



MODUL PRAKTIKUM WEB

LARAVEL (Instalasi, MVC)

TEKNIK INFORMATIKA
FAKULTAS TEKNIK UNIVERSITAS PASUNDAN
2025/2026

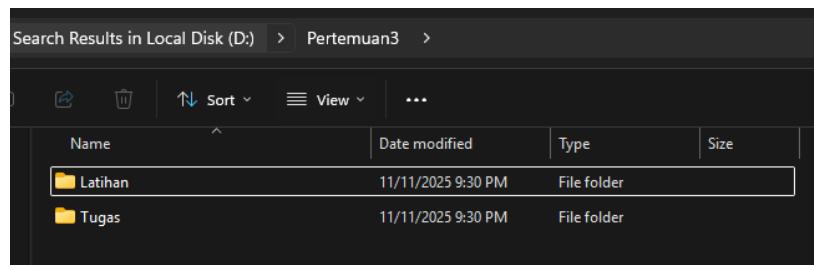
DAFTAR ISI

A. Instalasi Laravel 12	1
B. Struktur Folder Laravel 12	11
1. Folder app/	11
2. Folder resources/	12
3. Folder routes/	12
4. Folder public/	13
5. Folder database/	14
6. File .env	14
C. View & Blade Templating Laravel 12	15
1. View	15
2. Blade Templating	15
3. Konsep Blade	15
4. Slot	17
5. Named Slot	18
D. Migration	20
1. Pengertian Migration	20
2. Menghubungkan Database (SQLite)	20
3. Struktur Migration default di Laravel	23
4. Membuat Migration Baru untuk Category dan Post	23
E. Model	26
1. Penjelasan: Apa itu Model?	26
2. Apa itu ORM (Object Relational Mapping)?	27
3. Membuat Model	27
F. Seeder & Factory	30
1. Penjelasan Seeder & Factory	30
2. Membuat Factory	30
3. Isi File Factory	30
4. Membuat Seeder	31
G. Data dan Relasi Laravel 12	33

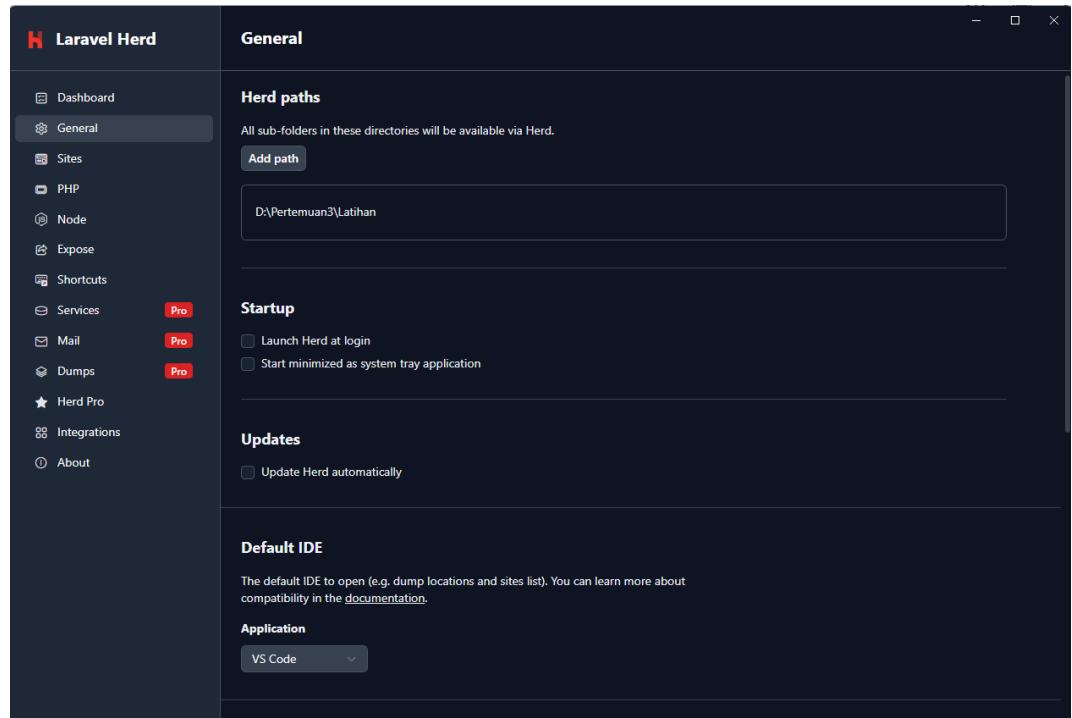
1. Relasi One to One	33
2. Relasi One to Many	33
TUGAS PRAKTEK	36
REFERENSI	37

A. Instalasi Laravel 12

- Buat struktur folder yang sama dengan pertemuan sebelumnya

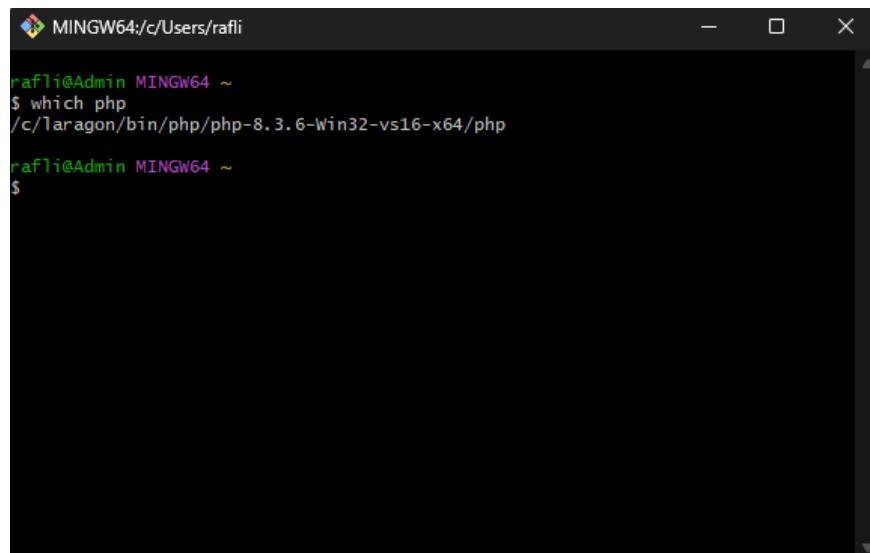


- Download Laravel Herd <https://herd.laravel.com/windows>, lalu buka general dan ubah pathnya ke folder pertemuan



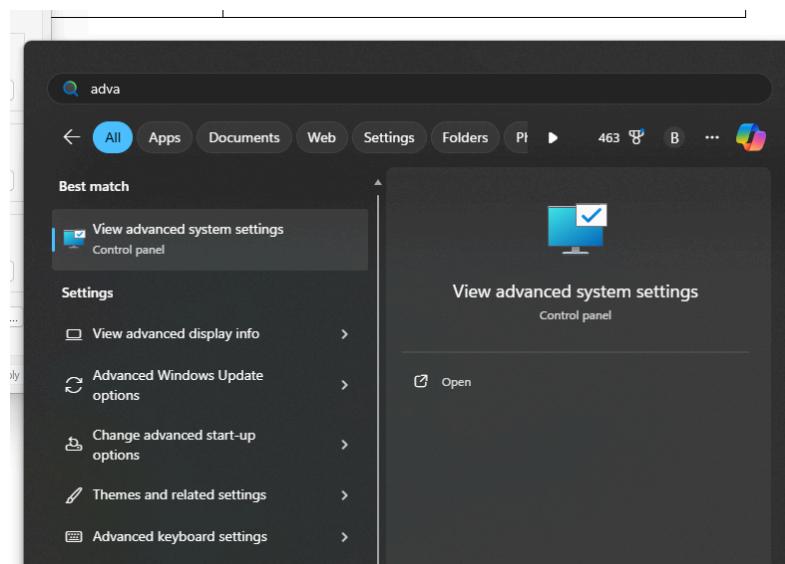
- Untuk merapihkan kode dan mempermudah dalam praktek Laravel, silahkan download ekstensi berikut ini di VSCode
 - Prettier
 - Auto Rename Tag
 - PHP Intelephense
 - PHP Namespace Resolver
 - Tailwind CSS Intellisense

- f. AlphineJS Intellisense
 - g. AlphineJS Syntax Highlight
 - h. Laravel
 - i. Laravel Blade Formatter
 - j. Laravel Blade Snippets
 - k. Laravel Blade Spacer
4. Pastikan PHP memiliki versi terbaru (PHP 8.3 keatas), caranya silahkan buka git bash, lalu tuliskan “**which php**”

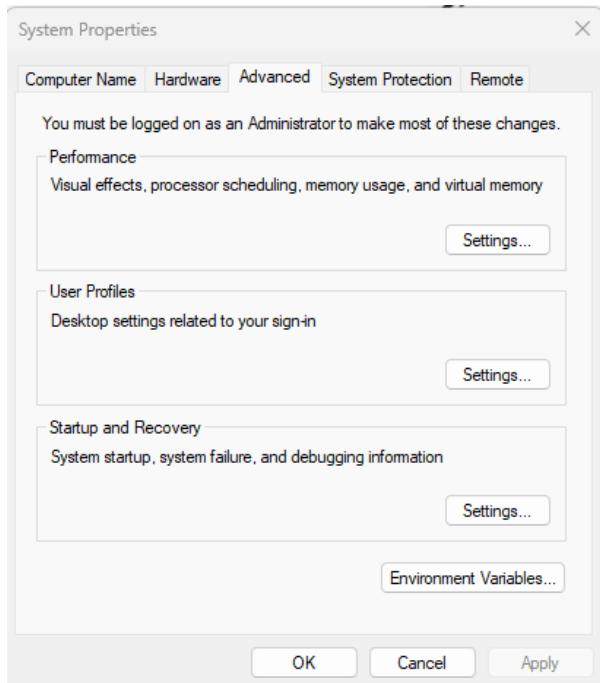


```
MINGW64:/c/Users/raffi
raffi@Admin MINGW64 ~
$ which php
/c/laragon/bin/php/php-8.3.6-Win32-vs16-x64/php
raffi@Admin MINGW64 ~
$
```

