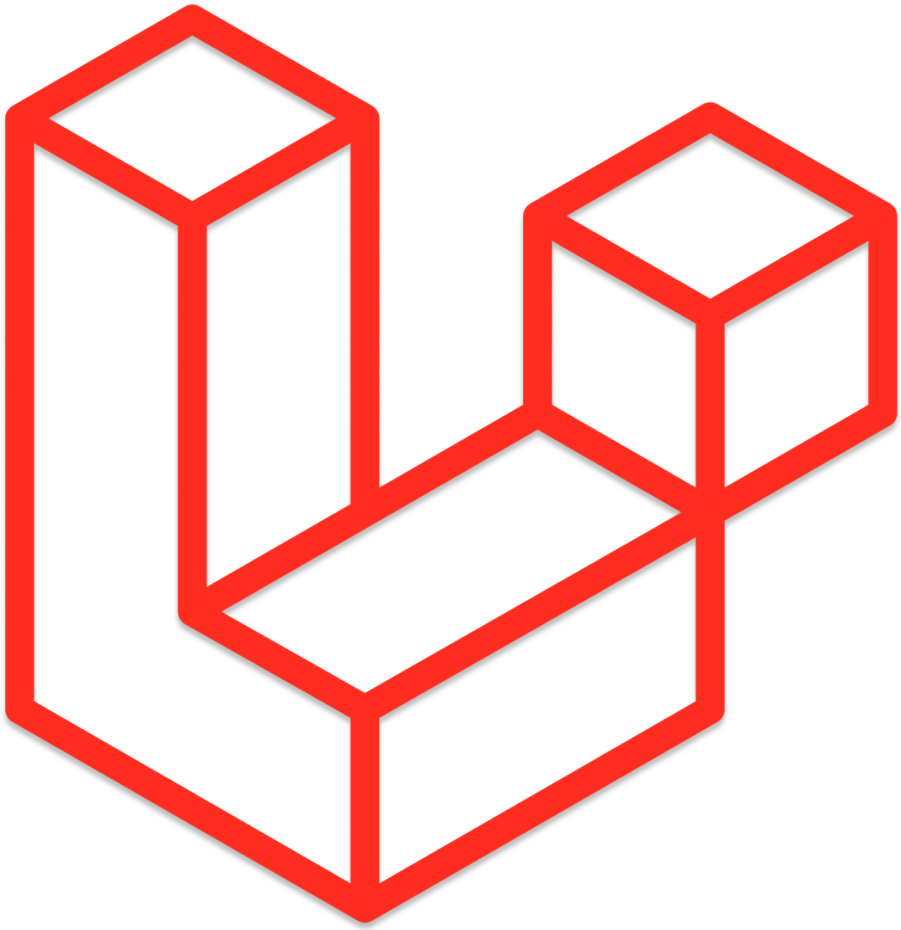
Buku Saku Laravel



Muhammad Amirul Ihsan

www.kawankoding.com

Daftar Isi

Menginstal Laravel

Untuk memulai proyek dalam buku ini ada beberapa hal yang perlu disiapkan untuk bisa menginstal Laravel. Salah satunya kebutuhan server yang diminta oleh Laravel harus dipenuhi begitu juga alat alat pendukungnya.

Kebutuhan Server

Untuk menginstal Laravel kita diharuskan untuk memenuhi kebutuhan server dari Laravel. Kali ini kita akan menggunakan Laravel 8 dan harus memenuhi kebutuhan seperti yang ada di bawah ini:

- PHP >= 7.4
- BCMath PHP Extension
- Ctype PHP Extension
- Fileinfo PHP Extension
- JSON PHP Extension
- Mbstring PHP Extension
- OpenSSL PHP Extension
- PDO PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension

Menginstal Laravel

Laravel menggunakan Composer untuk mengatur ketergantungan (*dependencies*). Jadi sebelum menginstal Laravel, pastikan kita sudah menginstal Composer di laptop / komputer kita.

Menggunakan Laravel Installer

Cara pertama kita bisa gunakan Laravel Installer yang bisa diunduh menggunakan Composer:

```
composer global require laravel/installer
```

Kemudian, pastikan direktori vendor bin dari Composer pada $PATH kawan kawan, jadi file executable Laravel bisa dijalankan. Adapun lokasi direktori ini berbeda berdasarkan sistem operasi.

- macOS: `$HOME/.composer/vendor/bin`
- Windows: `%USERPROFILE%\AppData\Roaming\Composer\vendor\bin`
- GNU / Linux Distributions: `$HOME/.config/composer/vendor/bin or $HOME/.composer/vendor/bin`

Kawan-kawan juga bisa menjalankan composer global about untuk mengetahui path global dari instalasi Composer.

Jika sudah terinstal, perintah laravel new akan memnbuat instalasi Laravel di direktori yang kita tentukan. Contohnya `laravel new crud` akan membuat sebuah direktori dengan nama crud berisi proyek Laravel dan semua dependencynya.

Menggunakan Composer create-project

Jika tidak ingin menginstal Laravel Installer, kita bisa menggunakan perintah composer create-project untuk membuat proyek Laravel.

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

Server development lokal

Setelah berhasil menginstal Laravel, jika kawan-kawan sudah menginstal PHP di laptop / komputer dan terbiasa menggunakan PHP Built in Server untuk menjalankan aplikasi, kawan bisa gunakan perintah artisan serve. Perintah tersebut akan menjalankan server development lokal pada [%base_url].

```
php artisan serve
```

Setelah instal dan menjalankan aplikasi pada server lokal, berikutnya kita akan membahas tentang konfigurasi pada proyek Laravel.

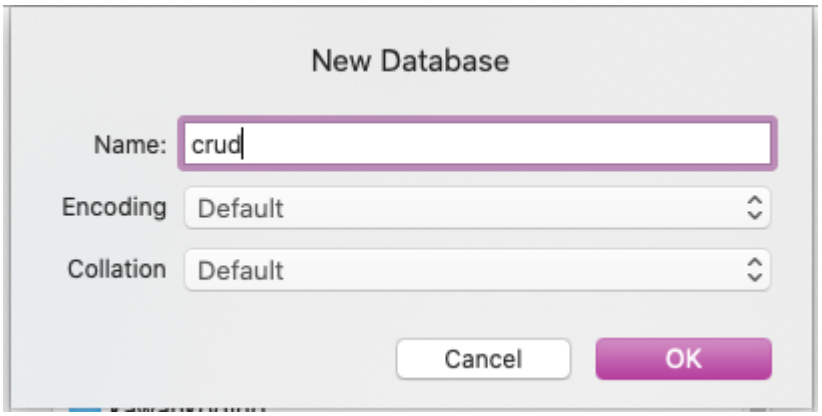
Konfigurasi Proyek Laravel

Untuk konfigurasi umum pada aplikasi Laravel bisa di lakukan pada file .env yang berada dalam direktori utama. Hal pertama yang perlu kita sesuaikan konfigurasinya tentu adalah *database*.

Kita buat dulu databasenya, dalam tulisan ini saya menggunakan Table Plus sebagai aplikasi untuk manajemen databasenya, kawan-kawan bebas pakai apapun, ***PHPMyAdmin***, ***HeidiSQL***, ***MySQL Workbench***, ***Adminer***, dll.

Membuat Database

Langkah awal mari kita buat database dengan nama `crud`.



