#### Daftar Isi

[BAB 1 1](#_Toc2037107120)

[PENGENALAN LARAVEL 1](#_Toc221889291)

[1.1 Fitur Utama : 1](#_Toc89908724)

[1.2 Keunggunlan 1](#_Toc1746603592)

[1.3 Kekurangan 1](#_Toc1149975181)

[1.4 Instalasi Laravel 2](#_Toc1854991867)

[1.5 Praktek: Membuat Halaman Web Sederhana 2](#_Toc720489811)

[BAB 2 4](#_Toc202057940)

[Struktur Direktori dan Konsep Dasar 4](#_Toc1713135201)

[2.1 Struktur Direktori Laravel 4](#_Toc116131623)

[2.2 Konsep Dasar MVC (Model-View-Controller) 5](#_Toc1466272488)

[2.3 Routing Laravel 6](#_Toc663779305)

[2.4 Praktikum: Membuat Controller dan View Sederhana 6](#_Toc136217053)

[2.4.1 Membuat Controller 6](#_Toc468672137)

[2.4.2 Menghubungkan Controller dengan Route 6](#_Toc1212658188)

[2.4.3 Mengakses View dari Browser 7](#_Toc2052657796)

[Bab 3 8](#_Toc1934005850)

[Routing dan Controller 8](#_Toc1396809677)

[3.1 Pengertian Routing 8](#_Toc1982713583)

[3.2 Jenis-jenis Routing 8](#_Toc1525532995)

[3.3 Group Routing 9](#_Toc1261879296)

[3.4 Pengertian Controller 9](#_Toc37376301)

[3.5 Membuat Controller 9](#_Toc2087711882)

[3.6 Menghubungkan Controller dengan Route 10](#_Toc245379365)

[3.7 Middleware 11](#_Toc222338406)

[3.8 Praktek: Membuat Aplikasi CRUD Sederhana 11](#_Toc480894001)

[3.8.1 Membuat Model dan Migration 11](#_Toc1776180792)

[3.8.2 Menghubungkan Controller dengan Model 12](#_Toc717037247)

[3.8.3 Membuat View untuk Produk 13](#_Toc533246092)

[Bab 4: 15](#_Toc1876385388)

[Membuat dan Mengelola Database dengan Laravel 15](#_Toc2138303150)

[4.1 Pengantar Database di Laravel 15](#_Toc422869564)

[4.2 Konfigurasi Database 15](#_Toc2098274679)

[4.3 Migrasi (Migration) 15](#_Toc80728226)

[4.3.1 Membuat Migrasi 15](#_Toc21989508)

[4.3.2 Modifikasi Tabel dengan Migrasi 16](#_Toc1100766212)

[4.4 Eloquent ORM 17](#_Toc1935720094)

[4.4.1 Membuat Model 17](#_Toc742479320)

[4.4.2 Menggunakan Eloquent untuk Operasi CRUD 17](#_Toc1302824153)

[4.4.3 Relasi Antar Model 18](#_Toc1501371647)

[4.5 Seeder 18](#_Toc858610943)

[4.5.1 Membuat Seeder 18](#_Toc621612993)

[4.5.2 Menulis Seeder 19](#_Toc17667304)

[4.5.3 Menjalankan Seeder 19](#_Toc994827996)

[4.6 Fakta dan Pengujian Menggunakan Factory 19](#_Toc1090285131)

[4.6.1 Membuat Factory 19](#_Toc1230325492)

[4.6.2 Menggunakan Factory 19](#_Toc900002144)

[4.6.3 Menggunakan Factory di Seeder 20](#_Toc876807333)

[Bab 5: 21](#_Toc479651521)

[Blade Templating 21](#_Toc735232079)

[5.1 Pengantar Blade 21](#_Toc254856680)

[5.2 Sintaks Dasar Blade 21](#_Toc1741530818)

[5.3 Layout dan Penggunaan Template 22](#_Toc772608381)

[5.3.1 Membuat Layout Dasar 22](#_Toc195084914)

[5.3.2 Menggunakan Layout di View 23](#_Toc1986910183)

[5.3.3 Membuat Partial Views 23](#_Toc994946787)

[5.4 Blade Components dan Slots 24](#_Toc675978916)

[5.4.1 Membuat Component 24](#_Toc1615607327)

[5.4.2 Menggunakan Component 24](#_Toc1711984034)

[5.5 Directive Khusus Blade 25](#_Toc1209225008)

[5.6 Blade Templating untuk Pengembangan Lanjutan 25](#_Toc1344509067)

[5.7 Praktek: Membuat Halaman Dinamis 26](#_Toc1702803536)

[Bab 6: 27](#_Toc1632094572)

[CRUD dengan Laravel 27](#_Toc1295300098)

[6.1 Pengantar CRUD 27](#_Toc1783531763)

[6.2 Membuat Model dan Migrasi 27](#_Toc1654084080)

[6.2.1 Membuat Model dan Migrasi 27](#_Toc248582663)

[6.2.2 Menulis Skema Tabel di Migrasi 27](#_Toc1571768209)

[6.3 Membuat Controller dan Routing 28](#_Toc249079752)

[6.3.1 Membuat Controller 28](#_Toc1551406816)

[6.3.2 Menambahkan Routing 28](#_Toc925656208)

[6.4 Implementasi CRUD 28](#_Toc1107690695)

[6.4.1 Membuat Post Baru (Create) 28](#_Toc25536161)

[6.4.2 Membaca dan Menampilkan Post (Read) 29](#_Toc943323512)

[6.4.3 Mengedit dan Memperbarui Post (Update) 30](#_Toc2102518691)

[6.4.4 Menghapus Post (Delete) 31](#_Toc1115821292)

[6.5 Validasi Data 32](#_Toc26165356)

[6.6 Menyempurnakan UI dengan Blade 32](#_Toc855037188)

[6.7 Tugas: Membuat CRUD Lengkap 32](#_Toc1992628626)

[Bab 7: 33](#_Toc505816877)

[Relasi Antar Tabel 33](#_Toc1590269267)

[7.1 Pengantar Relasi Antar Tabel 33](#_Toc100001658)

[7.2 Relasi One-to-One 33](#_Toc99864047)

[7.2.1 Membuat Migrasi dan Model 33](#_Toc215394000)

[7.2.2 Mendefinisikan Relasi di Model 34](#_Toc295086573)

[7.2.3 Menggunakan Relasi One-to-One 34](#_Toc2086774230)

[7.3 Relasi One-to-Many 34](#_Toc1210340788)

[7.3.1 Membuat Migrasi dan Model 34](#_Toc971065489)

[7.3.2 Mendefinisikan Relasi di Model 35](#_Toc1554897909)

[7.3.3 Menggunakan Relasi One-to-Many 35](#_Toc774841174)

[7.4 Relasi Many-to-Many 35](#_Toc32806849)

[7.4.1 Membuat Tabel Pivot 36](#_Toc751923328)

[7.4.2 Mendefinisikan Relasi di Model 37](#_Toc330161063)

[7.4.3 Menggunakan Relasi Many-to-Many 37](#_Toc1664901421)

[7.5 Relasi Has Many Through 37](#_Toc2047223427)

[7.5.1 Mendefinisikan Relasi di Model 37](#_Toc2113692826)

[7.5.2 Menggunakan Relasi Has Many Through 38](#_Toc1171501853)

[7.6 Relasi Polimorfik 38](#_Toc148322442)

[7.6.1 Membuat Relasi Polimorfik 38](#_Toc1537977387)

BAB 1

PENGENALAN LARAVEL

Laravel adalah sebuah framework PHP yang open-source, dirancang untuk memudahkan pengembangan aplikasi web dengan struktur MVC (Model-View-Controller). Laravel menyediakan berbagai fitur yang membantu dalam mengembangkan aplikasi web yang aman, cepat, dan skalabel. Beberapa fitur andalannya termasuk routing, Eloquent ORM, Blade templating engine, dan sistem migrasi database.