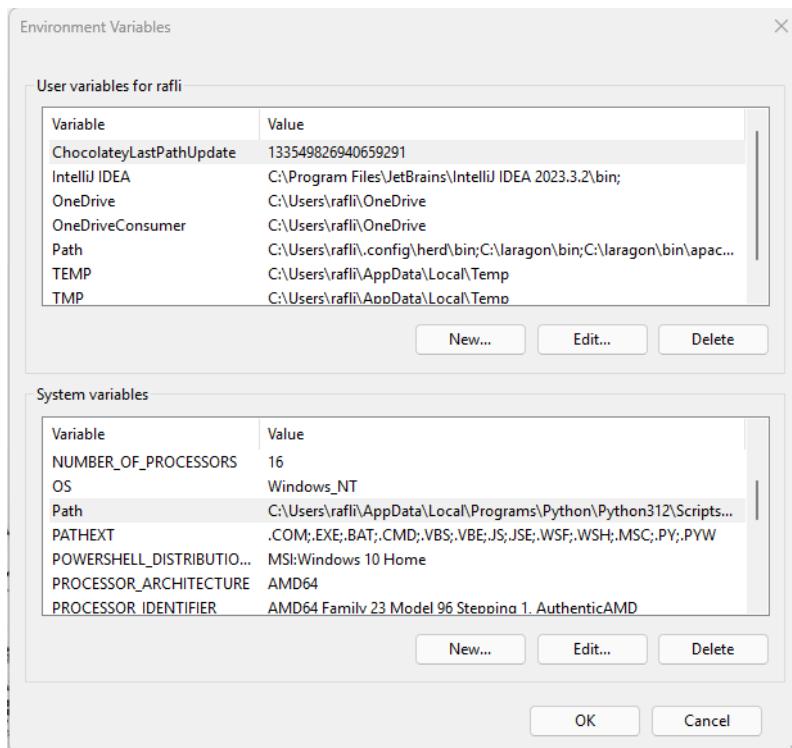
5. Jika belum, silahkan cari “Advance System Settings”, lalu buka



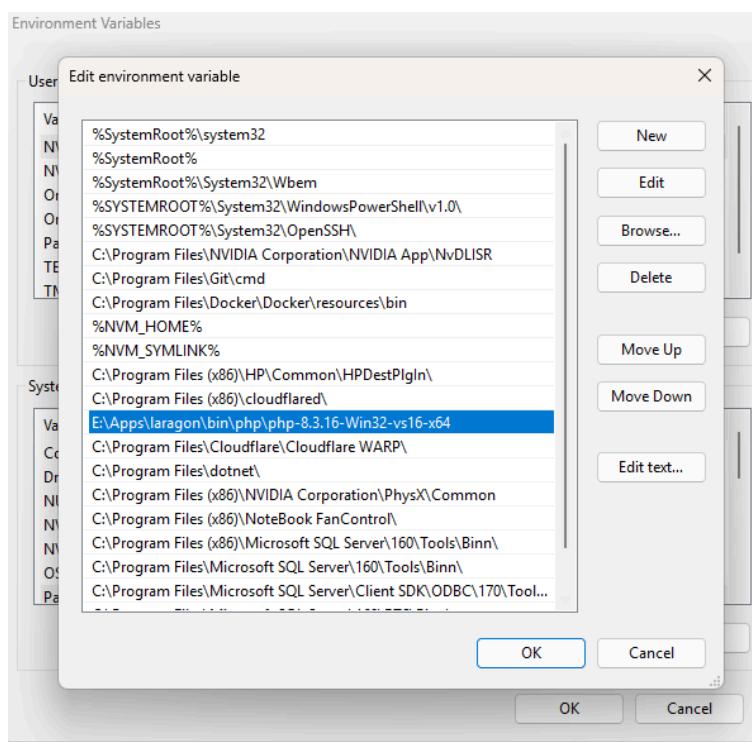
6. Lalu buka **Environment Variables** di kanan bawah



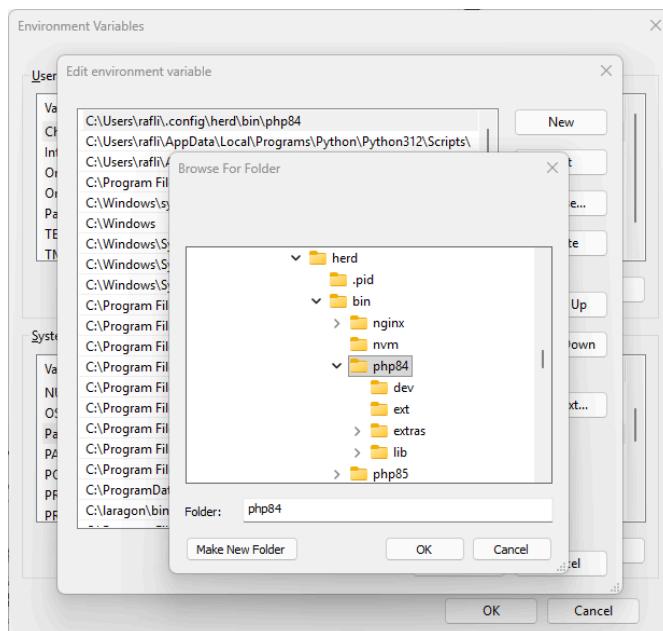
7. Pilih di "System Variables" cari "Path" lalu klik path tersebut kemudian klik edit



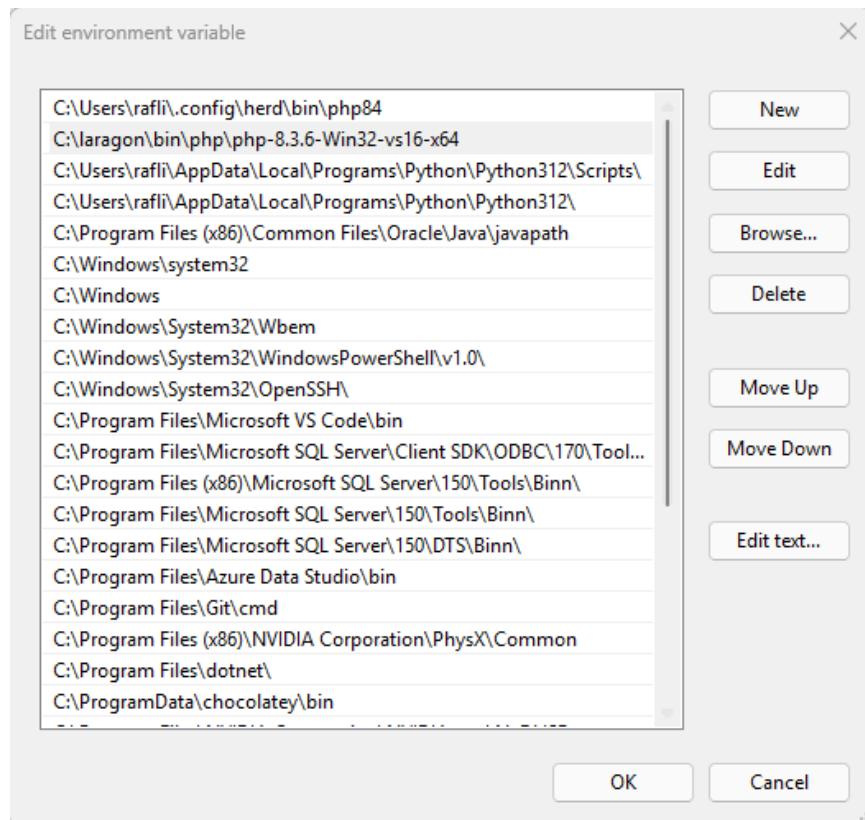
- Cari Path PHP lama kalian, jika ada, klik Path tersebut lalu hapus,



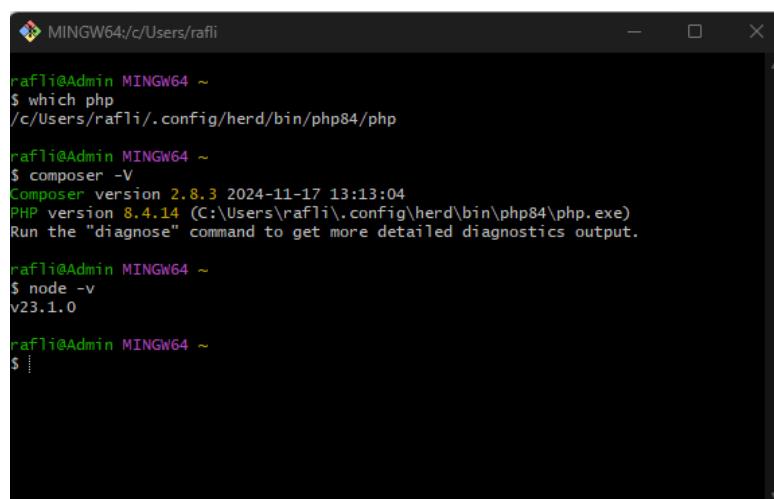
- Jika sudah di hapus, silahkan klik browse, lalu arahkan ke folder PHP 8.4 herd, jika anda install secara default, buka folder local disk C > Users > "User Computer" > .config > herd > bin > php84. Lalu klik "Ok"



10. Pastikan herd itu berada di atas laragon (jika ada) seperti pada gambar. Caranya cukup klik path yang tadi kita sudah tambahkan, lalu klik “Move Up”



11. Jika sudah, klik “Ok”, lalu cek lagi di git bash dengan menggunakan perintah **“which php”**. Cek juga, apabila versi nodejs dibawah 22 ataupun tidak mempunyai nodejs dan composer. Silahkan ikuti langkah ke 12



```
raflfi@Admin MINGW64 ~
$ which php
/c/Users/rafli/.config/herd/bin/php84/php

raflfi@Admin MINGW64 ~
$ composer -V
Composer version 2.8.3 2024-11-17 13:13:04
PHP version 8.4.14 (C:/Users/rafli/.config/herd/bin/php84/php.exe)
Run the "diagnose" command to get more detailed diagnostics output.

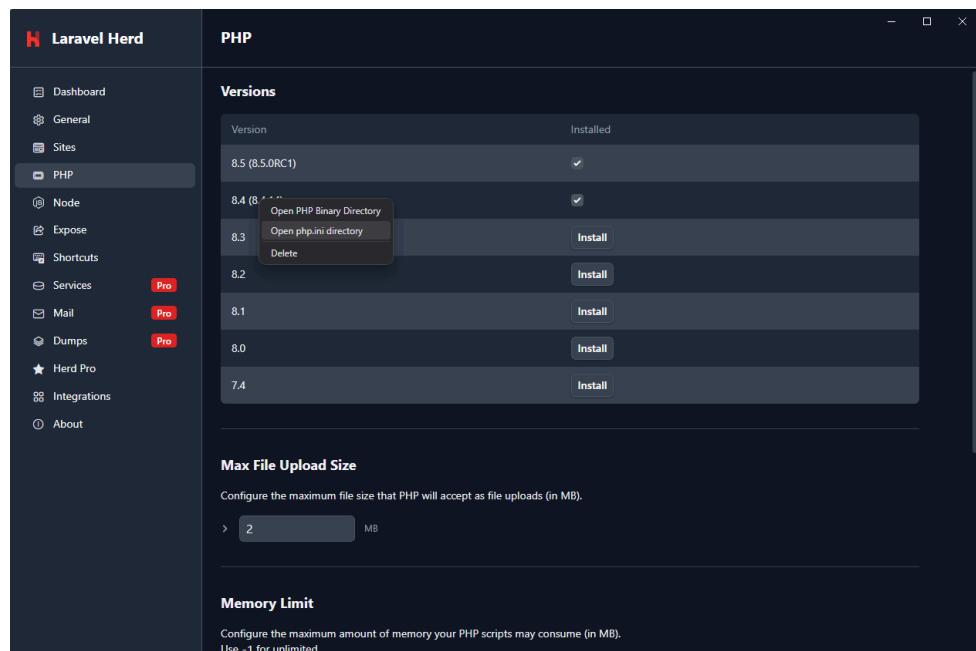
raflfi@Admin MINGW64 ~
$ node -v
v23.1.0

raflfi@Admin MINGW64 ~
$
```

