product.update', $product->id) }}"**
   - Form akan mengirim data ke **route product.update** dengan **ID produk yang diedit**.
   - Biasanya, route ini didefinisikan di routes/web.php seperti ini:

```php
Route::put('/product/update/{id}',
[ProductController::class, 'update'])-
>name('product.update');
```

2. **method="POST"**
   - Laravel hanya mendukung metode GET dan POST, jadi kita tetap menggunakan **method="POST"** dan menambahkan **@method('PUT')**.
3. **@csrf**
   - Token **CSRF protection** untuk keamanan Laravel.
4. **@method('PUT')**
   - Laravel tidak mendukung PUT langsung dalam <form>, jadi kita harus menambah **@method('PUT')** agar request diakui sebagai metode **PUT**.

5. **Input Nama Produk**

```blade
<div class="mb-3">
    <label for="title" class="form-label">Nama Produk</label>
    <input name="name" type="text" class="form-control"
id="title"
        aria-describedby="fet1" placeholder="Masukkan nama
produk"
        value="{{ old('name', $product->name) }}">
    <small id="fet1" class="form-text text-muted">Masukkan nama
produk yang dijual</small>
</div>
```

- **Membuat input teks untuk nama produk**.
- **name="name"** → Sesuai dengan field di database.
- **value="{{ old('name', $product->name) }}"**
  - Jika terjadi kesalahan validasi, Laravel akan mengisi kembali input dengan **data yang sebelumnya diketik (old('name'))**.
  - Jika tidak ada error, input diisi dengan **data dari database ($product->name)**.

- **aria-describedby="fet1"**
  - Untuk aksesibilitas, menghubungkan input dengan teks bantuan di <small>.

6. **Input Deskripsi Produk**

```blade
<div class="mb-3">
    <label for="desc" class="form-label">Deskripsi
Produk</label>
    <input name="description" type="text" class="form-control"
id="desc"
        aria-describedby="fet2" placeholder="Masukkan deskripsi
produk"
        value="{{ old('description', $product->description)
}}">
    <small id="fet2" class="form-text text-muted">Masukkan
keterangan produk yang dijual</small>
</div>
```

- **Membuat input teks untuk deskripsi produk**.
- **value="{{ old('description', $product->description) }}"**
  - Jika terjadi error, Laravel akan mengisi input dengan **data yang sebelumnya diketik**.
  - Jika tidak ada error, input diisi dengan **data dari database**.

7. **Input Gambar Produk**

```blade
<div class="mb-3">
    <label for="image" class="form-label">Gambar Produk</label>
    <div class="mb-2">
        <img src="{{ $product->image }}" alt="Gambar Produk"
class="img-fluid" width="150">
    </div>
    <input name="image" type="text" class="form-control"
id="image"
        placeholder="Masukkan URL gambar produk"
        value="{{ old('image', $product->image) }}">
    <small class="form-text text-muted">Masukkan URL gambar
produk (contoh: https://example.com/image.jpg)</small>
</div>
```

- **Menampilkan gambar produk yang tersimpan di database**.
- **Input teks untuk URL gambar produk**:
  - Jika terjadi error, Laravel akan mengisi kembali input dengan **data yang sebelumnya diketik (old('image'))**.

- Jika tidak ada error, input diisi dengan **data dari database ($product->image)**.
- **Teks bantuan** memberikan contoh URL gambar yang valid.

8. **Tombol Simpan & Batal**

```blade
<div class="d-flex gap-2">
    <button type="submit" class="btn btn-primary">Simpan</button>
    <a href="{{ route('home') }}" class="btn btn-secondary">Batal</a>
</div>
```

- **Tombol "Simpan"**
- Menggunakan **type="submit"** untuk mengirim form.
- **Bootstrap class btn btn-primary** untuk tampilan biru.
- **Tombol "Batal"**
- **Menggunakan <a> dengan href="{{ route('home') }}"** untuk kembali ke halaman utama tanpa menyimpan perubahan.
- **Bootstrap class btn btn-secondary** untuk tampilan abu-abu.

9. **Menutup @section**

```blade
@endsection
```

- **Menandakan akhir dari bagian content**, agar bisa digunakan dalam template layouts.master.