Setelah selesai membuat database, sekarang kita masuk kembali ke proyek Laravel untuk melakukan konfigurasi. Sejak Laravel 8 untuk konfigurasi nama database biasanya secara otomatis akan mengikuti nama dari proyeknya. Jadi ketika kita buat proyek dengan nama `crud` secara otomatis konfigurasi Laravel akan memberi nama database `crud` pada file `.env`.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=crud
DB_USERNAME=root
DB_PASSWORD=
```

Jika sudah sesuai dengan nama database yang dibuat, kita bisa lanjutkan ke langkah selanjutnya.

Membuat Tabel

Untuk membuat tabel kita tidak akan menggunakan aplikasi manajemen database atau dari perintah **CLI** (*Command Line Interface*).

Di Laravel ada fitur yang disebut dengan ***migration*** dengan fitur ini kita bisa membuat tabel database dengan menggunakan kode PHP.

Sebagai awalan proyek ada file migrations yang disediakan Laravel. Semua file migrations akan berada dalam direktori `database/migrations` sebagai contoh mari kita buka file migration `users` dari proyek Laravel.

```php
<?php

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

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

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

Pada kode di atas method `up()` adalah method yang digunakan untuk membuat database, di dalamnya ada berbagai macam method untuk mendefinisikan tipe data dari kolom yang akan dibuat, untuk lebih lengkapnya cek dokumentasinya di **sini**.

Sedangkan method `down()` adalah fungsi untuk rollback atau untuk membatalkan migration yang pernah dijalankan sebanyak 1 batch.

Untuk bisa membuat tabelnya kita perlu menjalankan migrationnya dengan perintah `php artisan migrate` dan Laravel akan menjalankan migration.

```
➜  crud  php artisan migrate
Migration table created successfully.
    Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table (18.89ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table (12.26ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated:  2019_08_19_000000_create_failed_jobs_table (11.53ms)
➜  crud
```

Jika menemui tampilan yang kurang lebih seperti gambar di atas dan tidak menemui error, berarti migration sukses dijalankan.

Untuk mengecek tabelnya benar sudah terbuat langsung cek di database kita.

name	schema	kind	charset	collation	engine	estimated_row
failed_jobs	crud	TABLE	utf8mb4	utf8mb4_…	InnoDB	0
migrations	crud	TABLE	utf8mb4	utf8mb4_…	InnoDB	0
password_resets	crud	TABLE	utf8mb4	utf8mb4_…	InnoDB	0
users	crud	TABLE	utf8mb4	utf8mb4_…	InnoDB	0

Membuat Migration

Setelah kita pahami cara membuat tabel pada Laravel, sekarang mari kita buat migration pertama kita.

Untuk membuat file migration kita gunakan perintah Artisan dengan perintah `php artisan make:migration nama_migration`, kita akan buat tabel dengan nama `posts` maka bisa kita jalankan perintah `php artisan make:migration create_posts_table`.

File yang dibuat dari perintah tersebut kurang lebih seperti ini.

```
<?php

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

class CreatePostsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
        });
    }

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

Pada tabel `posts` kita membutuhkan kolom `user_id` untuk menentukan relasi ke tabel

`users`, `title` untuk judul `body` untuk isi konten dari postingan, `cover` untuk gambar sampul postingan yang akan kita buat opsional dan akan menggunakan tipe data `string` karena hanya menyimpan *path file* saja. Pada method `up()` kita update isinya jadi seperti ini.

```
Schema::create('posts', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id');
    $table->string('title');
    $table->text('body');
    $table->string('image')->nullable()->default(null);
    $table->timestamps();
});
```

Setelah selesai dengan file migrationnya, jalankan `php artisan migrate` agar tabel terbuat ke dalam *database* dan pastikan untuk cek ke database dan terdapata tabel `post` dengan struktur seperti ini.

#	column_name	data_type	character_set	collation	is_nullable	column_default	extra	foreign_key	comment
1	id	bigint unsigned	NULL	NULL	NO	NULL	auto_increment	EMPTY	EMPTY
2	user_id	bigint unsigned	NULL	NULL	NO	NULL	EMPTY	EMPTY	EMPTY
3	title	varchar(255)	utf8mb4	utf8mb4_unicode_ci	NO	NULL	EMPTY	EMPTY	EMPTY
4	body	text	utf8mb4	utf8mb4_unicode_ci	NO	NULL	EMPTY	EMPTY	EMPTY
5	image	varchar(255)	utf8mb4	utf8mb4_unicode_ci	YES	NULL	EMPTY	EMPTY	EMPTY
6	created_at	timestamp	NULL	NULL	YES	NULL	EMPTY	EMPTY	EMPTY
7	updated_at	timestamp	NULL	NULL	YES	NULL	EMPTY	EMPTY	EMPTY

Membuat Fitur Autentikasi

Setelah selesai dengan tabel yang kita butuhkan, selanjutnya kita mulai untuk menulis kode untuk aplikasi kita. Mulai dari sistem autentikasi dulu yang jadi kebutuhan mayoritas aplikasi.

Di Laravel kita bisa dengan mudah untuk menyiapkan fitur ini sejak Laravel versi 5.2 dengan perintah `artisan make:auth`, dengan *package* `laravel/ui` sejak Laravel 6 - 7, dan di Laravel 8 kita dikenalkan `laravel/jetstream` yang membawa banyak sekali fitur & menuai banyak kontroversi. Hingga yang paling baru `laravel/breeze` yang dibuat untuk menjawab kebutuhan developer yang tidak membutuhkan semua fitur jetstream, Laravel Breeze ini lebih mirip dengan Laravel UI tapi dengan Tailwindcss sebagai *css framework*nya.

Kita akan gunakan Laravel Breeze untuk kali ini, untuk menginstall Laravel Breeze gunakan

perintah ini:

```
composer require laravel/breeze
```

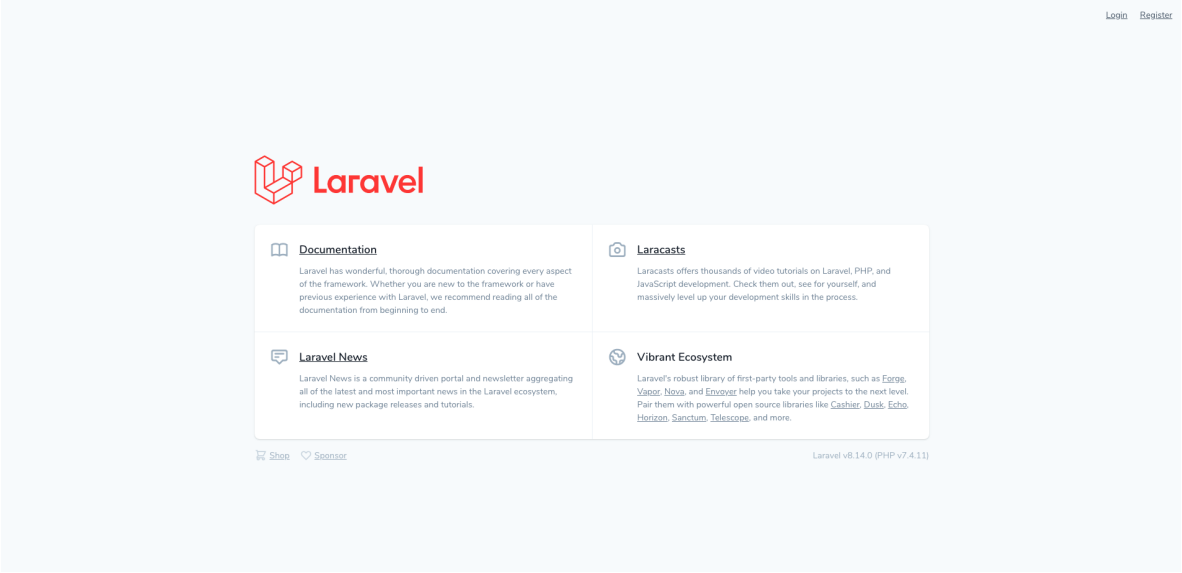
Setelah selesai mengintall Laravel Breeze, langkah selanjutnya jalankan perintah :