1.1 Fitur Utama :

· **Routing**: Laravel menyediakan routing yang sederhana dan fleksibel.

· **Eloquent ORM**: ORM yang powerful untuk interaksi dengan database menggunakan model.

· **Blade Templating Engine**: Sistem templating yang efisien dan ringan.

· **Migration**: Sistem untuk mengelola skema database dengan mudah.

· **Middleware**: Mekanisme untuk mengatur request HTTP sebelum mencapai controller.

1.2 Keunggunlan

· **Pengkodean yang sederhana**: Hal ini sudah dirasakan oleh banyak pengembang yang menggunakan Laravel bahwa pengkodean biasanya menjadi salah satu hal yang sangat rumit, tetapi Laravel membuat beberapa tugas menjadi lebih mudah, hal ini dikarenakan Laravel memiliki library yang luas yang memuat fitur-fitur yang fungsionalitas yang telah deprogram sebelumnya.

· **Skalabilitas:** Laravel merupakan salah satu framework scalable yang memudahkan pengembang menggunakannya untuk aplikasi web berukuran kecil dan menengah.

· **Aman**: Laravel memiliki sistem kontrol akses bawaan yang aman dan juga menyediakan mekanisme tanggung yang memungkinkan pengembang menangani bug atau masalah dengan mudah, selain itu kerangka kerja ini memungkinkan pengembang untuk menyimpan kata sandi dalam format terenskripsi sehingga web mendapatkan perlindungan yang ekstra.

· **Migration**: Migrasi data jauh lebih sederhara jika menggunakan Laravel, karena migrasi data dilakukan dengan meminimalisir waktu daripada biasanya.

· **Mudah Dipelajari**: Memiliki basis komunitas yang besar.

1.3 Kekurangan

· **Pengkodean yang sederhanaKinerja Lambat**: Kinerja Laravel lebih lambat dibandingkan dengan beberapa framework lainnya, terutama ketika mengelola permintaan yang tinggi.

· **Update yang Terlalu Cepat:** Laravel memiliki pembaruan secara rutin, namum hal ini akan menyulitkan pada software yang tidak sesuai dengan spesifikasi yang dibutuhkan.

· **Ketergantungan pada Komposer**: Dengan ketergantungan ini, pengguna Laravel perlu memphatikan dan juga familiar dengan composer.

· **Ukuran yang berat**: Hal ini dikarenkan Laravel membuat lebih banyak aset-aset dalam menjalankan aplikasinya.

1.4 Instalasi Laravel

Sebelum menginstal Laravel, pastikan sistem kamu telah memiliki:

· **Composer**: Dependency manager untuk PHP.

· **PHP**: Minimal versi 8.0.

· **MySQL** atau database lain yang didukung.

Berikut ini langkah-langkah install Laravel melalui composer :

1. I**nstal Composer**: Jika belum terinstal, unduh dan instal Composer dari <https://getcomposer.org/.>
2. Membuat Proyek Laravel: Buka terminal atau command prompt, lalu jalankan perintah berikut:

composer create-project --prefer-dist laravel/laravel nama-proyek

Contoh:

composer create-project --prefer-dist laravel/laravel belajar-laravel

1. Menjalankan Server Laravel: Masuk ke direktori proyek yang baru dibuat:

cd belajar-laravel

Jalankan server Laravel dengan perintah:

php artisan serve

Setelah dijalankan, kamu dapat mengakses aplikasi Laravel melalui browser di alamat [http://localhost:8000](http://localhost:8000.) .

1.5 Praktek: Membuat Halaman Web Sederhana

Laravel menggunakan konsep routing untuk mengarahkan request ke controller yang tepat. Untuk membuat route sederhana, buka file routes/web.php dan tambahkan kode berikut:

1. use Illuminate\Support\Facades\Route;
3. Route::get('/hello', function () {
4. return 'Hello, Laravel!';
5. });

Kunjungi http://localhost:8000/hello di browser. Kamu akan melihat teks Hello, Laravel! di layar.

Laravel menggunakan Blade sebagai templating engine. Untuk membuat view, buat file baru di direktori resources/views bernama welcome.blade.php:

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <meta name="viewport" content="width=device-width, initial-scale=1.0">
6. <title>Welcome to Laravel</title>
7. </head>
8. <body>
9. <h>Welcome to Laravel</h>
10. <p>This is your first Blade template.</p>
11. </body>
12. </html>

Kemudian, ubah route /hello di web.php menjadi:

1. Route::get('/hello', function () {
2. return view('welcome');
3. });

Sekarang, ketika kamu mengunjungi http://localhost:8000/hello, kamu akan melihat halaman web sederhana yang menampilkan pesan "Welcome to Laravel".

BAB 2

Struktur Direktori dan Konsep Dasar

2.1 Struktur Direktori Laravel

Saat pertama kali menginstal Laravel, kamu akan melihat beberapa folder dan file di dalam direktori proyek. Berikut adalah penjelasan singkat mengenai struktur direktori Laravel:

1. app/

Direktori ini berisi seluruh logika bisnis aplikasi. Sub-direktori penting di sini adalah:

· Console/: Menyimpan semua perintah artisan kustom.

· Exceptions/: Menangani pengecualian atau error.

· Http/: Berisi controller, middleware, dan form request.

· Models/: Berisi model yang mewakili entitas database.

· Providers/: Berisi service provider yang digunakan untuk inisialisasi berbagai komponen aplikasi.

1. bootstrap/

Berisi file untuk mem-boot aplikasi. Folder ini juga berisi cache aplikasi yang disimpan di cache/.

1. bootstrap/

Menyimpan berbagai file konfigurasi aplikasi seperti database, mail, session, dll.

1. database/

Menyimpan file migrasi, seeder, dan factory. Folder ini juga bisa digunakan untuk menyimpan database SQLite.

1. public/

Merupakan entry point untuk semua permintaan HTTP. Folder ini berisi file index.php yang mem-boot Laravel, serta file frontend seperti CSS, JavaScript, gambar, dll.

1. resources/

Menyimpan file view Blade, file bahasa, dan aset frontend yang belum dikompilasi.

1. routes/

Berisi semua definisi route. Ada beberapa file penting di sini:

· web.php: Berisi definisi route untuk web (dengan stateful).

· api.php: Berisi definisi route untuk API (stateless).

· console.php: Mendefinisikan route untuk perintah artisan kustom.

· channels.php: Mendefinisikan semua channel broadcasting.