Kesimpulan

Kode ini adalah form edit produk dalam Laravel, yang memiliki fitur:

- Menggunakan Blade Template Engine
- Memperbarui data produk dengan metode PUT
- Menampilkan data produk yang sudah ada
- Menampilkan gambar produk yang tersimpan
- Memiliki CSRF protection untuk keamanan
- Validasi dengan old() untuk mengisi kembali input jika terjadi error
- Menggunakan Bootstrap untuk tampilan modern

Langkah 6 : Konfigurasi File web.php

Langkah ini adalah Langkah yang ada pada konfigurasi basis data, pada Langkah ini kita cuma mengkonfigurasi ulang file web.php dengan mengupdate kode program untuk memastikan aplikasi berjalan dengan semestinya. Kita bisa langsung memasukkan kode programnya kedalam console web.php.

```php
<?php

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

Route::get('/', [ProductController::class, 'index'])->name('home');

Route::prefix('product')->group(function () {
    Route::get('/add', [ProductController::class, 'add'])->name('product.add'); // Halaman tambah produk
    Route::get('/{id}/edit', [ProductController::class, 'edit'])->name('product.edit'); // Halaman edit produk
    Route::post('/save', [ProductController::class, 'save'])->name('product.save'); // Simpan produk baru
    Route::put('/update/{id}', [ProductController::class, 'update'])->name('product.update'); // Update produk (perbaikan di sini)
    Route::delete('/delete/{id}', [ProductController::class, 'delete'])->name('product.delete'); // Hapus produk
    Route::get('/list', [ProductController::class, 'list'])->name('product.list'); // Halaman daftar produk
});

// Tambahkan route untuk products.index
Route::get('/products', [ProductController::class, 'index'])->name('products.index');
```

Kode di atas adalah konfigurasi **routing** dalam **Laravel** yang digunakan untuk menangani permintaan HTTP terkait produk dalam aplikasi web. Berikut adalah penjelasan rinci dari setiap bagian kode:

1. **Mengimpor Namespace yang Diperlukan**

```php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ProductController;
```

- **use Illuminate\Support\Facades\Route;** → Memanggil Route dari Laravel untuk mendefinisikan rute aplikasi.
- **use App\Http\Controllers\ProductController;** → Mengimpor ProductController, yaitu controller yang akan menangani logika bisnis untuk produk.

2. **Route Beranda (Home)**

```
php
Route::get('/', [ProductController::class, 'index'])-
>name('home');
```

- **Route::get('/', ...)** → Menangani permintaan **GET** ke URL **"/"** (halaman utama).
- **[ProductController::class, 'index']** → Memanggil metode index() dari ProductController.
- **->name('home')** → Memberikan nama **"home"** pada route ini untuk mempermudah pemanggilan dalam kode (misalnya di route('home') dalam Blade template).

3. **Route Kelompok (Prefix product)**

```
php
Route::prefix('product')->group(function () {
```

- **Route::prefix('product')->group(function () { ... })** → Mengelompokkan semua rute yang memiliki prefix **"product"** agar lebih rapi dan mudah dikelola.
- Semua rute di dalamnya akan otomatis memiliki prefix /product/... dalam URL.

4. **Route Tambah Produk**

```
php
Route::get('/add', [ProductController::class, 'add'])-
>name('product.add');
```

- **GET /product/add** → Menampilkan halaman form tambah produk.
- **Method add() dalam ProductController** akan dipanggil untuk merender halaman tersebut.

5. **Route Edit Produk**

```
php
Route::get('/{id}/edit', [ProductController::class, 'edit'])-
>name('product.edit');
```

- **GET /product/{id}/edit** → Menampilkan halaman form edit produk berdasarkan id.
- **Method edit($id) dalam ProductController** akan menampilkan data produk yang akan diedit.

6. **Route Simpan Produk Baru**

```
php
Route::post('/save', [ProductController::class, 'save'])-
>name('product.save');
```

- **POST /product/save** → Mengirimkan data produk baru ke server.
- **Method save(Request $request) dalam ProductController** akan menyimpan data ke database.