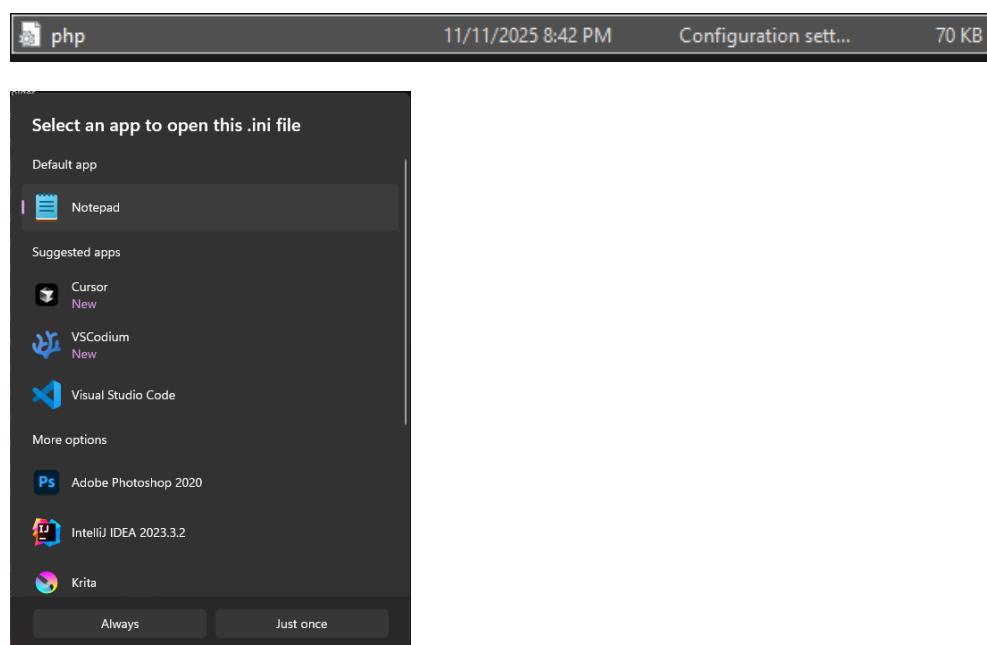
12. Silahkan download aplikasi pendukung (Jika belum terinstall)

- NodeJS 24: <https://nodejs.org/id/download>
- Composer Terbaru: <https://getcomposer.org/download/>

13. Setelah selesai, silahkan buka kembali laravel herd nya, lalu buka “PHP”, pilih PHP versi 8.4, klik kanan lalu klik “Open php.ini directory”



14. Cari file php.ini lalu pilih menggunakan notepad/vscode, lalu klik “Just Once”



15. Cari “**variables_order**” sesuai tampilan yang sama dengan gambar dibawah ini. Jika dia di awalan terdapat titik koma (;) maka hapus terlebih dahulu, lalu pastikan value “**variables_order**” adalah “EGPCS”

```

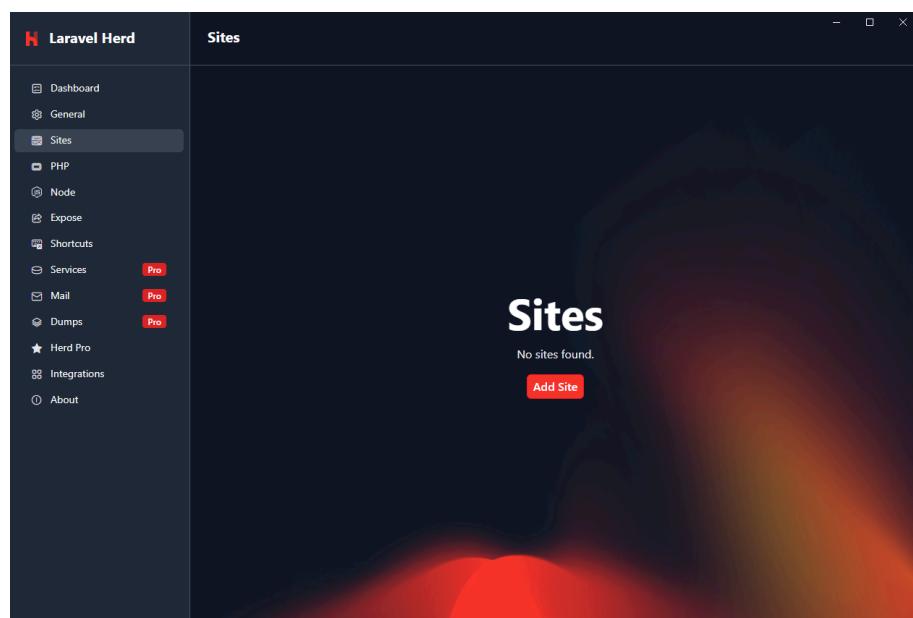
File Edit View
; Example:
;arg_separator.input = "&" ^ variables_order
;
; This directive determines which super global arrays are registered when PHP
; starts up. G,P,C,E & S are abbreviations for the following respective super
; globals: GET, POST, COOKIE, ENV and SERVER. There is a performance penalty
; paid for the registration of these arrays and because ENV is not as commonly
; used as the others, ENV is not recommended on production servers. You
; can still get access to the environment variables through getenv() should you
; need to.
; Default Value: "EGPCS"
; Development Value: "GPCS"
; Production Value: "GPCS";
; https://php.net/variables-order
variables_order = "EGPCS"

;
; This directive determines which super global data (G,P & C) should be
; registered into the super global array REQUEST. If so, it also determines
; the order in which that data is registered. The values for this directive
; are specified in the same manner as the variables_order directive,
; EXCEPT one. Leaving this value empty will cause PHP to use the value set
; in the variables_order directive. It does not mean it will leave the super
; globals array REQUEST empty.
; Default Value: None

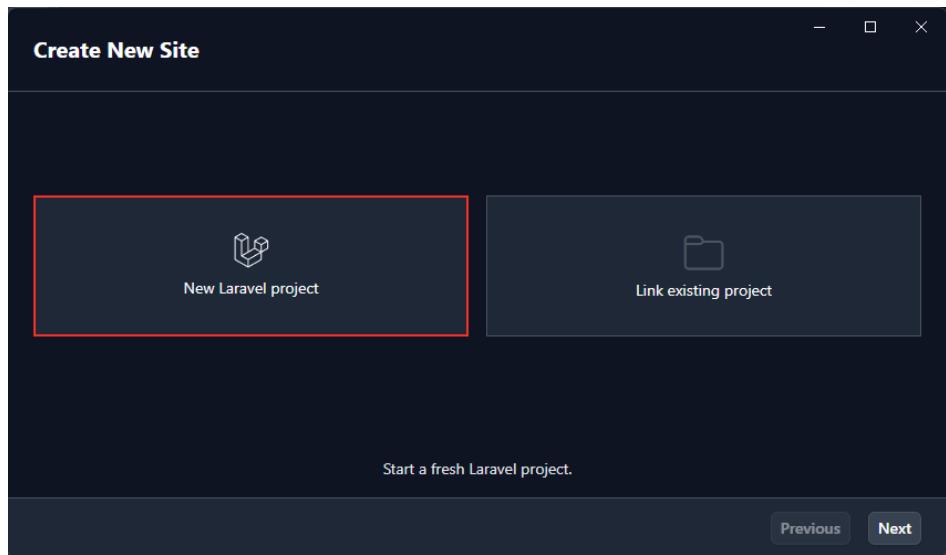
```

Ln 646, Col 16 - 15 of 69,444 characters | Plain text | 240% | Windows (CR/LF) | UTF-8

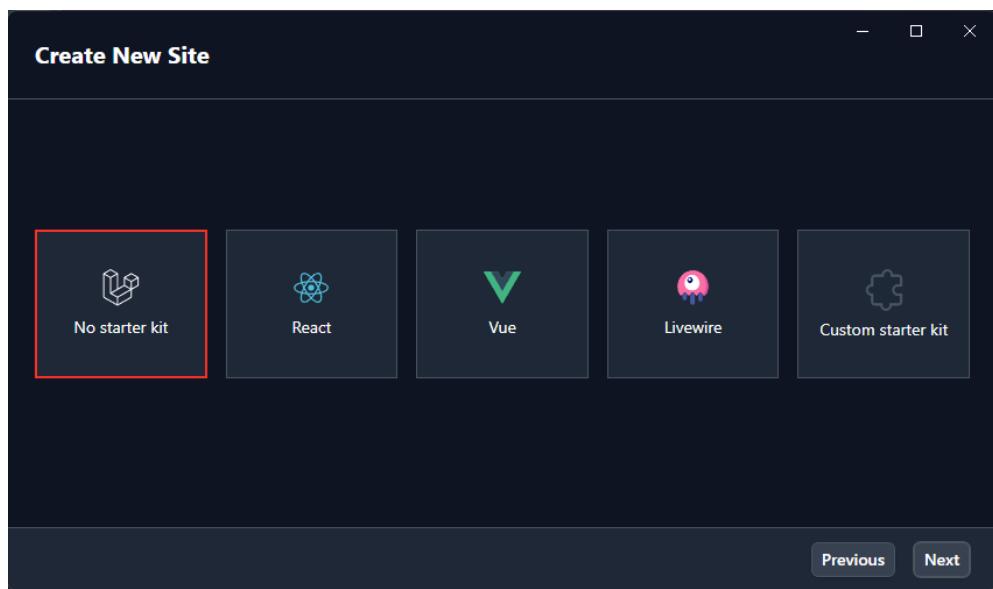
16. Silahkan restart server HERDnya, lalu silahkan buka tab “site”, lalu klik “add site”