1. storage/

Menyimpan file yang dibutuhkan saat runtime seperti log, cache, dan file upload. Subdirektori penting meliputi:

· app/: Tempat penyimpanan file yang diakses oleh aplikasi.

· framework/: Penyimpanan cache framework, session, dan view.

· logs/: Menyimpan log aplikasi.

1. tests/

Berisi file untuk testing aplikasi. Laravel mendukung PHPUnit secara default.

1. vendor/

Menyimpan semua dependensi yang dikelola oleh Composer. Semua paket pihak ketiga ada di sini.

1. .env/

File ini berisi variabel lingkungan yang digunakan untuk konfigurasi aplikasi seperti database, mail, dan pengaturan lainnya.

1. composer.json/

File konfigurasi untuk Composer, yang berisi informasi tentang dependensi dan skrip yang digunakan dalam proyek.

2.2 Konsep Dasar MVC (Model-View-Controller)

Laravel menggunakan arsitektur MVC (Model-View-Controller) yang membagi aplikasi menjadi tiga bagian utama:

1. Model/

Model bertanggung jawab untuk berinteraksi dengan database. Setiap tabel dalam database biasanya memiliki model yang sesuai untuk menangani operasi CRUD. Model berada di direktori app/Models/.

Contoh:

1. namespace App\Models;
2. use Illuminate\Database\Eloquent\Model;
3. *class* Post extends Model {
4. protected $fillable = ['title', 'content'];
5. }
6. view/

View adalah bagian dari aplikasi yang menangani presentasi. View biasanya berupa file Blade (template engine Laravel) yang terletak di resources/views/.

contoh :

1. <!-- resources/views/welcome.blade.php -->
2. <h>Welcome to Laravel</h>
3. <p>{{ $message }}</p>
4. Controller/

Controller menghubungkan model dan view. Controller menangani logika aplikasi dan mengarahkan data dari model ke view. Controller berada di direktori app/Http/Controllers/.

contoh :

1. namespace App\Http\Controllers;
2. use App\Models\Post;
3. use Illuminate\Http\Request;
4. *class* PostController extends Controller {
5. public function index() {
6. $ $posts = Post::all();
7. return view('posts.index', ['posts' => $posts]);
8. }
9. }

2.3 Routing Laravel

Routing adalah konsep yang mengatur bagaimana aplikasi merespons permintaan URL tertentu. Laravel menggunakan file routes/web.php untuk mendefinisikan rute web.

Contoh Routing Sederhana :

1. use Illuminate\Support\Facades\Route;
2. Route::get('/', function () {
3. return view('welcome');
4. });
5. Route::get('/posts', [*PostController*::*class*, 'index']);

2.4 Praktikum: Membuat Controller dan View Sederhana

Controller berisi program logic untuk pembacaan data dari model dan juga pengiriman data ke view.

2.4.1 Membuat Controller

1. Jalankan perintah artisan untuk membuat controller baru:

1. php artisan make:controller ExampleController

2. Tambahkan fungsi ke dalam controller untuk menangani request:

1. namespace App\Http\Controllers;
2. use Illuminate\Http\Request;
3. *class* ExampleController extends Controller {
4. public function showWelcome() {
5. $ $message = 'This is a message from the controller';
6. return view('welcome', ['message' => $message]);
7. }
8. }

2.4.2 Menghubungkan Controller dengan Route

Tambahkan rute di routes/web.php untuk menghubungkan URL ke controller yang baru saja dibuat:

1. Route::get('/welcome', [*ExampleController*::*class*, 'showWelcome']);

2.4.3 Mengakses View dari Browser

Sekarang, buka http://localhost:8000/welcome di browser. Kamu akan melihat halaman welcome.blade.php dengan pesan yang dikirim dari controller.

Bab 3

Routing dan Controller

3.1 Pengertian Routing

Routing adalah komponen yang mengarahkan permintaan HTTP ke bagian tertentu dari aplikasi, seperti controller atau closure. Dalam Laravel, routing didefinisikan di dalam file routes/web.php untuk aplikasi berbasis web atau routes/api.php untuk API. Routing berperan penting dalam menentukan URL mana yang harus mengarah ke fungsi atau controller tertentu.

3.2 Jenis-jenis Routing

Laravel mendukung beberapa jenis routing yang dapat digunakan sesuai kebutuhan:

1. **Route GET**: Untuk menangani permintaan GET, seperti menampilkan halaman atau data.
2. Route::get('/home', function () {
3. return view('home');
4. });
5. **Route POST**: Untuk menangani permintaan POST, biasanya digunakan saat mengirim data melalui form.
6. Route::post('/submit', function () {
7. // Logika untuk menangani data yang dikirimkan
8. });

3. Route PUT: Digunakan untuk memperbarui data.

1. Route::put('/update/{id}', function ($id) {
2. // Logika untuk memperbarui data dengan ID tertentu
3. });
4. Route DELETE: Digunakan untuk menghapus data.
5. Route::delete('/delete/{id}', function ($id) {
6. // Logika untuk menghapus data dengan ID tertentu
7. });

5. Route dengan Parameter: Mengambil data atau variabel dari URL.

1. Route::get('/user/{id}', function ($id) {
2. return "User ID: " . $id;
3. });

6. Route dengan Parameter Opsional: Parameter bisa bersifat opsional dengan menggunakan tanda ?.

1. Route::get('/post/{id?}', function ($id = null) {
2. return $id ? "Post ID: $id" : "No Post ID";
3. });

3.3 Group Routing

Laravel memungkinkan pengelompokan rute berdasarkan middleware, namespace, atau prefix untuk mengorganisasi routing dengan lebih baik.

**Contoh Grup Routing dengan Prefix:**

1. Route::prefix('admin')->group(function () {
2. Route::get('/dashboard', function () {
3. return "Admin Dashboard";
4. });
5. Route::get('/profile', function () {
6. return "Admin Profile";
7. });
8. });

URL akan menjadi http://localhost:8000/admin/dashboard dan http://localhost:8000/admin/profile.

3.4 Pengertian Controller

**Controller** adalah komponen dalam Laravel yang bertugas menangani logika aplikasi. Controller mengambil data dari model, memprosesnya, dan mengirimkan data ke view untuk ditampilkan. Dengan menggunakan controller, logika aplikasi bisa diorganisir dengan baik dan kode menjadi lebih mudah di-maintain.

3.5 Membuat Controller

Controller dapat dibuat secara otomatis menggunakan perintah artisan.

1. **Membuat Controller Baru**: Jalankan perintah berikut di terminal:

php artisan make:controller ProductController

Perintah ini akan membuat file ProductController.php di direktori app/Http/Controllers/.

1. **Menambahkan Metode di Controller**: Tambahkan beberapa metode di dalam controller untuk menangani permintaan.
2. namespace App\Http\Controllers;
3. use Illuminate\Http\Request;
4. *class* ProductController extends Controller {
5. public function index() {
6. return "Menampilkan daftar produk";
7. }
8. public function show($id) {
9. return "Menampilkan produk dengan ID: " . $id;
10. }
11. public function create() {
12. return "Form untuk membuat produk baru";
13. }
14. public function store(Request $request) {
15. return "Menyimpan produk baru";
16. }
17. public function update($id) {
18. return "Memperbarui produk dengan ID: " . $id;
19. }
20. public function destroy($id) {
21. return "Menghapus produk dengan ID: " . $id;
22. }
23. }

3.6 Menghubungkan Controller dengan Route

Setelah membuat controller dan menambahkan metode, langkah selanjutnya adalah menghubungkannya dengan routing.