7. **Route Update Produk**

```
php
Route::put('/update/{id}', [ProductController::class, 'update'])->name('product.update');
```

- **PUT /product/update/{id}** → Mengupdate produk berdasarkan id.
- **Method update(Request $request, $id) dalam ProductController** menangani logika pembaruan produk.
- Menggunakan metode PUT karena konvensi REST API untuk update data.

8. **Route Hapus Produk**

```
php
Route::delete('/delete/{id}', [ProductController::class, 'delete'])->name('product.delete');
```

- **DELETE /product/delete/{id}** → Menghapus produk berdasarkan id.
- **Method delete($id) dalam ProductController** akan menangani penghapusan produk dari database.

9. **Route Daftar Produk**

```
php
Route::get('/list', [ProductController::class, 'list'])->name('product.list');
```

- **GET /product/list** → Menampilkan daftar semua produk.
- **Method list() dalam ProductController** akan mengembalikan tampilan daftar produk.

10. **Route untuk Halaman Produk (Index)**

```
php
Route::get('/products', [ProductController::class, 'index'])->name('products.index');
```

- **GET /products** → Menampilkan halaman utama daftar produk.
- **Method index() dalam ProductController** akan digunakan untuk merender daftar produk.

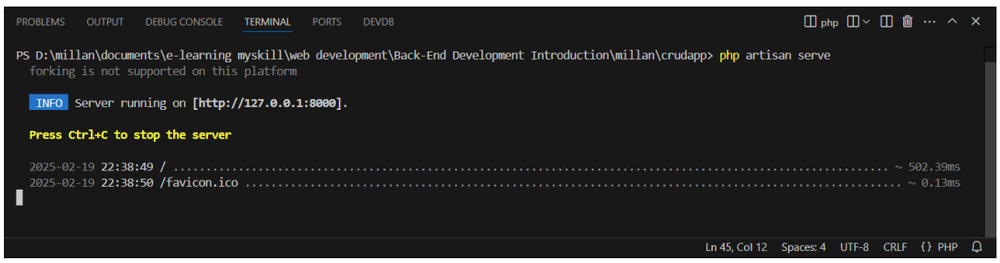
Kesimpulan

Kode ini mengatur semua **route** terkait produk, termasuk:

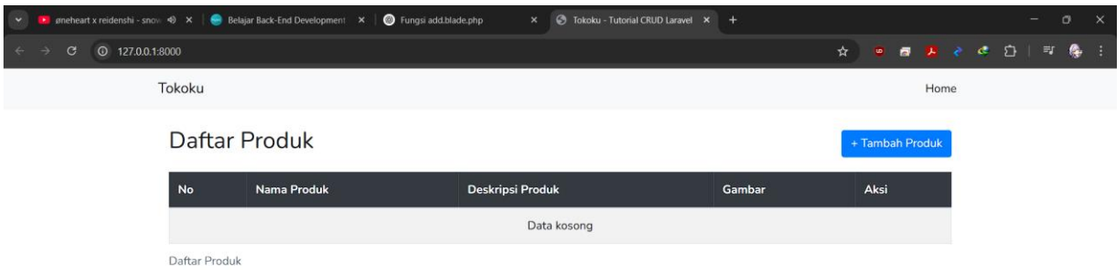
- Menampilkan produk
- Menambah produk
- Mengedit produk
- Menyimpan produk baru
- Memperbarui produk
- Menghapus produk