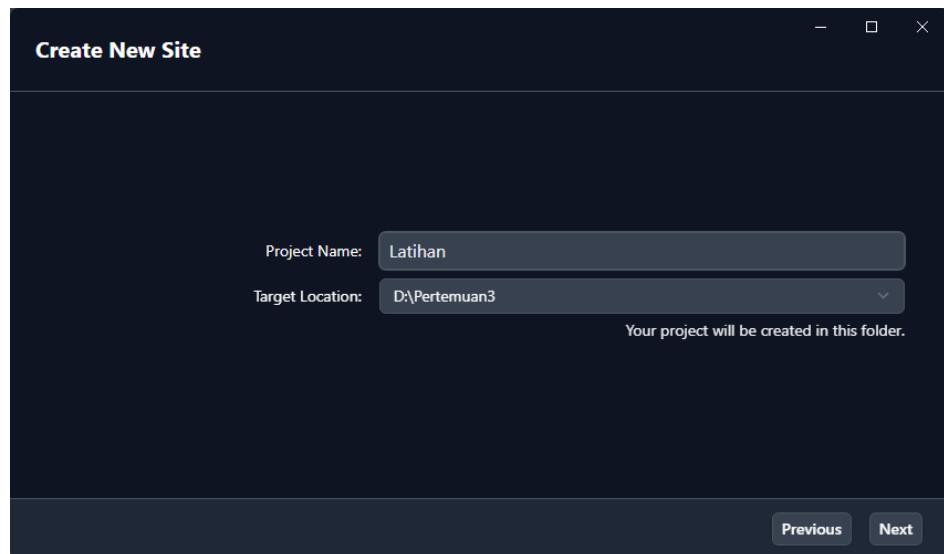
17. Pilih new laravel project



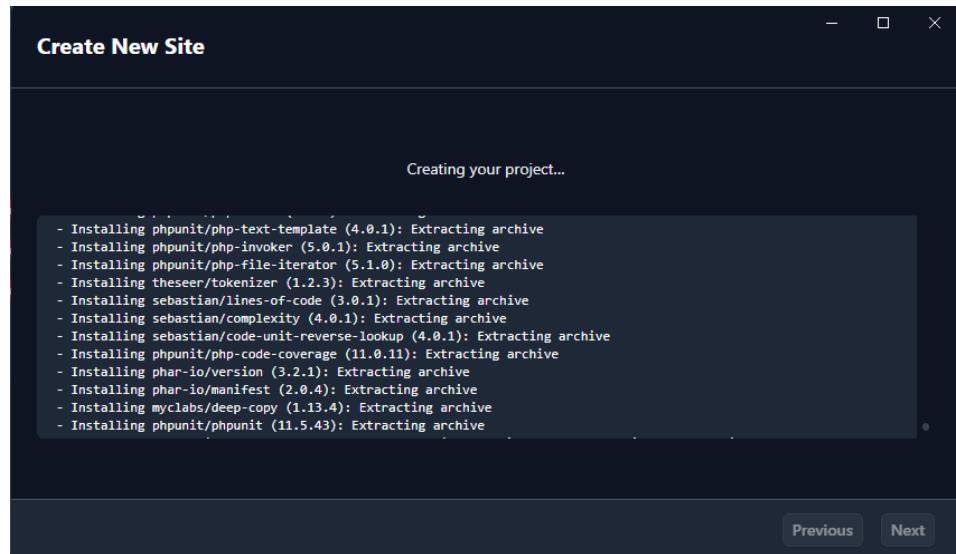
18. Pilih “No Stater Kit”



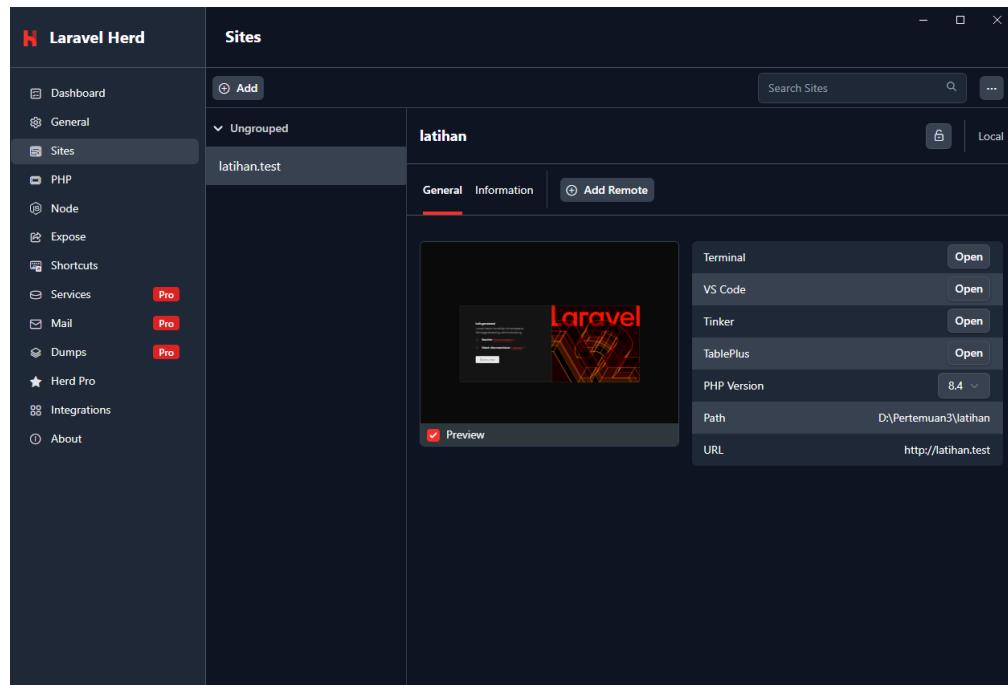
19. Tulis nama project name, dan kita point kan target location ke dalam folder latihan



20. Klik next dan tunggu hingga beres



21. Jika sudah silahkan klik “Open VSCode”



22. Lalu buka terminal vs code, tuliskan perintah “composer run dev”

```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS ... + x
PS D:\Pertemuan3\latihan> composer run dev
> Composer\Config::disableProcessTimeout
> npx concurrently -c "#93cfcd:#4b5fd:#fda74" "php artisan serve" "php artisan queue:listen --tries=1" "npm run dev" --names='server,queue,vite'
(node:5280) ExperimentalWarning: CommonJS module C:\Program Files\nodejs\node_modules\npm\node_modules\debug\src\node.js is loading ES Module C:\Program Files\nodejs\node_modules\npm\node_modules\supports-color\index.js using require()
Support for loading ES Module in require() is an experimental feature and might change at any time
(use node --trace-warnings ... to show where the warning was created)
[vite] > dev
[vite] > vite
[vite]
[queue]
[queue]   INFO Processing jobs from the [default] queue.
[queue]
[server]
[server]   INFO Server running on [http://127.0.0.1:8000].
[server]
[server] Press Ctrl+C to stop the server
[server]
[vite]
[vite]   VITE v7.2.2 ready in 1042 ms
[vite]
[vite]   Local:  http://localhost:5173/
[vite]   Network: use --host to expose
[vite]
[vite]   LARAVEL v12.37.0 plugin v2.0.1
[vite]
[vite]   APP_URL: http://latihan.test
```

The terminal output shows the execution of the 'composer run dev' command. It starts by disabling process timeout for Composer configuration. Then, it runs 'npx concurrently' with three tasks: 'php artisan serve' (localhost:8000), 'php artisan queue:listen --tries=1' (processing jobs from the default queue), and 'npm run dev' (VITE v7.2.2 ready in 1042 ms). Finally, it provides local and network URLs for the application.

23. Untuk mengetahui apakah laravel bisa berjalan atau tidak, silahkan tulis di browser <http://127.0.0.1:8000>

B. Struktur Folder Laravel 12

Laravel merupakan salah satu framework PHP berbasis MVC (Model-View-Controller) yang memiliki struktur folder rapi dan terorganisir. Setiap folder di dalamnya memiliki fungsi spesifik untuk memisahkan antara logika, tampilan, dan data. Memahami struktur ini sangat penting agar pengembangan aplikasi lebih mudah, terstruktur, dan efisien. Dalam praktik ini, kita akan mempelajari fungsi dari beberapa folder utama:

1. Folder app/

Folder ini merupakan inti dari aplikasi Laravel. Semua logika utama dan komponen aplikasi disimpan di sini. Secara default, di dalamnya terdapat beberapa sub-folder penting:

- **Http/Controller/**

Yaitu tempat menyimpan file controller. Controller sendiri berfungsi sebagai penghubung antara model (data) dan view (tampilan).

Mari kita praktek!

Silahkan jalankan perintah artisan dibawah ini di terminal:

php artisan make:controller PostController

php artisan make:controller CategoryController

lalu tambahkan method baru di dalamnya:

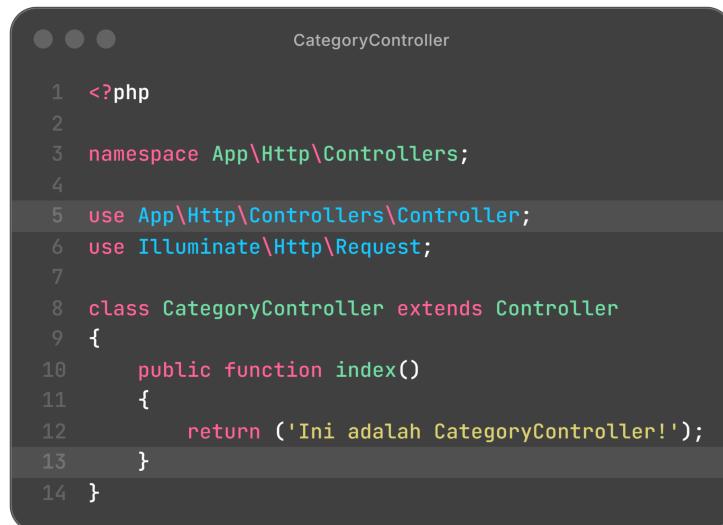
- PostController



```
PostController

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use App\Http\Controllers\Controller;
6 use Illuminate\Http\Request;
7
8 class PostController extends Controller
9 {
10     public function index()
11     {
12         return ('Ini adalah PostController!');
13     }
14 }
```

- CategoryController



```
CategoryController

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use App\Http\Controllers\Controller;
6 use Illuminate\Http\Request;
7
8 class CategoryController extends Controller
9 {
10     public function index()
11     {
12         return ('Ini adalah CategoryController!');
13     }
14 }
```

- Models/

Yaitu berisi file model untuk mendefinisikan struktur data dan relasi database.

2. Folder resources/

Folder ini digunakan untuk menyimpan semua file view (tampilan) dan asset yang digunakan sebelum dikompilasi. Secara default di dalamnya terdapat beberapa sub-folder penting:

- **resources/views/**

Di dalamnya berisi file tampilan Blade, seperti welcome.blade.php yang otomatis dibuat ketika membuat aplikasi laravel.