1. **Menghubungkan Route ke Controller**:
2. use App\Http\Controllers\ProductController;
3. Route::get('/products', [*ProductController*::*class*, 'index']);
4. Route::get('/products/{id}', [*ProductController*::*class*, 'show']);
5. Route::get('/products/create', [*ProductController*::*class*, 'create']);
6. Route::post('/products', [*ProductController*::*class*, 'store']);
7. Route::put('/products/{id}', [*ProductController*::*class*, 'update']);
8. Route::delete('/products/{id}', [*ProductController*::*class*, 'destroy']);
9. **Route Resource**: Laravel menyediakan cara cepat untuk membuat semua route CRUD dengan satu baris kode menggunakan Route::resource.
10. Route::resource('products', ProductController::class);

Ini secara otomatis menghasilkan semua route standar untuk CRUD:

* GET /products → index
* GET /products/create → create
* POST /products → store
* GET /products/{id} → show
* GET /products/{id}/edit → edit
* PUT/PATCH /products/{id} → update
* DELETE /products/{id} → destroy

3.7 Middleware

Middleware adalah lapisan logika tambahan yang bisa dijalankan sebelum atau sesudah request HTTP. Middleware bisa digunakan untuk berbagai keperluan seperti autentikasi, logging, dll.

Contoh Penggunaan Middleware:

1. use Illuminate\Support\Facades\Route;
2. use App\Http\Controllers\ProductController;
3. Route::middleware(['auth'])->group(function () {
4. Route::get('/products', [*ProductController*::*class*, 'index']);
5. Route::get('/products/{id}', [*ProductController*::*class*, 'show']);
6. });

Middleware ‘auth’ memastikan bahwa hanya pengguna yang telah terautentikasi yang bisa mengakses route tersebut.

3.8 Praktek: Membuat Aplikasi CRUD Sederhana

3.8.1 Membuat Model dan Migration

1. Jalankan perintah berikut untuk membuat model beserta migration untuk tabel products:

php artisan make:model Product -m

Buka file migration yang terletak di database/migrations/ dan tambahkan kolom berikut:

1. public function up() {
2. Schema::create('products', function (Blueprint $table) {
3. $table->id();
4. $table->string('name');
5. $table->text('description')->nullable();
6. $table->decimal('price', 8, 2);
7. $table->timestamps();
8. });
9. }

3. Jalankan migration untuk membuat tabel di database:

php artisan migrate

3.8.2 Menghubungkan Controller dengan Model

Buka ProductController.php dan sesuaikan dengan logika CRUD:

1. use App\Models\Product;
2. use Illuminate\Http\Request;
3. *class* ProductController extends Controller {
4. public function index() {
5. $ $products = Product::all();
6. return view('products.index', compact('products'));
7. }
8. public function create() {
9. return view('products.create');
10. }
11. public function store(Request $request) {
12. $ $product = new Product();
13. $ $product->name = $request->name;
14. $ $product->description = $request->description;
15. $ $product->price = $request->price;
16. $ $product->save();
17. return redirect('/products');
18. }
19. public function show($id) {
20. $ $product = Product::find($id);
21. return view('products.show', compact('product'));
22. }
23. public function edit($id) {
24. $ product = Product::find($id);
25. return view('products.edit', compact('product'));
26. }
27. public function update(Request $request, $id) {
28. $ $product = Product::find($id);
29. $ $product->name = $request->name;
30. $ $product->description = $request->description;
31. $ $product->price = $request->price;
32. $ $product->save();
33. return redirect('/products');
34. }
35. public function destroy($id) {
36. $ $product = Product::find($id);
37. $ $product->delete();
38. return redirect('/products');
39. }
40. }

3.8.3 Membuat View untuk Produk

Buat file view di resources/views/products/ dengan nama index.blade.php, create.blade.php, show.blade.php, dan edit.blade.php untuk menampilkan, menambahkan, mengedit, dan menghapus produk.

Contoh index.blade.php:

1. @extends('layouts.app')
2. @section('content')
3. <h1>Product List</h1>
4. <a href="{{ route('products.create') }}">Create New Product</a>
5. <ul>
6. @foreach($products as $product)
7. <li>
8. <a href="{{ route('products.show', $product->id ) }}"> {{ $product->name }}</a>
9. <form action="{{ route('products.destroy', $product->id) }}" method="POST">
10. @csrf
11. @method('DELETE')
12. <button type="submit">Delete</button>
13. </form>
14. </li>
15. @endforeach
16. </ul>
17. @endsection

Bab 4:

Membuat dan Mengelola Database dengan Laravel

4.1 Pengantar Database di Laravel

Laravel menyediakan alat yang kuat untuk mengelola database, yaitu Eloquent ORM dan Migrasi. Eloquent adalah ORM (Object-Relational Mapping) yang memudahkan interaksi dengan database, sementara migrasi memungkinkan pengelolaan struktur database melalui kode, menjaga konsistensi antar environment (development, staging, production).

4.2 Konfigurasi Database

Sebelum mulai menggunakan database, pastikan koneksi database sudah dikonfigurasi dengan benar.

1. **Konfigurasi di .env**Laravel menyimpan pengaturan database di file .env. Buka file ini dan sesuaikan pengaturan koneksi sesuai dengan database yang digunakan.
2. DB\_CONNECTION=mysql
3. DB\_HOST=127.0.0.1
4. DB\_PORT=3306
5. DB\_DATABASE=laravel\_db
6. DB\_USERNAME=root
7. DB\_PASSWORD=
8. **Konfigurasi di config/database.php**File config/database.php menyimpan konfigurasi database default dan cadangan. Pastikan konfigurasi di file ini sesuai dengan kebutuhan.

4.3 Migrasi (Migration)

Migrasi adalah cara Laravel untuk mengelola skema database. Dengan migrasi, Anda dapat membuat tabel, mengubah struktur tabel, dan melakukan rollback perubahan jika diperlukan.

4.3.1 Membuat Migrasi

1. **Membuat Migrasi Baru**Untuk membuat migrasi baru, gunakan perintah artisan:

php artisan make:migration create\_posts\_table

Perintah ini akan membuat file migrasi di direktori database/migrations/.

1. **Menulis Skema Tabel**  
   Buka file migrasi yang baru saja dibuat dan tambahkan skema tabel:
2. public function up() {
3. Schema::create('posts', function (Blueprint $table) {
4. $ table->id();
5. $ table->string('title');
6. $ table->text('content');
7. $ table->timestamps();
8. });
9. }
10. public function down() {
11. Schema::dropIfExists('posts');
12. }

Metode up() digunakan untuk mendefinisikan skema tabel saat migrasi dijalankan, sedangkan down() digunakan untuk menghapus atau membatalkan tabel ketika migrasi di-rollback.