Langkah 7 : Jalankan Aplikasi

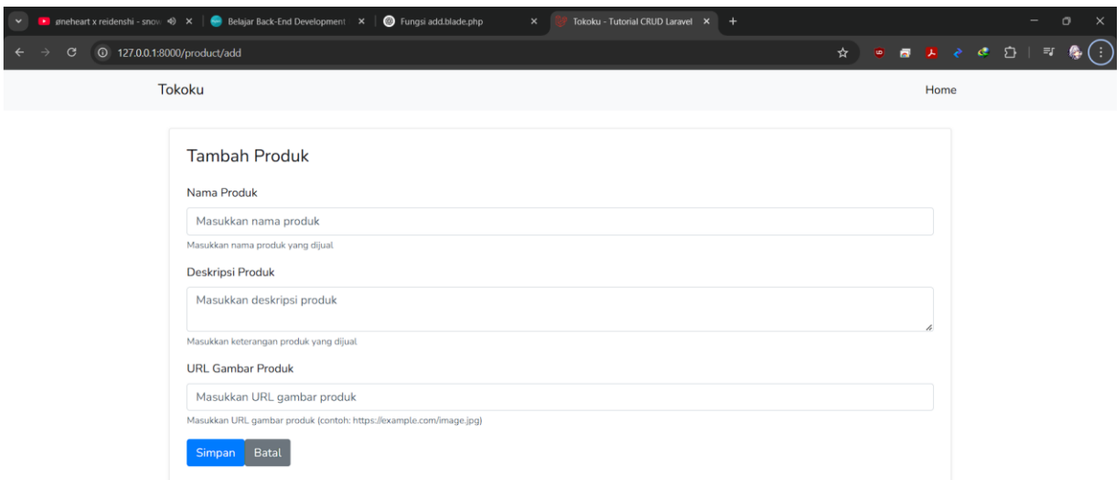
Selanjutnya Langkah yang terakhir adalah kita akan menjalankan program aplikasi CRUD Laravel yang sudah kita buat dan susun, kita akan mencoba membuka aplikasinya, menginput data, mengubah data, dan menghapus data. Kita mulai saja untuk menjalankan program tersebut dengan cara memasukan perintah berikut keterminal (php artisan serve) setelah berhasil akan tampil seperti ini.



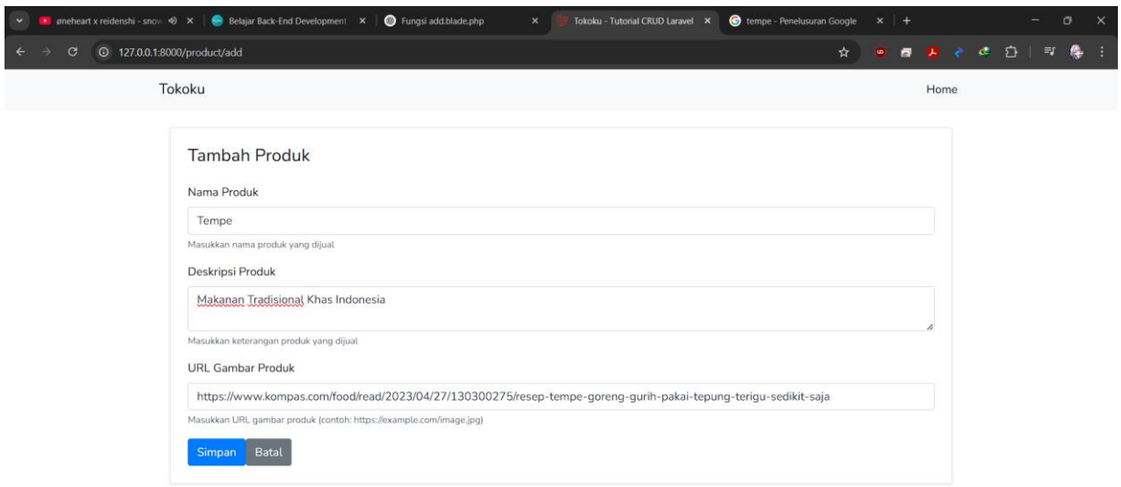
Setelah aplikasi berjalan maka akan terbuka jendala baru browser dengan menampilkan halaman utama aplikasi.