- **resources/css/ dan resources/js/**

Folder ini berisi sumber daya CSS dan JavaScript yang dapat diolah lebih lanjut sebelum dipublikasikan ke browser.

3. Folder routes/

Folder ini menyimpan semua definisi rute (URL) aplikasi, yang menentukan bagaimana permintaan (request) ditangani. Terdapat beberapa file penting:

- **web.php**

Biasanya digunakan untuk membuat route berbasis web (mengembalikan tampilan view).

- **api.php**

Biasanya digunakan untuk route API.

Mari kita praktik!

Di dalam file web.php buatlah route untuk menggunakan method yang ada di PostController dan CategoryController.



```
web.php

1 <?php
2
3 use App\Http\Controllers\CategoryController;
4 use App\Http\Controllers\PostController;
5 use Illuminate\Support\Facades\Route;
6
7 // contoh route untuk menampilkan view
8 Route::get('/', function () {
9     return view('welcome');
10 });
11
12 // Route untuk memanggil method di PostController
13 Route::get('posts', [PostController::class, 'index']);
14 Route::get('/categories', [CategoryController::class, 'index']);
15
```

4. Folder public/

Folder public adalah gerbang utama aplikasi laravel. semua file publik yang bisa diakses langsung dari browser disimpan di sini. Di dalamnya terdapat:

- **index.php**

Yaitu gerbang masuknya seluruh lalu lintas dimana semua request dari browser akan melewati file ini sebelum diteruskan ke sistem laravel.

- **css, js, images**

Digunakan untuk menyimpan asset-asset statis yang diperlukan oleh tampilan (views).

5. Folder database/

Folder database digunakan untuk mengelola seluruh file yang berkaitan dengan basis data (database) dalam aplikasi Laravel. Beberapa sub-folder di dalamnya terdapat:

- **migrations/**

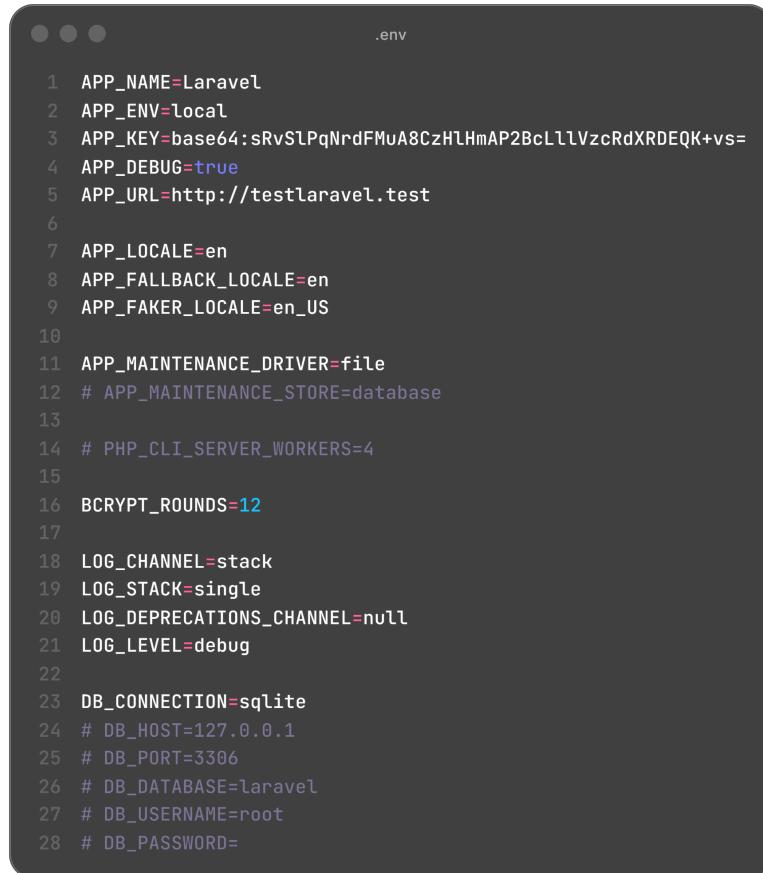
Folder ini berisi file-file migrasi yang digunakan untuk membuat atau mengubah struktur tabel database.

- **seeders/**

Biasanya digunakan untuk mengisi data awal (dummy data) ke dalam tabel.

6. File .env

File .env digunakan untuk menyimpan konfigurasi lingkungan (environment) aplikasi. misalnya, URL aplikasi, pengaturan database, atau API key. Contoh isi file .env:



```
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:sRv$LPqNrdFMuA8CzHlHmAP2BcLllVzcRdXRDEQK+vs=
4 APP_DEBUG=true
5 APP_URL=http://testlaravel.test
6
7 APP_LOCALE=en
8 APP_FALLBACK_LOCALE=en
9 APP_FAKE_LOCALE=en_US
10
11 APP_MAINTENANCE_DRIVER=file
12 # APP_MAINTENANCE_STORE=database
13
14 # PHP_CLI_SERVER_WORKERS=4
15
16 BCRYPT_ROUNDS=12
17
18 LOG_CHANNEL=stack
19 LOG_STACK=single
20 LOG_DEPRECATED_CHANNEL=null
21 LOG_LEVEL=debug
22
23 DB_CONNECTION=sqlite
24 # DB_HOST=127.0.0.1
25 # DB_PORT=3306
26 # DB_DATABASE=laravel
27 # DB_USERNAME=root
28 # DB_PASSWORD=
```

C. View & Blade Templating Laravel 12

1. View

Digunakan untuk pembuatan tampilan halaman web HTML. Dengan view, kita bisa membedakan logic aplikasi, dengan kode tampilan. Semua view disimpan di folder resources/views.

2. Blade Templating

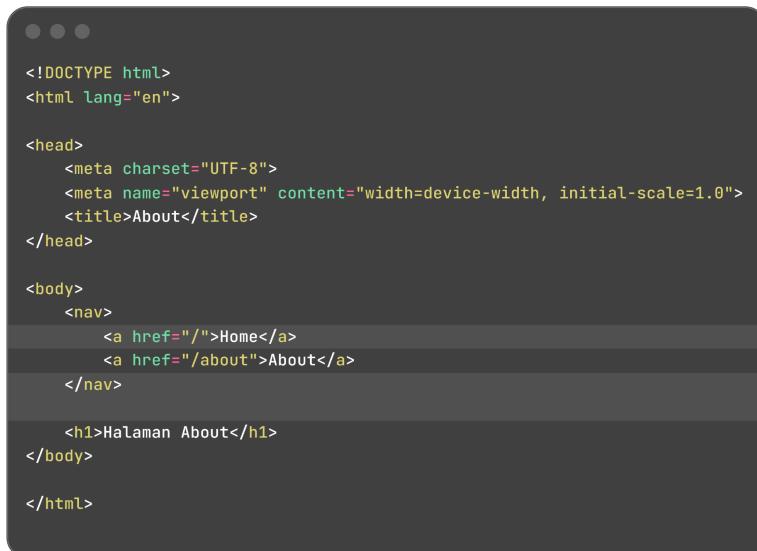
Laravel menggunakan **template engine** yang bernama **Blade** untuk membuat kode View nya. Blade menggunakan extension **blade.php** sebagai penamaan file nya, misal **index.blade.php**.

3. Konsep Blade

Blade Components adalah fitur utama untuk membuat layout dan elemen UI yang dapat digunakan kembali (**reusable**). Konsep ini memungkinkan kita mendefinisikan struktur halaman (seperti navigasi, header, dan footer) dalam satu file komponen yang kemudian dapat dipanggil dengan sintaks tag HTML di semua view lainnya.

Mari Kita Praktek!

- Buat halaman About menggunakan Blade
- Buat Link/Anchor Home dan About



```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>About</title>
</head>

<body>
    <nav>
        <a href="/">Home</a>
        <a href="/about">About</a>
    </nav>

    <h1>Halaman About</h1>
</body>

</html>
```

Mari Kita Praktek!

- Buat Component Layout : php artisan make:component Layout —view
- Pindahkan tag Nav ke component Layout.

resources/views/components/layout.blade.php

```
<!DOCTYPE html>
<html lang="en">

<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">
    <title>Document</title>
</head>

<body>
    <nav>
        <a href="/">Home</a>
        <a href="/about">About</a>
    </nav>
</body>

</html>
```

Blade Home dan About menjadi seperti ini sekarang

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home</title>
</head>

<body>
    <h1>Halaman Home</h1>
</body>

</html>
```

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>About</title>
</head>

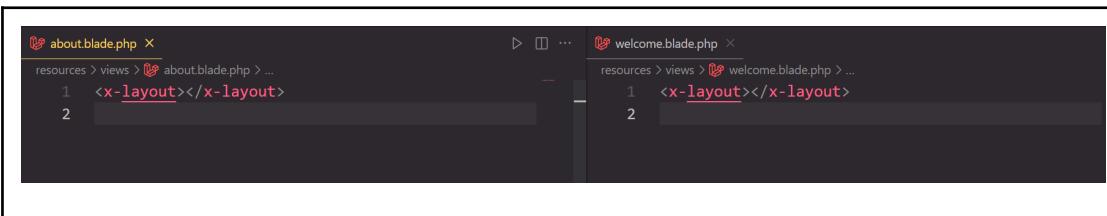
<body>
    <h1>Halaman About</h1>
</body>

</html>
```

- Sekarang hapus semua isi Home dan About, hanya ada Component Layout yang dipanggil.

Cara memanggil Component:

<x-layout></x-layout> atau bisa pake self closing tag <x-layout />



```
about.blade.php x
resources > views > about.blade.php > ...
1  <x-layout></x-layout>
2

welcome.blade.php x
resources > views > welcome.blade.php > ...
1  <x-layout></x-layout>
2
```

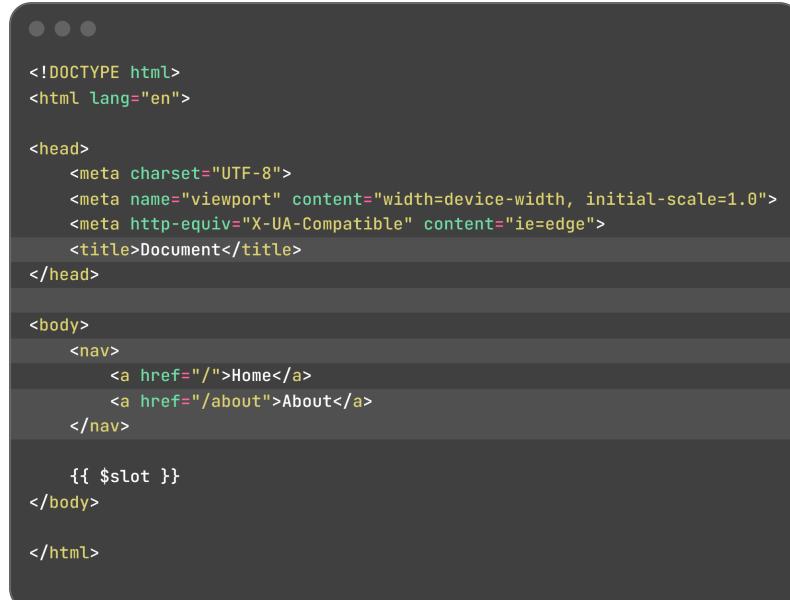
4. Slot

Karena sekarang kita telah menggunakan Layout Component. Untuk menambahkan konten di masing-masing file (home & about), kita perlu membuat agar Layout Component menjadi Dinamis dengan menggunakan fitur **\$slot**.