1. **Menjalankan Migrasi**  
   Jalankan migrasi dengan perintah:

php artisan migrate

Perintah ini akan membuat tabel posts di database.

1. **Rollback Migrasi**  
   Jika Anda perlu mengembalikan perubahan yang dibuat oleh migrasi, jalankan:

php artisan migrate:rollback

4.3.2 Modifikasi Tabel dengan Migrasi

Jika Anda perlu mengubah struktur tabel yang sudah ada, buat migrasi baru dengan perintah:

php artisan make:migration add\_excerpt\_to\_posts\_table --table=posts

Kemudian, tambahkan perubahan pada metode up():

1. public function up() {
2. Schema::table('posts', function (Blueprint $table) {
3. $table->string('excerpt')->nullable();
4. });
5. }
6. public function down() {
7. Schema::table('posts', function (Blueprint $table) {
8. $table->dropColumn('excerpt');
9. });
10. }

Jalankan migrasi untuk menerapkan perubahan:

php artisan migrate

4.4 Eloquent ORM

Eloquent adalah ORM (Object-Relational Mapping) Laravel yang memungkinkan Anda berinteraksi dengan database menggunakan model. Setiap tabel database memiliki model yang sesuai, dan model ini digunakan untuk berinteraksi dengan tabel tersebut.

4.4.1 Membuat Model

Untuk membuat model, jalankan perintah artisan:

php artisan make:model Post

Ini akan membuat file model Post.php di direktori app/Models/.

4.4.2 Menggunakan Eloquent untuk Operasi CRUD

Dengan Eloquent, Anda dapat dengan mudah melakukan operasi CRUD (Create, Read, Update, Delete).

1. **Membuat Data Baru:**
2. $post = new Post();
3. $post->title = 'Post Title';
4. $post->content = 'Post Content';
5. $post->save();
6. **Membaca Data:**
7. $posts = Post::all(); // Mengambil semua data
8. $post = Post::find(1); // Mengambil data berdasarkan ID
9. **Memperbarui Data:**
10. $post = Post::find(1);
11. $post->title = 'Updated Title';
12. $post->save();
13. **MenghapusData:**
14. $post = Post::find(1);
15. $post->delete();

4.4.3 Relasi Antar Model

Eloquent mendukung berbagai jenis relasi antara model seperti one-to-one, one-to-many, many-to-many, dll.

Contoh Relasi One-to-Many (Satu Post memiliki banyak Komentar):

1. **Model Post (Post.php):**
2. public function comments() {
3. return $this->hasMany(Comment::class);
4. }
5. Model Comment (Comment.php):
6. public function post() {
7. return $this->belongsTo(Post::class);
8. }

Dengan relasi ini, Anda bisa mengambil semua komentar dari satu post:

$post = Post::find(1);

$comments = $post->comments;

4.5 Seeder

Seeder digunakan untuk mengisi database dengan data awal atau dummy. Seeder sangat berguna untuk pengujian dan pengembangan.

4.5.1 Membuat Seeder

Untuk membuat seeder, gunakan perintah:

bash

Salin kode

php artisan make:seeder UsersTableSeeder

Seeder ini akan berada di direktori database/seeders/.

4.5.2 Menulis Seeder

Buka file seeder yang baru saja dibuat dan tambahkan logika untuk mengisi database:

1. use Illuminate\Database\Seeder;
2. use Illuminate\Support\Facades\DB;
3. use Illuminate\Support\Facades\Hash;
4. *class* UsersTableSeeder extends Seeder {
5. public function run() {
6. DB::table('users')->insert([
7. 'name' => 'John Doe',
8. 'email' => 'johndoe@example.com',
9. 'password' => *Hash*::*make*('password'),
10. ]);
11. }
12. }

4.5.3 Menjalankan Seeder

Untuk menjalankan seeder, gunakan perintah:

bash

Salin kode

php artisan db:seed --class=UsersTableSeeder

Atau, untuk menjalankan semua seeder yang ada:

bash

Salin kode

php artisan db:seed

4.6 Fakta dan Pengujian Menggunakan Factory

Selain seeder, Factory digunakan untuk membuat data dummy secara otomatis dengan cara yang lebih dinamis.

4.6.1 Membuat Factory

Buat factory dengan perintah:

bash

php artisan make:factory UserFactory --model=User

php artisan make:factory UserFactory --model=User

Factory ini akan terletak di database/factories/.

4.6.2 Menggunakan Factory

Di dalam UserFactory.php, definisikan bagaimana data dummy dibuat:

php

Salin kode

1. use Illuminate\Database\Eloquent\Factories\Factory;
2. *class* UserFactory extends Factory {
3. protected $model = User::class;
4. public function definition() {
5. return [
6. 'name' => $this->*faker*->*name*,
7. 'email' => $this->*faker*->*unique*()->*safeEmail*,
8. 'password' => *Hash*::*make*('password'),
9. ];
10. }
11. }

4.6.3 Menggunakan Factory di Seeder

Gunakan factory di dalam seeder untuk membuat data dummy:

php

Salin kode

1. use App\Models\User;
2. *class* UsersTableSeeder extends Seeder {
3. public function run() {
4. User::factory()->count(10)->create();
5. }
6. }

Bab 5:

Blade Templating

5.1 Pengantar Blade

Blade adalah mesin templating yang disediakan oleh Laravel. Blade memungkinkan Anda untuk menulis HTML yang dapat menyisipkan logika PHP dengan cara yang sederhana dan efisien. Blade memberikan kemudahan dalam membuat tampilan yang dinamis dan terstruktur, serta mendukung fitur seperti layout, komponen, dan bagian (sections).

5.2 Sintaks Dasar Blade

Blade menggunakan sintaks khusus yang mudah dipahami dan ringan. Berikut adalah beberapa sintaks dasar Blade:

1. **Output Data**:

{{ $variable }}

Blade secara otomatis akan melakukan escaping terhadap data yang di-output untuk mencegah serangan XSS (Cross-Site Scripting). Jika Anda ingin menampilkan data tanpa escaping, gunakan:

{!! $variable !!}

1. **Komentar**: Komentar dalam Blade tidak akan tampil di HTML output.

{{-- Ini adalah komentar Blade --}}

1. **Directives**: Blade menggunakan "directives" untuk mengimplementasikan logika seperti kondisi, loop, dan lain-lain.

* Kondisi If :

1. @if ($user->isAdmin())
2. <p>Welcome, Admin.</p>
3. @elseif ($user->isModerator())
4. <p>Welcome, Moderator.</p>
5. @else
6. <p>Welcome, User.</p>
7. @endif

* Looping :

1. @foreach ($users as $user)
2. <p>This is user {{ $user->name }}</p>
3. @endforeach

* Perulangan While:

1. @while (true)
2. <p>This will keep running.</p>
3. @endwhile

* Perulangan While:

1. @switch($user->role)
2. @case('admin')
3. <p>Admin User</p>
4. @break
5. @case('moderator')
6. <p>Moderator User</p>
7. @break
8. @default
9. <p>Guest User</p>
10. @endswitch

5.3 Layout dan Penggunaan Template

Blade memudahkan penggunaan template dengan konsep layout dan sections. Dengan menggunakan layout, Anda dapat membuat satu template dasar yang dapat digunakan oleh banyak halaman.