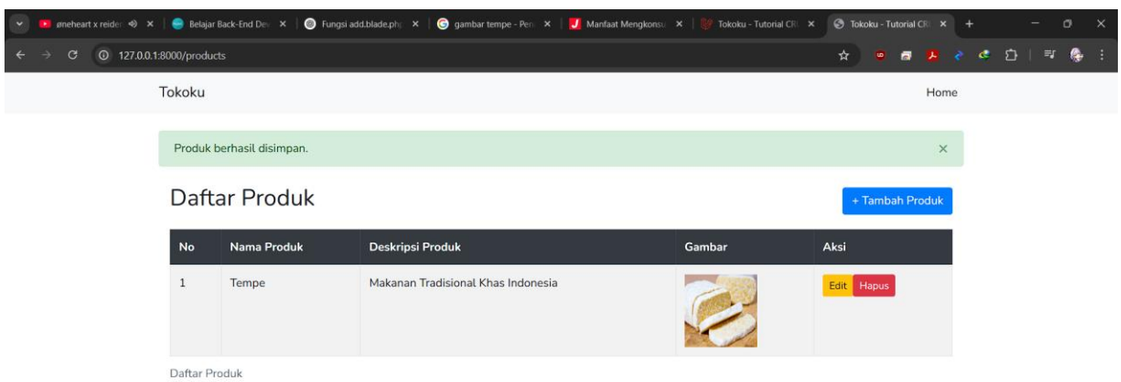
Seperti ini tampilan aplikasi CRUD yang kita buat, terlihat masih kosong dan belum ada data. Selanjutnya kita akan mencoba menginput data kedalam aplikasi dengan cara mengklik tombol biru bertuliskan “+Tambah Produk” pada halaman awal aplikasi.



Jika sudah maka hasil tampilannya akan seperti ini. Didalam tampilan tambah produk kitab isa lihat disana ada tiga kolom berupa Nama Produk, Deskripsi Produk, dan URL Gambar Produk. Kita akan memasukkan data apa saja kedalam aplikasi sebagai contoh kita akan memasukan Nama Produk seperti makanan “Tempe” lalu kita masukan Deskripsi Produknya dengan “Makanan Tradisional Khas Indonesia” selanjutnya kita salin URL gambar Tempe digoogle tadi kekolom URL Gambar Produk.

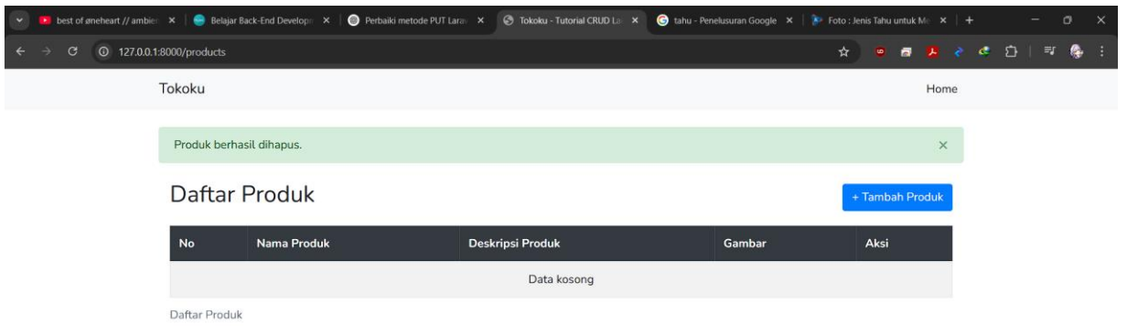


Jika sudah, kita klik langsung tombol simpan, jika berhasil tampilannya akan seperti dibawah ini.



Selanjutnya kita coba untuk mengubah data tersebut yang awalnya tempe menjadi tahu, kita klik tombol edit berwarna kuning tersebut lalu kita ubah mulai dari nama, deskripsi dan URL gambarnya menjadi tahu.

Ketika sudah berhasil mengubah data tersebut maka Langkah selanjutnya kita akan menghapusnya dengan cara klik tombol hapus berwarna merah, jika berhasil tampilan halaman aplikasi akan menjadi kosong seperti dibawah ini.



Dengan begitu kita sudah berhasil membuat program aplikasi berbasis CRUD menggunakan Larvel.

Kesimpulan

CRUDApp Laravel adalah aplikasi Laravel yang mengelola data dengan operasi dasar CRUD. Dengan Laravel, pembuatan aplikasi CRUD menjadi lebih mudah karena adanya fitur seperti **Eloquent ORM**, **Blade Templating**, dan **Routing Resourceful**.

Setelah pengenalan apa itu CRUD kita akan mulau melakukan Langkah-langkah pembuatannya dan tools apa yang akan kita butuhkan sebelum memulai.