```
php artisan breeze:install
```

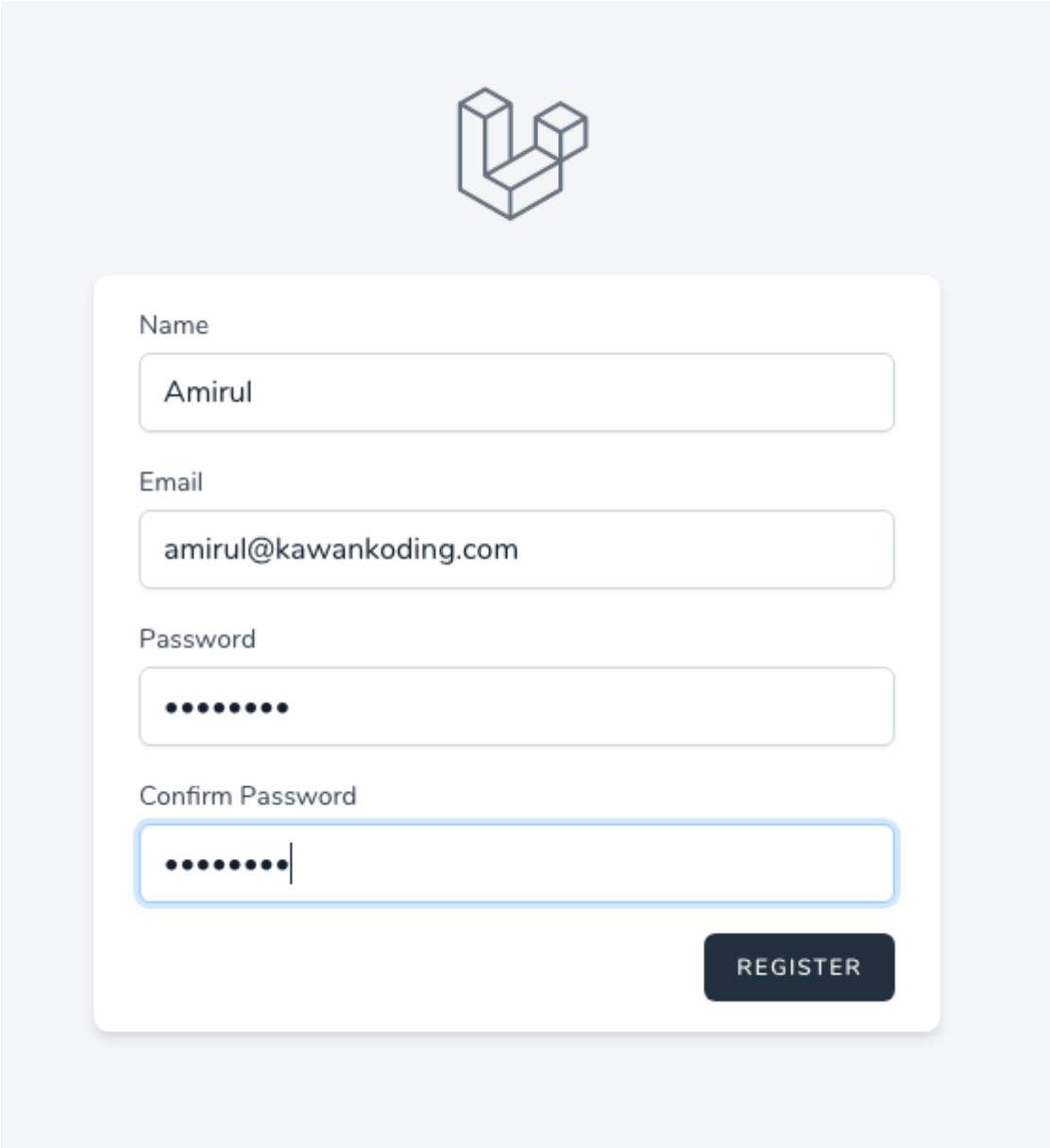
Dan karena untuk aset-asetnya belum secara out of the box disertakan, kita perlu *install* & *build* asetnya terlebih dulu.

```
npm install && npm run dev
```

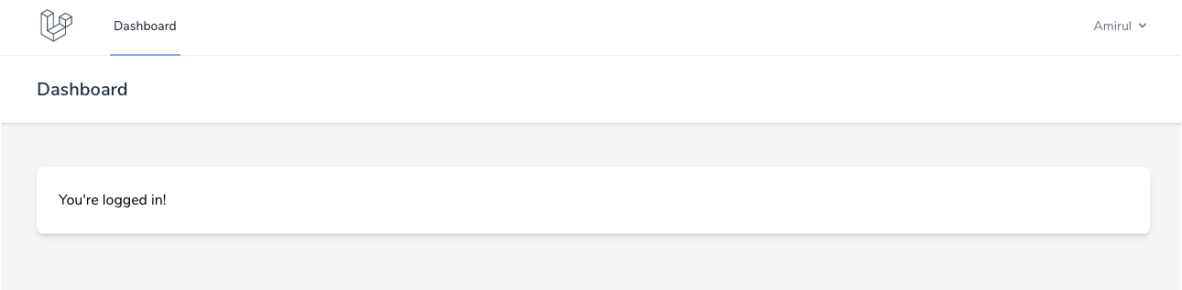
Setelah semua proses selesai, sekarang kita coba akses lagi aplikasi kita di `http://localhost:8000` Akan muncul tautan ***Login*** & ***Register*** pada pojok kanan atas.



Coba daftarkan seorang *user* pada tautan ***Register***



Setelah mendaftar, kita akan diarahkan ke `http://localhost:8000/home`.



Berikutnya kita pastikan semua flow sistem autentikasi berjalan sesuai ekspektasi dengan ***Logout*** & ***Login*** kembali.

Membuat Fungsi CRUD Post

Untuk membuat fungsi CRUD minimal kita paham tentang Route, Model, View, Controller.

- **Route** - secara gampang route ibaratnya adalah gerbang utama aplikasi kita.
- **Model** - secara umum digambarkan sebagai file / class yang berhubungan dengan proses database.
- **View** - bagian dari framework yang tanggung jawabnya adalah mengurusi tampilan.
- **Controller** - biasanya berisi logika atau sebagai jembatan dari Model & View. View menerima inputan dari user diproses controller untuk disimpan ke Model.

Setelah sedikit memahami tentang kebutuhan untuk membuat fungsi CRUD, selanjutnya kita akan mulai praktek.

Membuat Model Post

Agar kita dapat berhubungan dengan tabel `posts` yang sudah kita buat sebelumnya, kita butuh sebuah file / class Model yang akan berinteraksi dengan tabel `posts`. Dalam konvensi Laravel penamaan model harus berbentuk tunggal / *singular*, misal kita memiliki tabel `posts` maka nama modelnya `Post`.

Kita bisa saja membuat nama model dengan sesuka kita dengan syarat menambahkan properti `portected $table = 'nama_tabel`. Untuk membuat file model ini kita bisa gunakan perintah `Artisan`.