5.3.1 Membuat Layout Dasar

Pertama, buat layout dasar di resources/views/layouts/app.blade.php:

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <meta name="viewport" content="width=device-width, initial-scale=1.0">
6. <title>@yield('title', 'Laravel App')</title>
7. <link rel="stylesheet" href="{{ asset('css/app.css') }}">
8. </head>
9. <body>
10. <header>
11. <h1>My Laravel App</h1>
12. @include('partials.nav')
13. </header>
14. <div class="container">
15. @yield('content')
16. </div>
17. <footer>
18. <p>&copy; 2024 My Laravel App</p>
19. </footer>
20. </body>
21. </html>

@yield('title'): Menyediakan tempat untuk judul halaman yang bisa diisi oleh template yang menggunakan layout ini.

@yield('content'): Tempat untuk konten utama halaman.

5.3.2 Menggunakan Layout di View

Sekarang, buat file view yang menggunakan layout ini. Misalnya, buat file resources/views/home.blade.php:

1. @extends('layouts.app')
2. @section('title', 'Home Page')
3. @section('content')
4. <h2>Welcome to My Laravel App</h2>
5. <p>This is the home page.</p>
6. @endsection

@extends('layouts.app'): Menunjukkan bahwa view ini menggunakan layout app.

@section('title', 'Home Page'): Mengisi placeholder title di layout dengan "Home Page".

@section('content'): Mengisi placeholder content di layout dengan konten spesifik halaman ini.

5.3.3 Membuat Partial Views

Partial views adalah bagian dari tampilan yang dapat dipisah dan digunakan kembali. Contoh umum adalah menu navigasi atau footer.

1. Buat partial view untuk navigasi di resources/views/partials/nav.blade.php:
2. <nav>
3. <ul>
4. <li><a href="{{ url('/') }}">Home</a></li>
5. <li><a href="{{ url('/about') }}">About</a></li>
6. <li><a href="{{ url('/contact') }}">Contact</a></li>
7. </ul>
8. </nav>

2. Gunakan partial view ini di layout atau view lain dengan @include:

@include('partials.nav')

5.4 Blade Components dan Slots

Components di Blade adalah cara untuk membuat UI yang dapat digunakan kembali, mirip dengan partials, tetapi lebih fleksibel dan mendukung slot.

5.4.1 Membuat Component

Buat component untuk kartu (card) di resources/views/components/card.blade.php :

<div class="card">

<div class="card-header">

{{ $header }}

</div>

<div class="card-body">

{{ $slot }}

</div>

</div>

{{ $slot }}: Digunakan untuk menempatkan konten utama dari component.

{{ $header }}: Slot khusus yang dapat diisi dengan konten tertentu.

5.4.2 Menggunakan Component

Sekarang gunakan component ini di view lain:

<x-card>

<x-slot name="header">

Card Title

</x-slot>

This is the content of the card.

</x-card>

<x-card>: Memanggil component card.

<x-slot name="header">: Mengisi slot header dengan "Card Title".

This is the content of the card.: Mengisi slot utama (slot) dengan konten.

5.5 Directive Khusus Blade

Blade juga menyediakan beberapa directive khusus yang berguna dalam pengembangan aplikasi Laravel:

1. **@csrf**: Directive untuk menyertakan token CSRF (Cross-Site Request Forgery) di form.
2. <form method="POST" action="/submit">
3. @csrf
4. <!-- Form fields -->
5. </form>
6. **@method**: Directive untuk menyertakan metode HTTP seperti PUT atau DELETE dalam form.
7. <form method="POST" action="/update">
8. @csrf
9. @method('PUT')
10. <!-- Form fields -->
11. </form>
12. **@auth** & **@guest**: Directive untuk menampilkan konten berdasarkan status autentikasi pengguna.
13. @auth
14. <p>Welcome, {{ Auth::user()->name }}!</p>
15. @endauth
16. @guest
17. <p>Please login to continue.</p>
18. @endguest

**@foreach**, **@for**, **@if**: Directive untuk logika looping dan kondisi yang telah dibahas sebelumnya.

5.6 Blade Templating untuk Pengembangan Lanjutan

Blade mendukung berbagai fitur lain untuk pengembangan aplikasi yang lebih kompleks, seperti:

Blade Include Parameters:

@include('partials.nav', ['items' => $menuItems])

Component dan Inline Component: Laravel juga mendukung inline components, yang dapat dibuat dan didefinisikan langsung dalam satu file view tanpa perlu membuat file terpisah.

Blade Alias: Anda dapat menggunakan alias untuk component agar penulisannya lebih ringkas.

5.7 Praktek: Membuat Halaman Dinamis

Mari buat halaman dinamis sederhana menggunakan Blade:

1. Buat Layout Dasar di resources/views/layouts/app.blade.php (seperti yang dijelaskan di atas).
2. Buat Partial View untuk navigasi di resources/views/partials/nav.blade.php.
3. Buat Halaman Home di resources/views/home.blade.php:
4. @extends('layouts.app')
5. @section('title', 'Home Page')
6. @section('content')
7. <h2>Welcome to My Laravel App</h2>
8. <p>This is the home page.</p>
9. @endsection

Tambahkan Route di routes/web.php:

1. Route::get('/', function () {
2. return view('home');
3. });

Jalankan Aplikasi dan buka http://localhost:8000/ untuk melihat hasilnya.

Bab 6:

CRUD dengan Laravel

6.1 Pengantar CRUD

CRUD adalah singkatan dari Create, Read, Update, Delete yang merupakan operasi dasar dalam pengelolaan data. Dalam Laravel, CRUD dapat diimplementasikan dengan mudah menggunakan Eloquent ORM dan fitur-fitur lain seperti routing, controller, dan Blade templating.

Pada bab ini, kita akan membuat aplikasi sederhana untuk mengelola data post (artikel) dengan fitur CRUD.

6.2 Membuat Model dan Migrasi

6.2.1 Membuat Model dan Migrasi

Langkah pertama adalah membuat model dan migrasi untuk entitas Post. Jalankan perintah berikut:

php artisan make:model Post -m

Perintah ini akan membuat model Post di app/Models/Post.php dan file migrasi di database/migrations/.

6.2.2 Menulis Skema Tabel di Migrasi

Buka file migrasi yang baru saja dibuat di database/migrations/ dan tuliskan skema tabel untuk posts:

1. public function up()
2. {
3. Schema::create('posts', function (Blueprint $table) {
4. $table->id();
5. $table->string('title');
6. $table->text('content');
7. $table->timestamps();
8. });
9. }
10. public function down()
11. {
12. Schema::dropIfExists('posts');
13. }

Jalankan migrasi untuk membuat tabel di database:

php artisan migrate

6.3 Membuat Controller dan Routing

6.3.1 Membuat Controller

Laravel menyediakan cara cepat untuk membuat controller dengan resource method menggunakan perintah berikut:

php artisan make:controller PostController --resource

Perintah ini akan membuat PostController di app/Http/Controllers/PostController.php dengan metode CRUD bawaan seperti index, create, store, show, edit, update, dan destroy.