Mari Kita Praktek!

- Tambahkan {{ \$slot }} ke dalam Layout Component
- Tambahkan konten di masing-masing file dan lihat hasilnya

resources/views/components/layout.blade.php



```
<!DOCTYPE html>
<html lang="en">

<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">
    <title>Document</title>
</head>

<body>
    <nav>
        <a href="/">Home</a>
        <a href="/about">About</a>
    </nav>

    {{ $slot }}
</body>

</html>
```

home.blade.php dan about.blade.php

```
<x-layout>
    <h1>Halaman About</h1>
</x-layout>
```

```
<x-layout>
    <h1>Halaman Home</h1>
</x-layout>
```

5. Named Slot

Walaupun sudah mengimplementasikan Layout Component dan menggunakan Slot untuk menampung seluruh konten body. Namun, tag title masih bersifat statis. Untuk mengatasi masalah ini dan membuat title halaman menjadi dinamis, kita perlu menggunakan Named Slot.

Mari Kita Praktek!

resources/views/components/layout.blade.php:

```
<!DOCTYPE html>
<html lang="en">

    <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">
        {{-- Tambahkan slot baru dengan nama $title --}}
        <title>{{ $title }}</title>
    </head>

    <body>
        <nav>
            <a href="/">Home</a>
            <a href="/about">About</a>
        </nav>

        {{ $slot }}
    </body>

</html>
```

- Panggil slot title yang telah dibuat pada Home dan About
- resources/views/about.blade.php:

```
<x-layout>
    {{-- Cara memanggil slot $title yang telah dibuat --}}
    <x-slot:title>
        About
    </x-slot:title>
    <h1>Halaman About</h1>
</x-layout>
```

Mari Latihan!

- Buat Footer di dalam Layout.blade.php
- Buat view Blog dan Contact
- Terapkan Tailwind pada Layout (opsional)

D. Migration

1. Pengertian Migration

Migration di Laravel adalah versi “git” untuk database. Kalau git menyimpan perubahan kode, migration menyimpan perubahan struktur tabel database. Bayangkan kamu sedang membangun rumah:

- *Blueprint* = Migration
- *Bangunan fisik* = Database

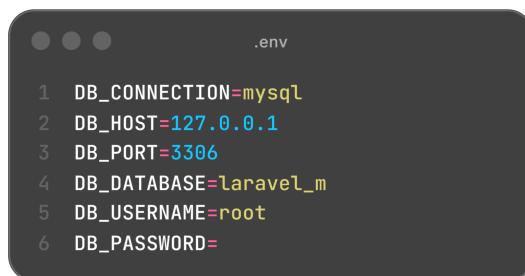
Jadi setiap kali kita “ubah blueprint”, Laravel bisa otomatis membangun ulang struktur database sesuai file migration.

2. Menghubungkan Database (SQLite)

Kita akan menggunakan SQLite, database ringan yang tidak butuh server tambahan seperti MySQL atau PostgreSQL. SQLite menyimpan semua data dalam satu file tunggal, mirip seperti file Excel, tapi jauh lebih canggih dan terintegrasi dengan Laravel.

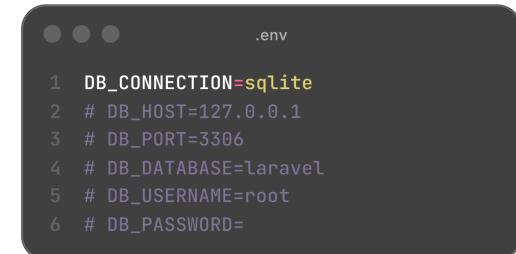
Mari Kita Praktek!

- Pertama buka visual studio code dan buka file .env



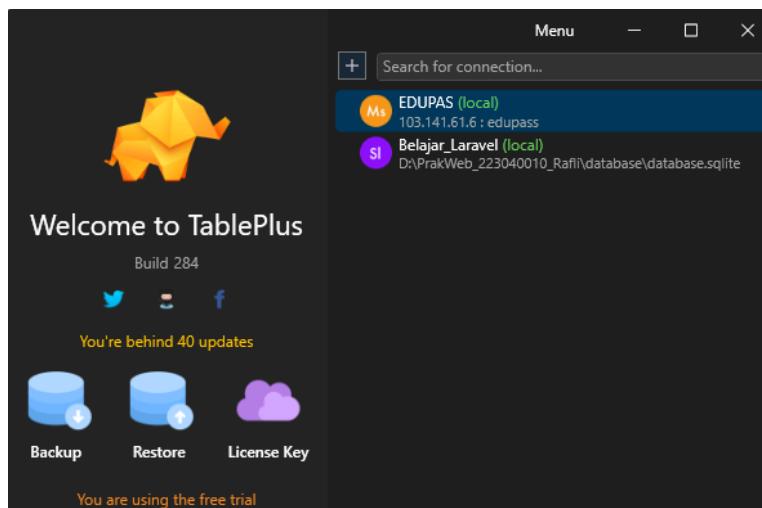
```
.env
1 DB_CONNECTION=mysql
2 DB_HOST=127.0.0.1
3 DB_PORT=3306
4 DB_DATABASE=laravel_m
5 DB_USERNAME=root
6 DB_PASSWORD=
```

- Secara default di env seperti gambar diatas
- Selanjutnya kita modifikasi seperti gambar dibawah

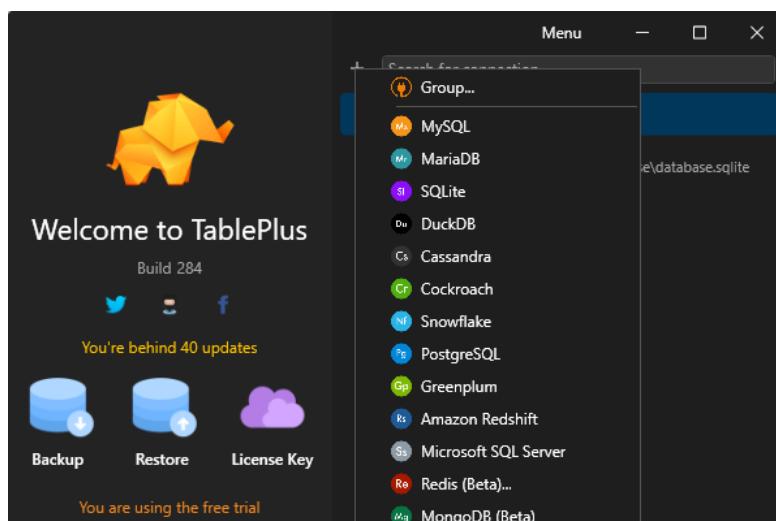


```
.env
1 DB_CONNECTION=sqlite
2 # DB_HOST=127.0.0.1
3 # DB_PORT=3306
4 # DB_DATABASE=laravel
5 # DB_USERNAME=root
6 # DB_PASSWORD=
```

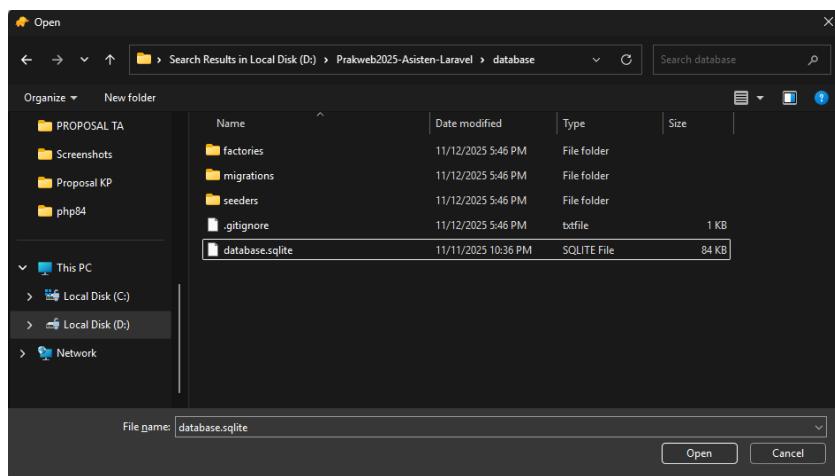
- “#” adalah komentar untuk mengabaikan semua baris kode
- Setelah di modifikasi kalian buka Aplikasi TablePlus



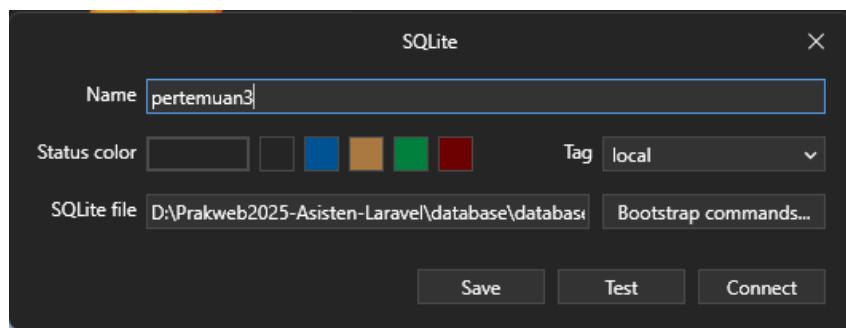
- Tekan icon “+” setelah itu klik SQLite



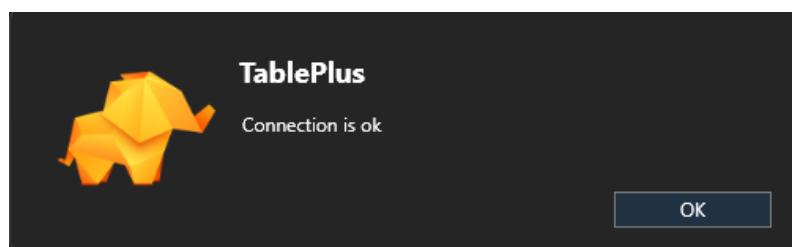
- Buat SQLite dengan nama “pertemuan3” dan klik “Select SQLite database file...”
- Cari folder yang kalian telah buat sebelumnya, terus masuk ke database dan setelah itu klik database.sqlite dan klik open