```
php artisan make:model Post
```

Setelah dijalankan dan berhasil maka ada file baru yang terbuat di `app/Models` dengan nama `Post.php`, kurang lebih isinya seperti ini.

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    use HasFactory;
}
```

Untuk sementara cukup kita biarkan seperti itu dulu.

Membuat PostController

Selanjutnya yang perlu dibuat adalah controller, dalam pembuatan fungsi CRUD sebenarnya Laravel menyediakan yang namanya `Resource Controller` yang akan memberikan method - method untuk kebutuhan CRUD.

- `index()` untuk menampilkan semua data.
- `create()` untuk menampikan halaman / form yang digunakan membuat data.
- `store()` digunakan untuk memproses data yang dikirim dari view / method `create()` untuk disimpan ke *database*.
- `show()` untuk menampilkan data tunggal, biasanya untuk halaman detail.
- `edit()` untuk menampikan halaman / form yang digunakan memperbarui data.
- `update()` untuk proses memperbarui data ke *database*.
- `destroy()` untuk proses hapus data.

Untuk membuat controller kita juga bisa gunakan perintah `Artisan` dan untuk `Resource Controller` tambahkan flag `-r` atau `--resource`.

```
php artisan make:controller PostController -r
```

File controller baru akan terbuat di `app/Http/Controllers` dengan nama `PostController`.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PostController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
    }

    /**
```

```
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        //
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        //
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
```

```
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {
        //
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
    {
        //
    }
}
```

Kita biarkan seperti itu dulu.

Mendefinisikan Route

Dalam kasus kali ini, kita akan membuat sebuah aplikasi web, maka route didaftarkan pada file `routes/web.php`. Untuk mendaftarkan route masing-masing sesuaikan dengan HTTP Verb dari masing-masing method yang ada. Kurang lebih seperti di bawah ini.

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

Route::get('/post', [PostController::class, 'index']);
Route::get('/post/create', [PostController::class, 'create']);
Route::post('/post', [PostController::class, 'store']);
Route::get('/post/{id}', [PostController::class, 'show']);
Route::get('/post/{id}/edit', [PostController::class, 'edit']);
Route::put('/post/{id}', [PostController::class, 'update']);
Route::delete('/post/{id}', [PostController::class, 'destroy']);
```

Tapi, karena kita menggunakan Resource Controller, Laravel menyediakan penulisan route yang lebih ringkas dengan menggunakan *method* `resource()`.

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

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

Setelah itu, untuk mengecek bahwa *route* di atas sudah terdaftar kita bisa gunakan perintah artisan :

```
php artisan route:list
```

Setelah dijalankan, coba cari *route* yang menggunakan kata yang sudah didaftarkan, dalam kasus kita adalah `post`.

```
|        | GET|HEAD  | post                  | post.index   | App\Http\Controllers\PostController@index   | web |
|        | POST      | post                  | post.store   | App\Http\Controllers\PostController@store   | web |
|        | GET|HEAD  | post/create           | post.create  | App\Http\Controllers\PostController@create  | web |
|        | DELETE    | post/{post}           | post.destroy | App\Http\Controllers\PostController@destroy | web |
|        | PUT|PATCH | post/{post}           | post.update  | App\Http\Controllers\PostController@update  | web |
|        | GET|HEAD  | post/{post}           | post.show    | App\Http\Controllers\PostController@show    | web |
|        | GET|HEAD  | post/{post}/edit      | post.edit    | App\Http\Controllers\PostController@edit    | web |
```

Mengisi Fungsi Setiap Method

Setelah berhasil mendaftarkan *route*, selanjutnya mari kita buat fungsi untuk masing-masing *method* pada `PostController`. Karena kita belum memiliki data *post* sama sekali, maka kita akan mulai dengan fungsi yang berhubungan dengan membuat data.

Dalam *Resource Controller* untuk membuat data, kita akan menggunakan method `create()` dan `store()`.

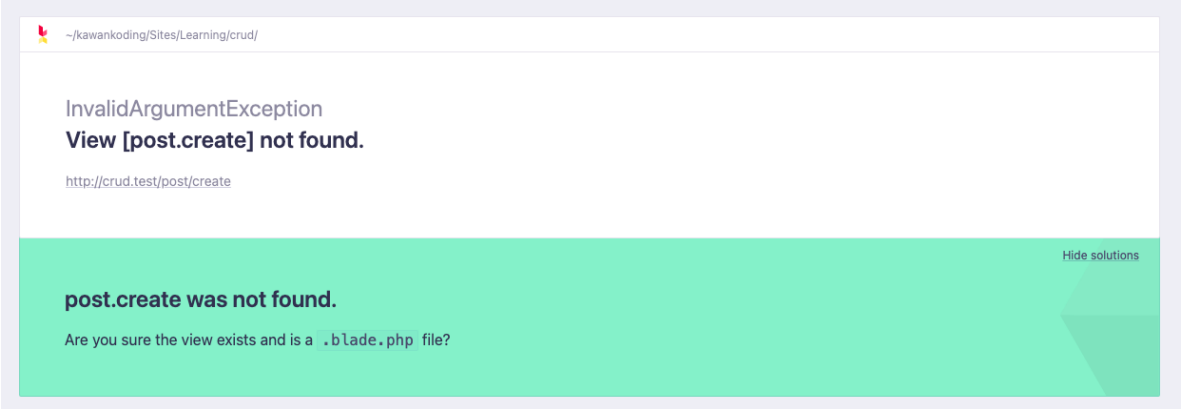
Method `create()` akan berfungsi untuk menampilkan form yang digunakan untuk membuat data, sedangkan *method* `store()` digunakan untuk proses menyimpan data ke dalam *database*.

Membuat Fungsi Menambahkan Data

Kita buat fungsi untuk menampilkan halaman yang akan kita gunakan untuk menambah data. Kita akan return sebuah view yang akan kita gunakan.

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

Setelah kita buat fungsinya, sekarang mari kita coba akses halamannya ke `http://localhost:8000/post/create` , maka akan muncul halaman *error* seperti di bawah ini.



Dari pesan error di atas kita bisa membaca bahwa file *view* `post/create` tidak ditemukan pada `resources/views`, untuk menyelesaikan masalah ini solusinya buat sebuah folder dengan nama `post` di dalam `resources/views` kemudian di dalam folder `post` buat sebuah *file* dengan nama `create.blade.php`.