6.3.2 Menambahkan Routing

Tambahkan route resource untuk PostController di routes/web.php:

use App\Http\Controllers\PostController;

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

Route resource ini secara otomatis akan menghasilkan route untuk operasi CRUD:

* GET /posts – Menampilkan daftar semua post (index).
* GET /posts/create – Menampilkan form untuk membuat post baru (create).
* POST /posts – Menyimpan post baru (store).
* GET /posts/{post} – Menampilkan post tertentu (show).
* GET /posts/{post}/edit – Menampilkan form untuk mengedit post (edit).
* PUT/PATCH /posts/{post} – Memperbarui post yang ada (update).
* DELETE /posts/{post} – Menghapus post (destroy).

6.4 Implementasi CRUD

6.4.1 Membuat Post Baru (Create)

Formulir untuk Membuat Post Baru

Buat file view resources/views/posts/create.blade.php untuk menampilkan form pembuatan post:

1. @extends('layouts.app')
2. @section('content')
3. <h1>Create New Post</h1>
4. <form action="{{ route('posts.store') }}" method="POST">
5. @csrf
6. <div class="form-group">
7. <label for="title">Title:</label>
8. <input type="text" class="form-control" id="title" name="title" required>
9. </div>
10. <div class="form-group">
11. <label for="content">Content:</label>
12. <textarea class="form-control" id="content" name="content" rows="5" required></textarea>
13. </div>
14. <button type="submit" class="btn btn-primary">Create Post</button>
15. </form>
16. @endsection

Menyimpan Post Baru

Tambahkan logika untuk menyimpan post baru di PostController:

1. public function store(Request $request)
2. {
3. $request->validate([
4. 'title' => 'required|max:255',
5. 'content' => 'required',
6. ]);
7. Post::create([
8. 'title' => $request->*title*,
9. 'content' => $request->*content*,
10. ]);
11. return redirect()->route('posts.index')->with('success', 'Post created successfully.');
12. }

6.4.2 Membaca dan Menampilkan Post (Read)

Menampilkan Semua Post

Buat file view resources/views/posts/index.blade.php untuk menampilkan daftar semua post:

1. @extends('layouts.app')
2. @section('content')
3. <h1>All Posts</h1>
4. @if (session('success'))
5. <div class="alert alert-success">
6. {{ session('success') }}
7. </div>
8. @endif
9. <a href="{{ route('posts.create') }}" class="btn btn-primary mb-3">Create New Post</a>
10. @if($posts->count())
11. <ul class="list-group">
12. @foreach($posts as $post)
13. <li class="list-group-item">
14. <a href="{{ route('posts.show', $post->id) }}">{{ $post->title }}</a>
15. </li>
16. @endforeach
17. </ul>
18. @else
19. <p>No posts found.</p>
20. @endif
21. @endsection
22. Tambahkan logika untuk mengambil dan mengirimkan semua post ke view di PostController:
23. public function index()
24. {
25. $posts = Post::all();
26. return view('posts.index', compact('posts'));
27. }
28. Menampilkan Detail Post

Buat file view resources/views/posts/show.blade.php untuk menampilkan detail post tertentu:

1. @extends('layouts.app')
2. @section('content')
3. <h1>{{ $post->title }}</h1>
4. <p>{{ $post->content }}</p>
5. <a href="{{ route('posts.edit', $post->id) }}" class="btn btn-secondary">Edit</a>
6. <form action="{{ route('posts.destroy', $post->id) }}" method="POST" style="display:inline;">
7. @csrf
8. @method('DELETE')
9. <button type="submit" class="btn btn-danger">Delete</button>
10. </form>
11. <a href="{{ route('posts.index') }}" class="btn btn-primary">Back to All Posts</a>
12. @endsection

Tambahkan logika untuk mengambil dan menampilkan post tertentu di PostController:

1. public function show($id)
2. {
3. $post = Post::findOrFail($id);
4. return view('posts.show', compact('post'));
5. }

6.4.3 Mengedit dan Memperbarui Post (Update)

Formulir untuk Mengedit Post

Buat file view resources/views/posts/edit.blade.php untuk menampilkan form pengeditan post:

1. @extends('layouts.app')
2. @section('content')
3. <h1>Edit Post</h1>
4. <form action="{{ route('posts.update', $post->id) }}" method="POST">
5. @csrf
6. @method('PUT')
7. <div class="form-group">
8. <label for="title">Title:</label>
9. <input type="text" class="form-control" id="title" name="title" value="{{ $post->title }}" required>
10. </div>
11. <div class="form-group">
12. <label for="content">Content:</label>
13. <textarea class="form-control" id="content" name="content" rows="5" required>{{ $post->content }}</textarea>
14. </div>
15. <button type="submit" class="btn btn-primary">Update Post</button>
16. </form>
17. @endsection

Tambahkan logika untuk menampilkan form pengeditan dan memperbarui post di PostController:

1. public function edit($id)
2. {
3. $post = Post::findOrFail($id);
4. return view('posts.edit', compact('post'));
5. }
6. public function update(Request $request, $id)
7. {
8. $request->validate([
9. 'title' => 'required|max:255',
10. 'content' => 'required',
11. ]);
12. $post = Post::findOrFail($id);
13. $post->update([
14. 'title' => $request->*title*,
15. 'content' => $request->*content*,
16. ]);
17. return redirect()->route('posts.index')->with('success', 'Post updated successfully.');
18. }

6.4.4 Menghapus Post (Delete)

Tambahkan logika untuk menghapus post di PostController:

1. public function destroy($id)
2. {
3. $post = Post::findOrFail($id);
4. $post->delete();
5. return redirect()->route('posts.index')->with('success', 'Post deleted successfully.');
6. }

6.5 Validasi Data

Laravel menyediakan fitur validasi yang kuat untuk memastikan data yang diterima oleh aplikasi sesuai dengan aturan yang telah ditentukan. Anda telah melihat contoh validasi sederhana di atas dalam metode store dan update:

1. $request->validate([
2. 'title' => 'required|max:255',
3. 'content' => 'required',
4. ]);

Validasi ini memastikan bahwa title dan content harus diisi dan title tidak lebih dari 255 karakter.

6.6 Menyempurnakan UI dengan Blade

Untuk menyempurnakan antarmuka pengguna, Anda bisa memanfaatkan Blade untuk mengelola layout dan component seperti yang telah dibahas pada bab sebelumnya.

6.7 Tugas: Membuat CRUD Lengkap

Mari kita gabungkan semua langkah di atas untuk membuat aplikasi CRUD sederhana. Berikut adalah langkah-langkah utamanya:

* Buat Model dan Migrasi untuk posts.
* Buat Controller PostController dengan Resource Method.
* Definisikan Route Resource di routes/web.php.
* Implementasikan Metode CRUD di PostController.
* Buat View untuk Menampilkan, Membuat, Mengedit, dan Menghapus Post.
* Jalankan aplikasi dan kunjungi rute /posts untuk melihat dan mengelola post.

Bab 7:

Relasi Antar Tabel

7.1 Pengantar Relasi Antar Tabel

Dalam database relasional, relasi antar tabel digunakan untuk menghubungkan data yang terkait dalam tabel yang berbeda. Laravel menyediakan dukungan yang kuat untuk mengelola relasi ini menggunakan Eloquent ORM. Dengan Eloquent, Anda dapat dengan mudah mendefinisikan dan mengelola berbagai jenis relasi seperti One-to-One, One-to-Many, Many-to-Many, dan Has Many Through.