- Setelah klik “open” maka akan muncul



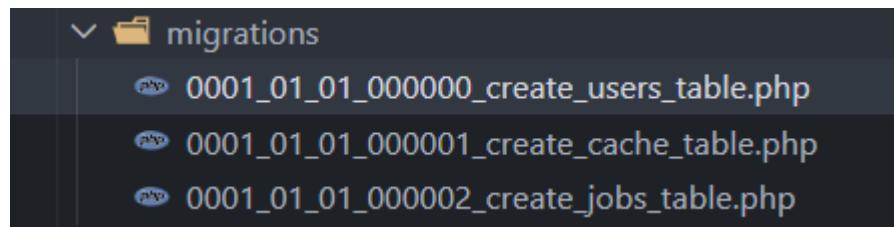
- Silahkan klik “Test” apakah sudah terhubung atau belum
- Jika berhasil maka tampilan akan seperti dibawah



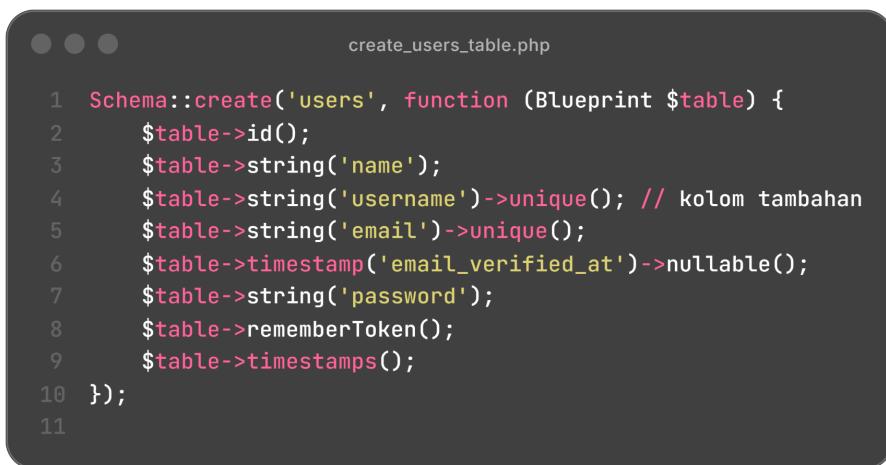
- kalian bisa klik “ok” dan klik “Save”

3. Struktur Migration default di Laravel

Laravel secara otomatis sudah menyediakan beberapa file migration di folder:



Akan tetapi kita modifikasi lagi untuk file _create_users_table.php



```
create_users_table.php

1 Schema::create('users', function (Blueprint $table) {
2     $table->id();
3     $table->string('name');
4     $table->string('username')->unique(); // kolom tambahan
5     $table->string('email')->unique();
6     $table->timestamp('email_verified_at')->nullable();
7     $table->string('password');
8     $table->rememberToken();
9     $table->timestamps();
10 });
11
```

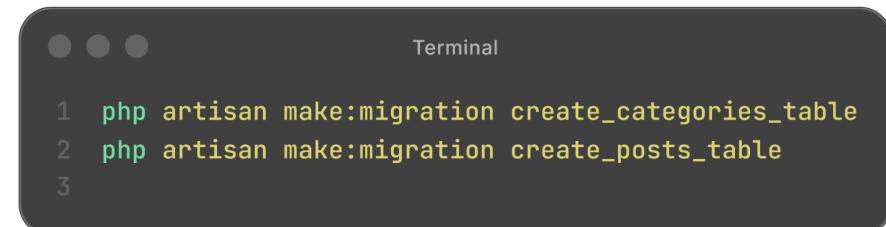
4. Membuat Migration Baru untuk Category dan Post

Kita akan membuat dua tabel tambahan:

- categories → untuk menyimpan nama kategori.
- posts → untuk menyimpan data postingan.

Mari Praktek!

- Buka terminal kalian lalu ketik seperti gambar dibawah



```
Terminal

1 php artisan make:migration create_categories_table
2 php artisan make:migration create_posts_table
3
```

- Setelah itu buka file migration “create_categories_table.php” lalu modifikasi seperti yang ada di gambar

```
create_categories_table.php

1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 return new class extends Migration
8 {
9     public function up(): void
10    {
11        Schema::create('categories', function (Blueprint $table) {
12            $table->id();
13            $table->string('name');
14            $table->timestamps();
15        });
16    }
17
18    public function down(): void
19    {
20        Schema::dropIfExists('categories');
21    }
22};
23
```

- Setelah itu buka file migration “create_posts_table.php” lalu modifikasi seperti yang ada di gambar

```
create_posts_table.php

1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 return new class extends Migration
8 {
9     public function up(): void
10    {
11        Schema::create('posts', function (Blueprint $table) {
12            $table->id();
13            $table->foreignId('user_id')->constrained('users')-
>onDelete('cascade');
14            $table->foreignId('category_id')->constrained('categories')-
>onDelete('cascade');
15            $table->string('title');
16            $table->string('slug')->unique();
17            $table->text('excerpt');
18            $table->text('body');
19            $table->string('image')->nullable();
20            $table->timestamps();
21        });
22    }
23
24    public function down(): void
25    {
26        Schema::dropIfExists('posts');
27    }
28};
29
```

- Penjelasan:
 - `user_id` → relasi ke tabel `users`.
 - `category_id` → relasi ke tabel `categories`.
 - `onDelete('cascade')` → jika user atau kategori dihapus, postingannya ikut terhapus.
- Jalankan semua migration di terminal

```
1  php artisan migrate:fresh
2
```

E. Model

1. Penjelasan: Apa itu Model?

Model di Laravel adalah representasi langsung dari tabel database. Jika di database ada tabel `users`, maka di Laravel biasanya ada model bernama `User`.

Analogi:

Bayangkan database kamu adalah lemari arsip besar.

- Setiap laci adalah tabel (misal: laci “users”, laci “posts”).
- Di dalam setiap laci ada banyak berkas (baris data).
- Nah, Model adalah *petugas arsip* yang tahu cara membuka laci, membaca berkas, dan menulis data baru.

Dengan model, kamu bisa:

- Mengambil data:

```
1 $users = User::all();  
2
```

- Menambah data:

```
1 User::create(['name' => 'Rafli']);
```

Menjalin relasi antar tabel (misalnya 1 user punya banyak post).

2. Apa itu ORM (Object Relational Mapping)?

ORM (Object Relational Mapping) adalah teknik untuk mengelola database melalui objek dalam kode. Laravel menyediakan ORM-nya sendiri yang disebut Eloquent.

- Tanpa ORM (manual SQL):

```
...  
1 SELECT * FROM users WHERE id = 1;
```

- Dengan ORM (Eloquent):

```
...  
1 $user = User::find(1);
```

Eloquent otomatis menerjemahkan perintah PHP menjadi SQL yang dijalankan di database, sehingga kita bisa bekerja lebih cepat dan aman.

3. Membuat Model

Mari Kita Praktek!

- Jalankan perintah berikut di terminal:

```
... Membuat Model  
1 php artisan make:model Category  
2 php artisan make:model Post
```

- Setelah itu buka file Models/User kita modifikasi terlebih dahulu

```

Model User

1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Foundation\Auth\User as Authenticatable;
7 use Illuminate\Notifications\Notifiable;
8 use Illuminate\Database\Eloquent\Relations\hasMany;
9
10 class User extends Authenticatable
11 {
12     use HasFactory, Notifiable;
13
14     // Kolom yang boleh diisi secara mass assignment
15     protected $fillable = [
16         'name',           // Nama lengkap user
17         'username',       // Username unik untuk login
18         'email',          // Email unik untuk login
19         'password',       // Password yang akan di-hash otomatis
20     ];
21
22     // Kolom yang disembunyikan saat serialisasi (response JSON/Array)
23     protected $hidden = [
24         'password',       // Jangan tampilkan password di response
25         'remember_token', // Jangan tampilkan token di response
26     ];
27
28     // Tipe data casting untuk kolom tertentu
29     protected function casts(): array
30     {
31         return [
32             'email_verified_at' => 'datetime', // Cast ke object DateTime
33             'password' => 'hashed',           // Otomatis hash password saat
34             insert/update
35         ];
36
37     // Relasi: Satu user memiliki banyak posts (One-to-Many)
38     public function posts(): HasMany
39     {
40         return $this->hasMany(Post::class, 'user_id');
41         // 'user_id' adalah foreign key di tabel posts yang menunjuk ke
42         // users.id
43     }
44 }

```

- Setelah dimodifikasi masuk ke Models/category

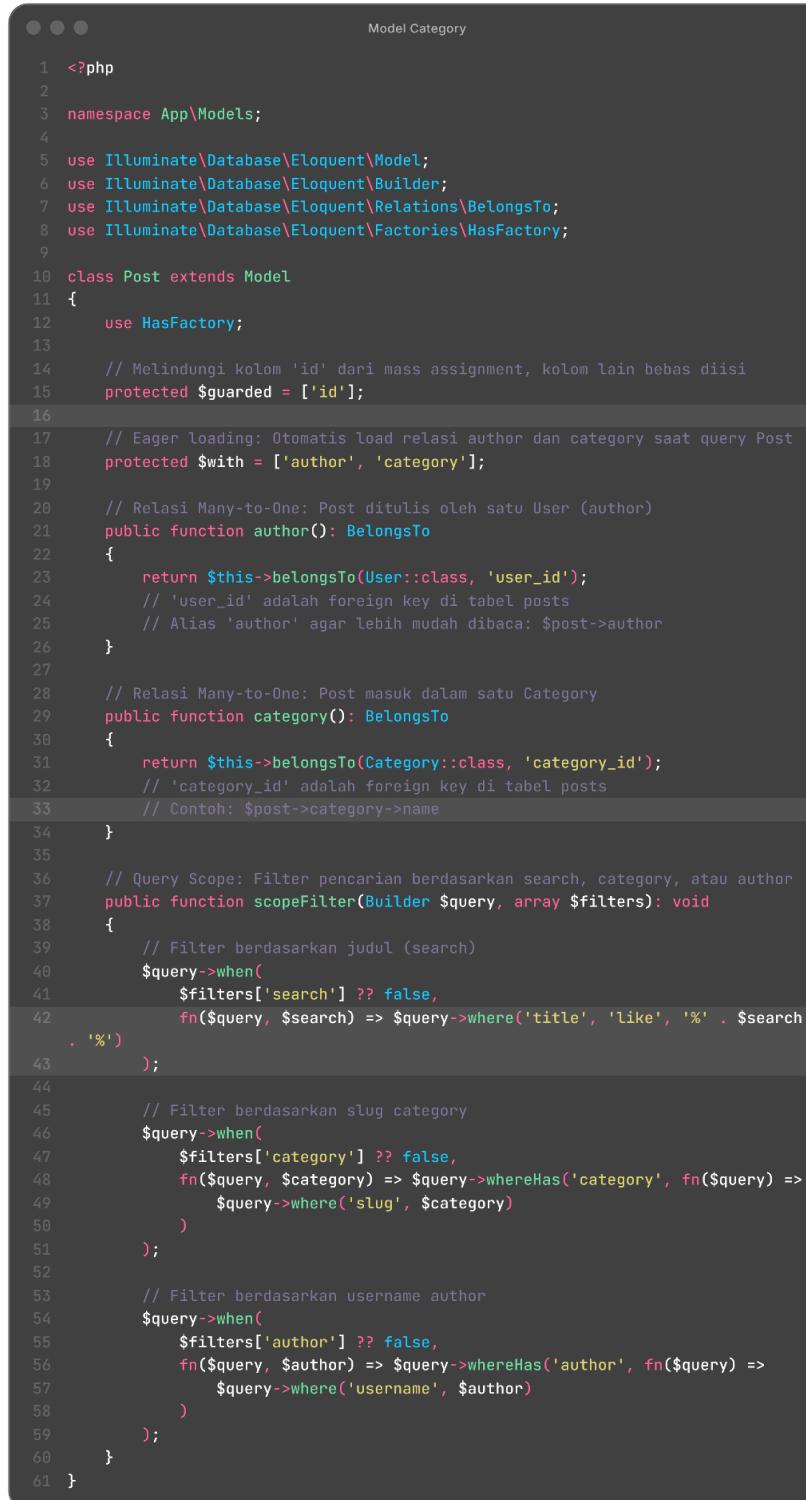
```

Model Category

1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7 use Illuminate\Database\Eloquent\Relations\hasMany;
8
9 class Category extends Model
10 {
11     use HasFactory;
12
13     // Kolom yang dilindungi dari mass assignment (hanya 'id' yang tidak boleh
14     // diisi manual)
15     protected $guarded = ['id'];
16
17     // Relasi: Satu category memiliki banyak posts (One-to-Many)
18     public function posts(): HasMany
19     {
20         return $this->hasMany(Post::class, 'category_id');
21         // 'category_id' adalah foreign key di tabel posts yang menunjuk ke
22         // categories.id
23     }
24 }