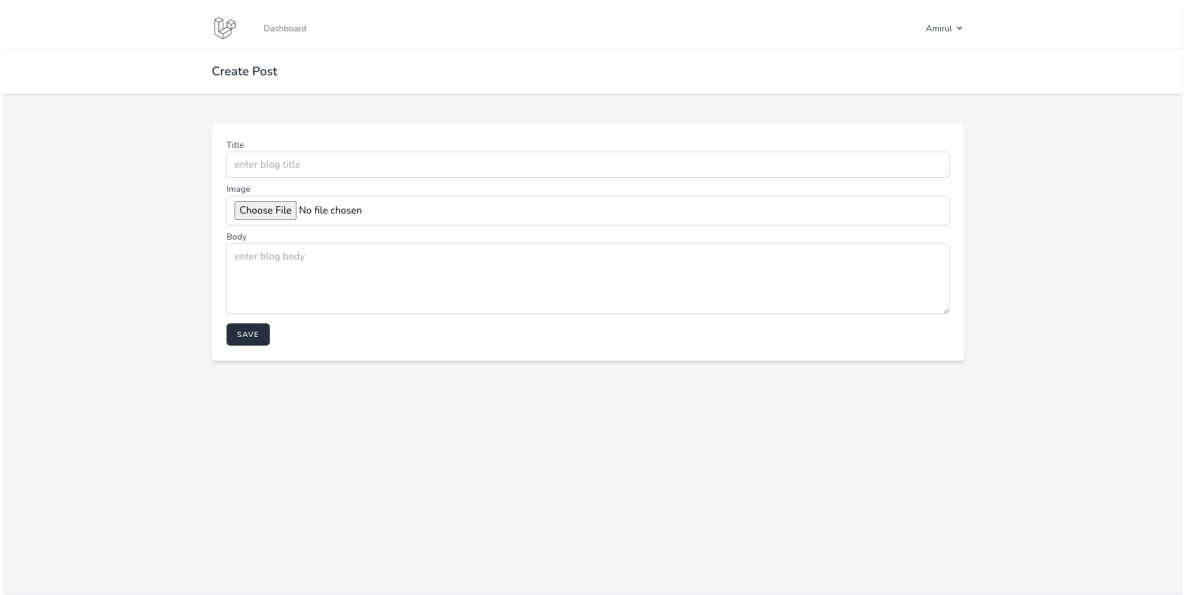
Kemudian isi file tersebut dengan kode html untuk menampilkan form yang dibutuhkan.

```
<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 leading-tight">
            {{ __('Create Post') }}
        </h2>
    </x-slot>

    <div class="py-12">
        <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
```

```
            <div class="bg-white overflow-hidden shadow-md sm:rounded-
lg">
                <div class="p-6 bg-white border-b border-gray-200">
                    <form action="{{ route('post.store') }}"
method="POST" enctype="multipart/form-data">
                        @csrf
                        <div class="mb-2">
                            <x-label for="title">Title</x-label>
                            <input type="text" name="title"
                                   class="w-full form-input rounded-md
shadow-sm @error('title') border border-red-500 @enderror"
placeholder="enter blog title">
                            @error('title')
                            <span class="text-red-500">{{ $message
}}</span>
                            @enderror
                        </div>
                        <div class="mb-2">
                            <x-label for="title">Image</x-label>
                            <input type="file" name="image"
                                     class="w-full form-input rounded-md
shadow-sm @error('image') border border-red-500 @enderror">
                            @error('image')
                            <span class="text-red-500">{{ $message
}}</span>
                            @enderror
                        </div>
                        <div class="mb-2">
                            <x-label for="body">Body</x-label>
                            <textarea name="body" id="body" rows="4"
                                      class="form-input rounded-md
shadow-sm w-full @error('body')
                                          border border-red-500
@enderror"
                                      placeholder="enter blog
body"></textarea>
                            @error('body')
                            <span class="text-red-500">{{ $message
}}</span>
                            @enderror
                        </div>
                        <div>
                            <x-button type="submit">Save</x-button>
                        </div>
                    </form>
                </div>
```

```
            </div>
        </div>
    </div>
</x-app-layout>
```

Dari kode di atas akan menghasilkan tampilan seperti di bawah ini.



Setelah mendapatkan tampilan yang diinginkan, berikutnya kita butuh fungsi untuk memproses simpan datanya, di **Laravel** pada ***Resource Controller*** ini ditampung pada method `store()`. Jadi berikutnya kita akan buat fungsinya dalam *method* tersebut.

```
public function store(Request $request)
{
    $this->validate($request, [
        'title' => ['required'],
        'image' => ['required', 'image'],
        'body' => ['required', 'min:20'],
    ]);

    Post::create([
        'user_id' => auth()->id(),
        'title' => $request->title,
        'image' => $request->file('image')->store('post/image/'),
        'body' => $request->body,
    ]);

    return redirect()->route('post.index');
}
```

Penjelasan Kode

```
$this->validate($request, [
    'title' => ['required'],
    'image' => ['required', 'image'],
    'body' => ['required', 'min:20'],
]);
```

Kode ini untuk memvalidasi data *input* yang masuk dengan ketentuan ketentuannya masing-masing, *key* yang di sebelah kiri adalah nama *field input*, *value* yang ada di kanan adalah *rules* / aturan validasinya.

```
Post::create([
        'user_id' => auth()->id(),
    'title' => $request->title,
    'image' => $request->file('image')->store('post/image/'),
    'body' => $request->body,
]);
```

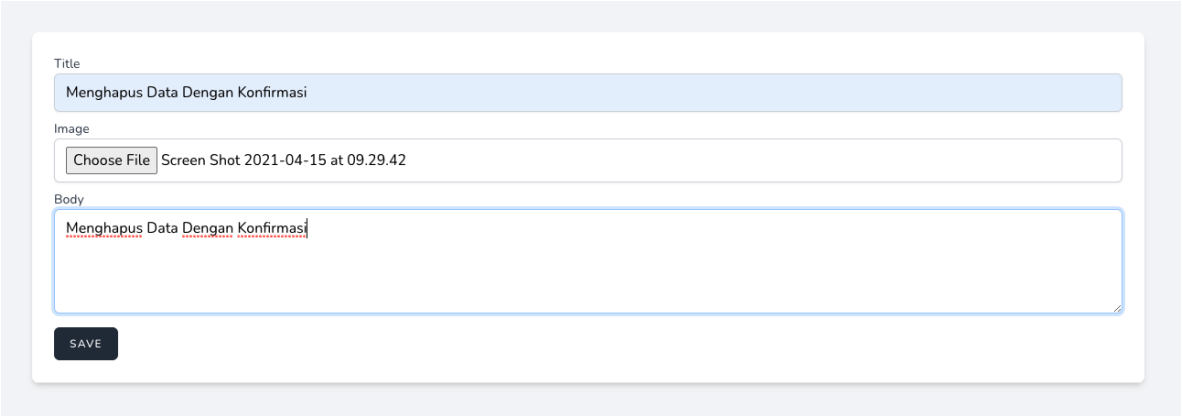
Kode untuk menyimpan data input ke dalam *database* tabel `posts`, *key* di sebelah kiri adalah nama kolom pada tabel `posts` sedangkan *value* yang ada di kanan adalah nilai yang ingin diberikan.

```
return redirect()->route('post.index');
```