Pada bab ini, kita akan membahas cara mengimplementasikan relasi-relasi ini dalam Laravel dan bagaimana memanfaatkannya dalam pengembangan aplikasi.

7.2 Relasi One-to-One

Relasi One-to-One menghubungkan satu baris dalam tabel dengan satu baris dalam tabel lainnya. Sebagai contoh, anggap kita memiliki dua tabel: users dan profiles. Setiap user memiliki satu profil, dan setiap profil hanya dimiliki oleh satu user.

7.2.1 Membuat Migrasi dan Model

Buat migrasi dan model untuk tabel profiles:

php artisan make:model Profile -m

Kemudian, tambahkan kolom user\_id di tabel profiles untuk menyimpan relasi ke tabel users:

1. public function up()
2. {
3. Schema::create('profiles', function (Blueprint $table) {
4. $table->id();
5. $table->unsignedBigInteger('user\_id')->unique();
6. $table->text('bio')->nullable();
7. $table->string('twitter')->nullable();
8. $table->timestamps();
9. $table->foreign('user\_id')->references('id')->on('users')->onDelete('cascade');
10. });
11. }

Jalankan migrasi:

php artisan migrate

7.2.2 Mendefinisikan Relasi di Model

Tambahkan relasi One-to-One di model User dan Profile:

Model User (app/Models/User.php):

1. public function profile()
2. {
3. return $this->hasOne(Profile::class);
4. }

Model Profile (app/Models/Profile.php):

1. public function user()
2. {
3. return $this->belongsTo(User::class);
4. }

7.2.3 Menggunakan Relasi One-to-One

Untuk mengakses data profil pengguna:

1. $user = User::find(1);
2. $profile = $user->profile;

Untuk mengakses data pengguna dari profil:

1. $profile = Profile::find(1);
2. $user = $profile->user;

7.3 Relasi One-to-Many

Relasi One-to-Many menghubungkan satu baris dalam tabel dengan banyak baris dalam tabel lain. Sebagai contoh, satu post dapat memiliki banyak comments.

7.3.1 Membuat Migrasi dan Model

Buat migrasi dan model untuk tabel comments:

php artisan make:model Comment -m

Tambahkan kolom post\_id di tabel comments:

1. public function up()
2. {
3. Schema::create('comments', function (Blueprint $table) {
4. $table->id();
5. $table->unsignedBigInteger('post\_id');
6. $table->string('author');
7. $table->text('content');
8. $table->timestamps();
9. $table->foreign('post\_id')->references('id')->on('posts')->onDelete('cascade');
10. });
11. }

Jalankan migrasi:

php artisan migrate

7.3.2 Mendefinisikan Relasi di Model

Tambahkan relasi One-to-Many di model Post dan Comment:

Model Post (app/Models/Post.php):

1. public function comments()
2. {
3. return $this->hasMany(Comment::class);
4. }

Model Comment (app/Models/Comment.php):

7.3.3 Menggunakan Relasi One-to-Many

Untuk mengakses komentar dari sebuah post:

1. $post = Post::find(1);
2. $comments = $post->comments;

Untuk mengakses post dari sebuah komentar:

1. $comment = Comment::find(1);
2. $post = $comment->post;

7.4 Relasi Many-to-Many

Relasi Many-to-Many menghubungkan banyak baris dalam satu tabel dengan banyak baris dalam tabel lain. Contohnya adalah hubungan antara posts dan tags, di mana satu post bisa memiliki banyak tag, dan satu tag bisa dimiliki oleh banyak post.

7.4.1 Membuat Tabel Pivot

Relasi Many-to-Many membutuhkan tabel pivot untuk menyimpan relasi antara dua tabel. Buat migrasi untuk tabel tags dan tabel pivot post\_tag:

php artisan make:model Tag -m

Edit migrasi tags dan buat migrasi baru untuk tabel pivot post\_tag:

Migrasi Tags:

1. public function up()
2. {
3. Schema::create('tags', function (Blueprint $table) {
4. $table->id();
5. $table->string('name')->unique();
6. $table->timestamps();
7. });
8. }

Migrasi Tabel Pivot Post\_Tag:

1. public function up()
2. {
3. Schema::create('post\_tag', function (Blueprint $table) {
4. $table->id();
5. $table->unsignedBigInteger('post\_id');
6. $table->unsignedBigInteger('tag\_id');
7. $table->timestamps();
8. $table->foreign('post\_id')->references('id')->on('posts')->onDelete('cascade');
9. $table->foreign('tag\_id')->references('id')->on('tags')->onDelete('cascade');
10. });
11. }

Jalankan migrasi:

php artisan migrate

7.4.2 Mendefinisikan Relasi di Model

Tambahkan relasi Many-to-Many di model Post dan Tag:

Model Post (app/Models/Post.php):

1. public function tags()
2. {
3. return $this->belongsToMany(Tag::class);
4. }

Model Tag (app/Models/Tag.php):

1. public function posts()
2. {
3. return $this->belongsToMany(Post::class);
4. }

7.4.3 Menggunakan Relasi Many-to-Many

Untuk menambahkan tag ke sebuah post:

1. $post = Post::find(1);
2. $tag = Tag::find(1);
3. $post->tags()->attach($tag);

Untuk mengakses semua tag yang terkait dengan sebuah post:

1. $post = Post::find(1);
2. $tags = $post->tags;

Untuk mengakses semua post yang terkait dengan sebuah tag:

1. $tag = Tag::find(1);
2. $posts = $tag->posts;

7.5 Relasi Has Many Through

Relasi Has Many Through memungkinkan Anda untuk mengakses data melalui relasi perantara. Sebagai contoh, anggap kita memiliki tiga tabel: countries, users, dan posts. Jika kita ingin mendapatkan semua post dari suatu negara, kita bisa menggunakan relasi hasManyThrough.

7.5.1 Mendefinisikan Relasi di Model

Tambahkan relasi hasManyThrough di model Country:

Model Country (app/Models/Country.php):

public function posts()

{

return $this->hasManyThrough(Post::class, User::class);

}

7.5.2 Menggunakan Relasi Has Many Through

Untuk mengakses semua post dari suatu negara:

1. $country = Country::find(1);
2. $posts = $country->posts;

7.6 Relasi Polimorfik

Laravel juga mendukung Relasi Polimorfik yang memungkinkan model untuk memiliki relasi ke beberapa model lainnya melalui satu model. Contoh relasi polimorfik adalah ketika comments dapat dimiliki oleh posts dan videos.

7.6.1 Membuat Relasi Polimorfik

Buat migrasi untuk tabel comments:

Tambahkan kolom commentable\_id dan commentable\_type untuk relasi polimorfik:

php artisan make:migration create\_comments\_table --create=comments

Tambahkan relasi di model Comment:

1. public function commentable()
2. {
3. return $this->morphTo();
4. }

Tambahkan relasi di model Post dan Video:

Model Post (app/Models/Post.php):

public function comments()

1. {
2. return $this->morphMany(Comment::class, 'commentable');
3. }