```

- Terakhir kita masuk ke Models/Post



```
Model Category

1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Model;
6 use Illuminate\Database\Eloquent\Builder;
7 use Illuminate\Database\Eloquent\Relations\BelongsTo;
8 use Illuminate\Database\Eloquent\Factories\HasFactory;
9
10 class Post extends Model
11 {
12     use HasFactory;
13
14     // Melindungi kolom 'id' dari mass assignment, kolom lain bebas diisi
15     protected $guarded = ['id'];
16
17     // Eager loading: Otomatis load relasi author dan category saat query Post
18     protected $with = ['author', 'category'];
19
20     // Relasi Many-to-One: Post ditulis oleh satu User (author)
21     public function author(): BelongsTo
22     {
23         return $this->belongsTo(User::class, 'user_id');
24         // 'user_id' adalah foreign key di tabel posts
25         // Alias 'author' agar lebih mudah dibaca: $post->author
26     }
27
28     // Relasi Many-to-One: Post masuk dalam satu Category
29     public function category(): BelongsTo
30     {
31         return $this->belongsTo(Category::class, 'category_id');
32         // 'category_id' adalah foreign key di tabel posts
33         // Contoh: $post->category->name
34     }
35
36     // Query Scope: Filter pencarian berdasarkan search, category, atau author
37     public function scopeFilter(Builder $query, array $filters): void
38     {
39         // Filter berdasarkan judul (search)
40         $query->when(
41             $filters['search'] ?? false,
42             fn($query, $search) => $query->where('title', 'like', '%' . $search
43             . '%')
44         );
45
46         // Filter berdasarkan slug category
47         $query->when(
48             $filters['category'] ?? false,
49             fn($query, $category) => $query->whereHas('category', fn($query) =>
50                 $query->where('slug', $category)
51             )
52         );
53
54         // Filter berdasarkan username author
55         $query->when(
56             $filters['author'] ?? false,
57             fn($query, $author) => $query->whereHas('author', fn($query) =>
58                 $query->where('username', $author)
59             )
60         );
61     }
}
```

F. Seeder & Factory

1. Penjelasan Seeder & Factory

Setelah struktur database (migration) dan model dibuat, database masih kosong. Seeder dan Factory membantu mengisi data otomatis untuk pengujian.

Analogi:

- Migration = membangun rumah (struktur).
- Model = menentukan siapa penghuninya.
- Seeder + Factory = mengisi rumah dengan furnitur dan orang-orangnya.

Factory membuat pola data (bagaimana data dummy terlihat). **Seeder** mengeksekusi factory untuk memasukkan data ke database.

2. Membuat Factory

Jalankan di terminal:

```
php artisan make:factory UserFactory --model=User  
php artisan make:factory CategoryFactory --model=Category  
php artisan make:factory PostFactory --model=Post
```

3. Isi File Factory

Mari Kita Praktek!

- database/factories/UserFactory.php

```
database/factories/UserFactory.php  
  
use Illuminate\Support\Facades\Hash;  
  
public function definition(): array  
{  
    return [  
        'name' => fake()->name(),  
        'username' => fake()->unique()->userName(),  
        'email' => fake()->unique()->safeEmail(),  
        'password' => Hash::make('password'), // default password  
    ];  
}
```

- database/factories/CategoryFactory.php

```
database/factories/CategoryFactory.php

public function definition(): array
{
    return [
        'name' => fake()->unique()->word(),
    ];
}
```

- database/factories/PostFactory.php

```
database/factories/PostFactory.php

use Illuminate\Support\Str;

public function definition(): array
{
    $title = fake()->sentence(4);

    return [
        'user_id' => \App\Models\User::factory(),
        'category_id' => \App\Models\Category::factory(),
        'title' => $title,
        'slug' => Str::slug($title),
        'excerpt' => fake()->paragraph(),
        'body' => fake()->paragraphs(3, true),
        'image' => null,
    ];
}
```

4. Membuat Seeder

Buka file database/seeders/DatabaseSeeder.php, setelah itu ubah kodennya seperti gambar dibawah

```
<?php

namespace Database\Seeders;

use App\Models\User;
use App\Models\Category;
use App\Models\Post;
use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    use WithoutModelEvents;

    /**
     * Seed the application's database.
     */
    public function run(): void
    {
        // Membuat 10 User secara manual dengan username user1-user10
        for ($i = 1; $i <= 10; $i++) {
            User::create([
                'name' => 'User ' . $i,
                'username' => 'user' . $i,
                'email' => 'user' . $i . '@example.com',
                'password' => bcrypt('password'),
            ]);
        }

        // Membuat Category secara otomatis
        Category::factory(5)->create();

        // Membuat Post secara otomatis (akan otomatis assign ke user dan category
        // yang ada)
        Post::factory(50)->recycle(User::all())->recycle(Category::all())->create();
    }
}
```

Setelah itu jalankan di terminal

```
Menjalankan Seeder
php artisan migrate:fresh --seed
```

G. Data dan Relasi Laravel 12

Laravel menyediakan sistem Eloquent ORM (Object Relational Mapping) untuk mempermudah pengelolaan data di database. Dengan Eloquent, setiap tabel di database diwakili oleh satu model dalam kode PHP. Salah satu fitur utama Eloquent adalah kemampuannya dalam mengatur relasi antar data.

Relasi adalah cara untuk menghubungkan satu tabel dengan tabel lainnya. Contohnya: satu user memiliki satu profile, atau satu user memiliki banyak posts. Laravel menyediakan berbagai jenis relasi, dan dua di antaranya yang paling umum adalah:

1. Relasi One to One

Relasi one to one (satu ke satu) adalah hubungan di mana satu baris data pada tabel pertama berhubungan dengan tepat satu baris pada tabel kedua. Misalnya, satu user memiliki satu profile.



The screenshot shows a code editor window with the title "Contoh Relasi One to One". The code is written in PHP and uses Eloquent relationships. It defines two models: User and Profile. The User model has a method named "profile" which returns a single Profile object using the "hasOne" relationship. The Profile model has a method named "user" which returns a single User object using the "belongsTo" relationship. The code is as follows:

```
1 // User.php
2 public function profile()
3 {
4     return $this->hasOne(Profile::class);
5 }
6
7 // Profile.php
8 public function user()
9 {
10    return $this->belongsTo(User::class);
11 }
```

2. Relasi One to Many

Relasi one to many (satu ke banyak) adalah hubungan di mana satu baris data di tabel pertama dapat memiliki banyak baris terkait di tabel kedua. Contohnya, satu user memiliki banyak posts.

Contoh Relasi One to Many

```
1 // User.php
2 public function posts()
3 {
4     return $this->hasMany(Post::class);
5 }
6
7 // Post.php
8 public function user()
9 {
10    return $this->belongsTo(User::class);
11 }
```

Mari kita praktek!

Sekarang kita akan coba membuat halaman posts yang menampilkan daftar post yang ada di dalam tabel posts.

- PostController di app/Http/Controller/

```
PostController

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use App\Http\Controllers\Controller;
6 use App\Models\Post;
7 use Illuminate\Http\Request;
8
9 class PostController extends Controller
10 {
11     public function index()
12     {
13         $posts = Post::all();
14         return view('posts', compact('posts'));
15     }
16 }
17
```

- posts.blade.php di resource/views/

```
posts.blade.php

1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>Halaman Daftar Posts</title>
8 </head>
9
10 <body>
11     <h1>Daftar Posts</h1>
12
13     @foreach ($posts as $post)
14         <article>
15             <h2><a href="/posts/{{ $post->slug }}">{{ $post->title }}</a></h2>
16             <p>{{ $post->excerpt }}</p>
17         </article>
18     @endforeach
19 </body>
20
21 </html>
22
```

- web.php di routes/

```
web.php

1 <?php
2
3 use App\Http\Controllers\CategoryController;
4 use App\Http\Controllers\PostController;
5 use Illuminate\Support\Facades\Route;
6
7 // contoh route untuk menampilkan view
8 Route::get('/', function () {
9     return view('welcome');
10 });
11
12 // Route untuk memanggil method di PostController
13 Route::get('/posts', [PostController::class, 'index'])->name('posts.index');
14
```

TUGAS PRAKTEK

1. Buat 10 data posts, 5 Users, dan 2 Category bebas kedalam seeders
2. Buat halaman '/categories' yang menampilkan semua kategori.

REFERENSI

Playlist Laravel	https://www.youtube.com/watch?v=00o1vJYTp4I&list=PLFIM0718LjIWIXb7cVj7LdAr32ATDQMdr
WPUCourse - Belajar Laravel	https://wpucourse.id/course/belajar-laravel