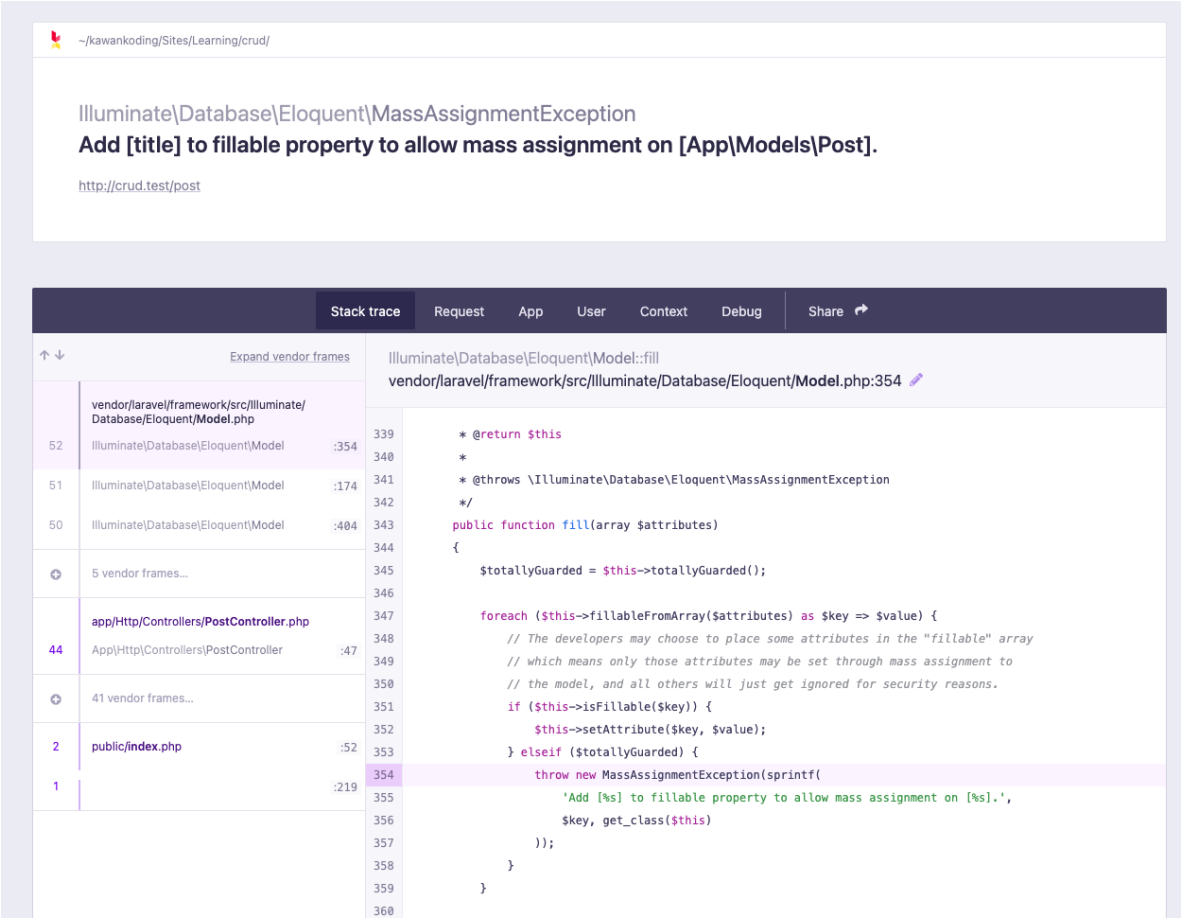
Kode di atas digunakan untuk mengalihkan pengguna setelah proses sebelumnya sukses dan tidak ada *error*, akan diarahkan ke `route('post.index')` atau ke `/post` sesuai URI *route* dari *route* `post.index`.

Jangan lupa juga untuk menambahkan `use App\Models\Post` di atas `class` supaya tidak menemukan *error*.

Setelah membuat fungsi untuk store datanya sekarang kita coba jalankan form sebelumnya dan mengisikan data sesuai kebutuhan dan klik "Save".



Kemudian akan muncul sesuatu yang tidak kita harapkan, YA ! kita ketemu **ERROR**.



Oke, jangan panik, ini sudah biasa di Laravel, yang perlu kita lakukan adalah masuk ke file model `Post` dan menambahkan properti `$fillable` seperti ini.

```
class Post extends Model
{
    use HasFactory;

    protected $fillable = [
        'user_id',
        'title',
        'image',
        'body'
    ];
}
```

Setelah menambahkan kode di atas ke model `Post` sekarang coba ulangi proses menambahkan datanya. Harusnya sukses dan akan di *redirect* ke halaman / *route* `http://localhost:8000/post`, tapi tampilannya hanya putih bersih seperti bayi tanpa dosa, jelas saja kan kita belum buat fungsi & tampilannya.

Menampilkan Data Pada Method Index

Sekarang untuk mengisi halaman `http://localhost:8000/post` agar tidak kosong, akan kita tampilkan data `Post` yang dibuat oleh masing - masing `User` yang sudah buat post. Bayangkan saja kita buat dengan SQL Query, kira kira bagaimana querynya ?.

```
select * from `posts` where `user_id` = ?
```

Karena kita menggunakan Laravel, kita bisa memanfaatkan model dan *query builder*.

```
App\Models\Post::where('user_id', auth()->id())->get();
```

Oke, langsung aja implementasi ke controller untuk membuat fungsinya berguna.

```
public function index()
{
    return view('post.index', [
       'posts' => Post::where('user_id', auth()->id())->get(),
    ]);
}
```

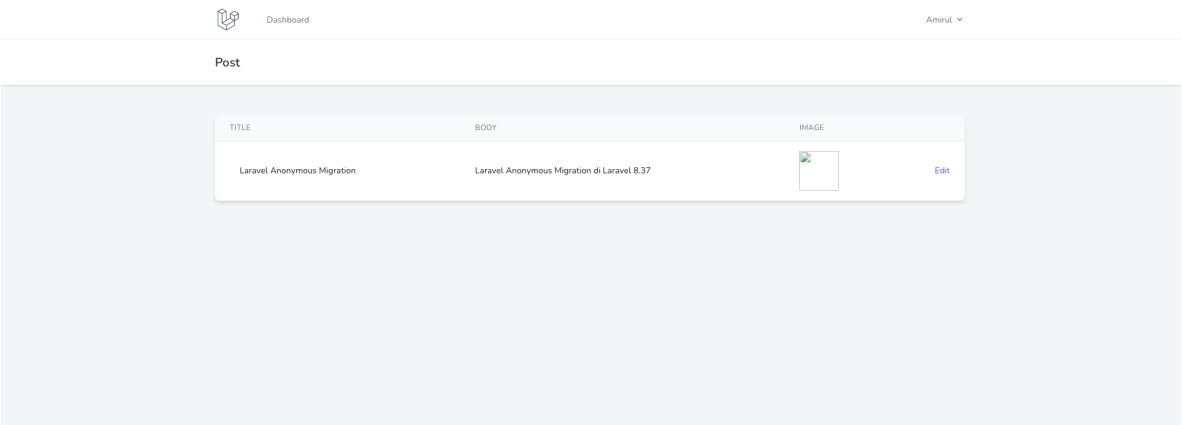
Setelah menambahkan kode tersebut, selanjutnya kita buat file *views* di dalam folder `post` dengan nama `index.blade.php` atau path lengkapnya `resources/views/post/index.blade.php` karena kita belum punya filenya memang. Isi filenya dengan kode di bawah ini.

```
<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 leading-tight">
            {{ __('Post') }}
        </h2>
    </x-slot>

    <div class="py-12">
        <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
            <div class="bg-white overflow-hidden shadow-md sm:rounded-
lg">
                <div class="flex flex-col">
```

```
                    <div class="-my-2 overflow-x-auto sm:-mx-6 lg:-
mx-8">
                        <div class="py-2 align-middle inline-block min-
w-full sm:px-6 lg:px-8">
                            <div class="shadow overflow-hidden border-b
border-gray-200 sm:rounded-lg">
                                <table class="min-w-full divide-y
divide-gray-200">
                                    <thead class="bg-gray-50">
                                    <tr>
                                        <th scope="col"
                                            class="px-6 py-3 text-left
text-xs font-medium text-gray-500 uppercase tracking-wider">
                                            Title
                                        </th>
                                        <th scope="col"
                                            class="px-6 py-3 text-left
text-xs font-medium text-gray-500 uppercase tracking-wider">
                                            Body
                                        </th>
                                        <th scope="col"
                                            class="px-6 py-3 text-left
text-xs font-medium text-gray-500 uppercase tracking-wider">
                                            Image
                                        </th>
                                        <th scope="col" class="relative
px-6 py-3">
                                            <span class="sr-
only">Edit</span>
                                        </th>
                                    </tr>
                                    </thead>
                                    <tbody class="bg-white divide-y
divide-gray-200">
                                    @foreach($posts as $post)
                                        <tr>
                                            <td class="px-6 py-4
whitespace-nowrap">
                                                <div class="flex items-
center">
                                                    <div class="ml-4">
                                                        <div
                                                            class="text-
sm font-medium text-gray-900">{{ $post->title }}</div>
                                                    </div>
                                                </div>
```

```
                                        </td>
                                        <td class="px-6 py-4
whitespace-nowrap">
                                            <div class="text-sm
text-gray-900">{{ $post->body }}</div>
                                        </td>
                                        <td class="px-6 py-4
whitespace-nowrap">
                                            <img src="{{
asset('storage/' . $post->image) }}" alt=""
                                                 class="h-16 w-16">
                                        </td>
                                        <td class="px-6 py-4
whitespace-nowrap text-right text-sm font-medium">
                                            <a href="#" class="text-
indigo-600 hover:text-indigo-900">Edit</a>
                                        </td>
                                    </tr>
                                @endforeach
                                </tbody>
                            </table>
                        </div>
                    </div>
                </div>
            </div>
        </div>
</x-app-layout>
```

Muat ulang halaman `/post`, kemudian akan tampil, tampilan yang kurang lebih seperti ini.



Seperti yang kita lihat, gambar yang sudah diupload tidak tampil, kenapa begitu? karena

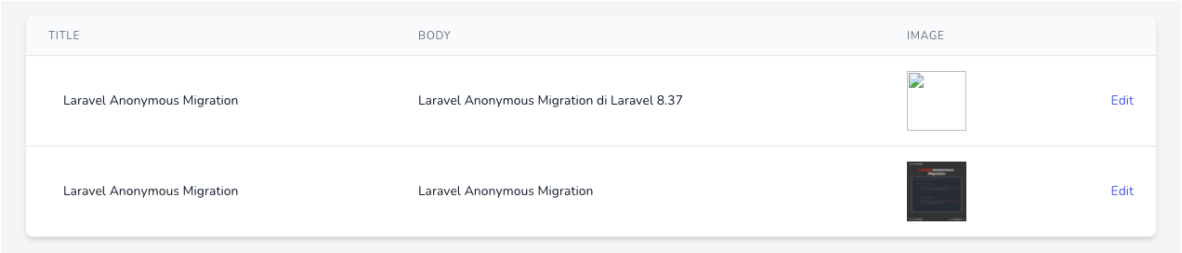
konfigurasi *default* Laravel menggunakan *filesystem* lokal yang dimana dia tidak akan bisa diakses ke oleh *public*. Tambahkan *environment variable* pada `.env`.

```
FILESYSTEM_DRIVER=public
```

Sekarang aplikasi kita sudah menggunakan *driver public* dan visibilitinya public, kemudian buat *symlink* dari *path storage* ke *path public*.

```
php arisan storage:link
```

Sekarang kita coba buat data lagi di `post/create` lengkap dengan gambarnya. Maka pada data yang baru kita bisa melihat gambarnya bisa ditampilkan dengan baik.



Membuat Halaman & Fungsi Edit

Selanjutnya membuat halaman edit, agar aplikasinya berkesinambungan, kita sesuaikan dulu *link* pada *file* `index.blade.php`

```
<!-- sebelum /-->
<a href="#" class="text-indigo-600 hover:text-indigo-900">Edit</a>

<!-- sesudah /-->
<a href="{{ route('post.edit', $post)}}" class="text-indigo-600
hover:text-indigo-900">Edit</a>
```

Pada `PostController` kita tambahkan pada method `edit()` untuk menampilkan halaman view `post/edit.blade.php`

```
public function edit($id)
{
    return view('post.edit', [
        'post' => Post::find($id), // untuk mengambil data Post sesuai
dengan id yang diterima
    ]);
}
```

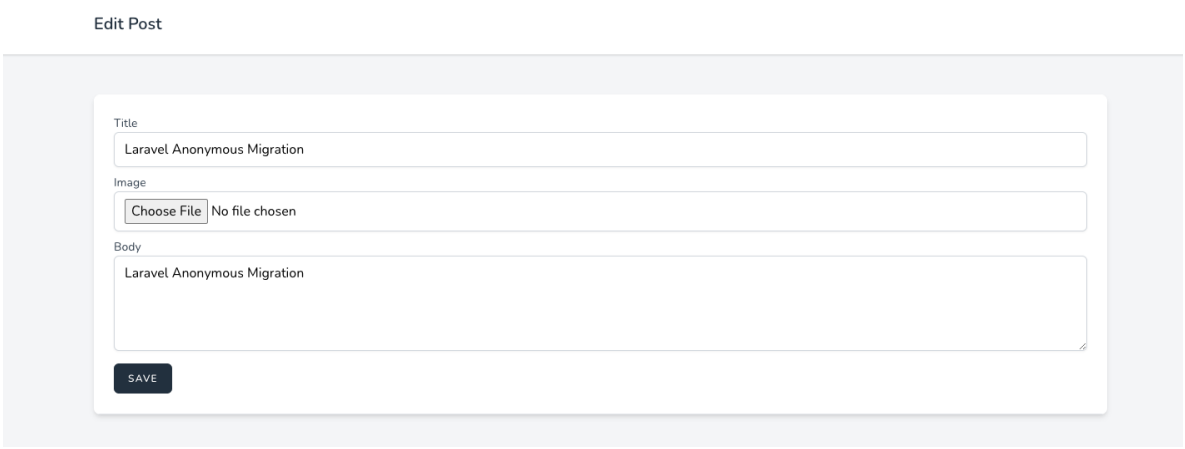
Kemudian buat *file views* `post/edit.blade.php` karena kita belum memilikinya. Kemudian isinya kurang lebih sama dengan `post/create.blade.php` hanya ada sedikit perubahan agar menampilkan data yang sudah ada atau data yang akan diubah.

```
<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 leading-tight">
            {{ __('Edit Post') }}
        </h2>
    </x-slot>

    <div class="py-12">
        <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
            <div class="bg-white overflow-hidden shadow-md sm:rounded-
lg">
                <div class="p-6 bg-white border-b border-gray-200">
                    <form action="{{ route('post.update', $post) }}"
```

```
method="POST" enctype="multipart/form-data">
                        @csrf
                        @method('PUT')
                        <div class="mb-2">
                            <x-label for="title">Title</x-label>
                            <input type="text" name="title"
                                   class="w-full form-input rounded-md
shadow-sm @error('title') border border-red-500 @enderror"
placeholder="enter blog title" value="{{ old('title') ?? $post->title
}}">
                            @error('title')
                            <span class="text-red-500">{{ $message
}}</span>
                            @enderror
                        </div>
                        <div class="mb-2">
                            <x-label for="title">Image</x-label>
                            <input type="file" name="image"
                                   class="w-full form-input rounded-md
shadow-sm @error('image') border border-red-500 @enderror">
                            @error('image')
                            <span class="text-red-500">{{ $message
}}</span>
                            @enderror
                        </div>
                        <div class="mb-2">
                            <x-label for="body">Body</x-label>
                            <textarea name="body" id="body" rows="4"
                                      class="form-input rounded-md
shadow-sm w-full @error('body')
                                          border border-red-500
@enderror"
                                      placeholder="enter blog body">{{
old('body') ?? $post->body }}</textarea>
                            @error('body')
                            <span class="text-red-500">{{ $message
}}</span>
                            @enderror
                        </div>
                        <div>
                            <x-button type="submit">Save</x-button>
                        </div>
                    </form>
                </div>
            </div>
        </div>
```

```
    </div>
</x-app-layout>
```

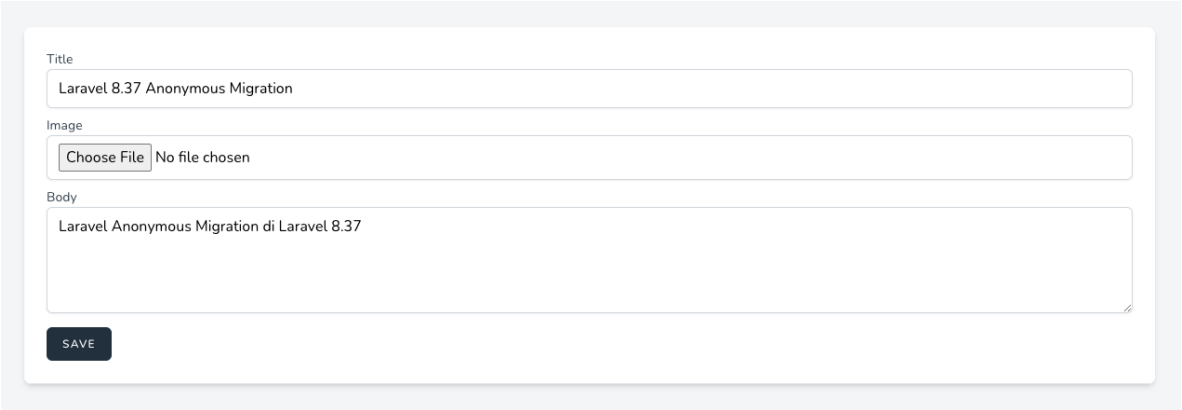
Kemudian klik salah satu *link* 'edit' pada halaman `/post`, kemudian kita akan diarahkan ke `/post/{id}/edit` dan disajikan tampilan yang seperti ini.



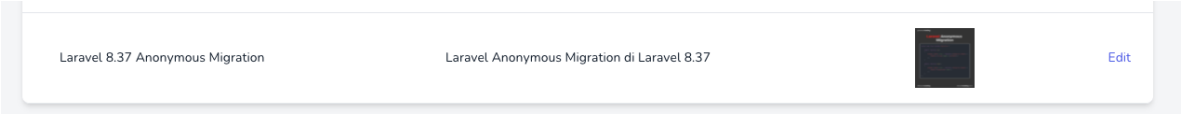
Jika kita klik "Save" kita tidak akan menemukan *error* karena *route* sudah otomatis terdaftar dengan menggunakan *route resource*. Tentu supaya berfungsi kita akan tambahkan kode untuk method `update()`.

```
use Illuminate\Support\Facades\Storage;

/**
* kode kode lainnya
*/

public function update(Request $request, $id)
{
    $this->validate($request, [
        'title' => ['required'],
        'body' => ['required', 'min:20'],
    ]);

    $post = Post::find($id);
    $image = $post->image; // membuat variabel $image dengan nilai
adalah image lama data yang diubah

    if ($request->hasFile('image')) { // mengecek jika request memiliki
file pada field image, jika tidak ada file maka operasi didalam ini
tidak akan diekseskusi
       Storage::delete($image); // digunakan menghapus file lama karena
tidak akan digunakan lagi, memanfaatkan variabel $image yang berisi path
file sebelumnya
       $image = $request->file('image')->store('post/image/'); //
mengoverride variabel $image dengan file baru yang diupload dan
digunakan untuk mengupdate data.
    };

    $post->update([
        'title' => $request->title,
        'image' => $image,
        'body' => $request->body,
    ]);

    return redirect()->route('post.index');
}
```

Kemudian mari kita ubah salah satu post yang sudah ada dan simpan perubahannya.



Maka setelah berhasil kita akan mendapati data yang sudah ada berubah sesuai dengan data yang baru.

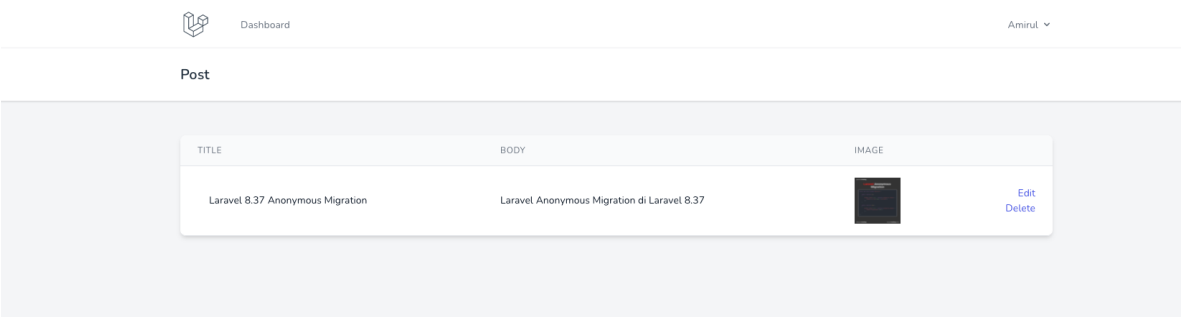


Membuat Fungsi Hapus Data

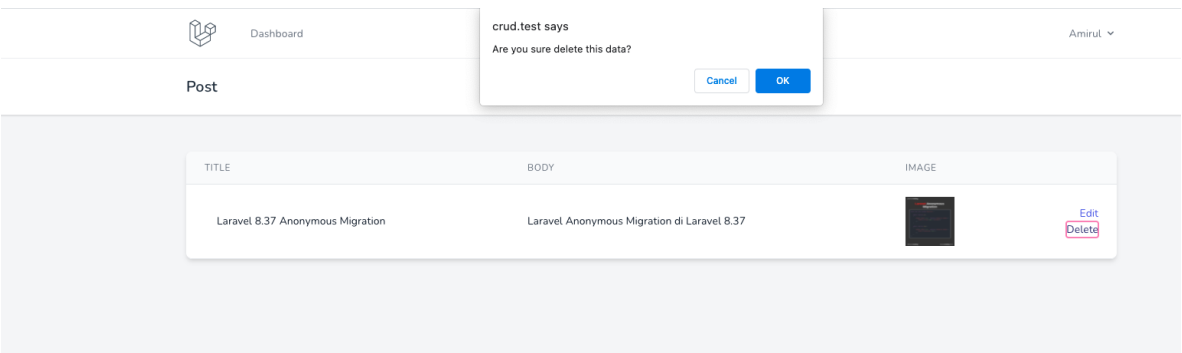
Setelah memiliki fungsi edit data untuk memperbaiki data yang salah, lumrahnya kita juga akan membutuhkan fungsi untuk menghapus data yang mungkin salah dimasukkan atau sudah tidak dibutuhkan. Pertama kita akan perbarui terlebih dulu untuk tampilan yang ada di `post/index.blade.php` setelah bagian tautan untuk edit data.

```
<td class="px-6 py-4 whitespace-nowrap text-right text-sm font-medium">
  <a href="{{ route('post.edit', $post) }}" class="text-indigo-600
hover:text-indigo-900">Edit</a>
  <form action="{{ route('post.destroy', $post) }}" method="POST">
      @csrf
      @method('DELETE')
      <button class="text-indigo-600 hover:text-indigo-900"
onclick="return confirm('Are you sure delete this
data?')">Delete</button>
  </form>
</td>
```

Tampilannya akan seperti ini.



Ketika klik "Delete" akan muncul *alert* untuk konfirmasi hapus data atau tidak.



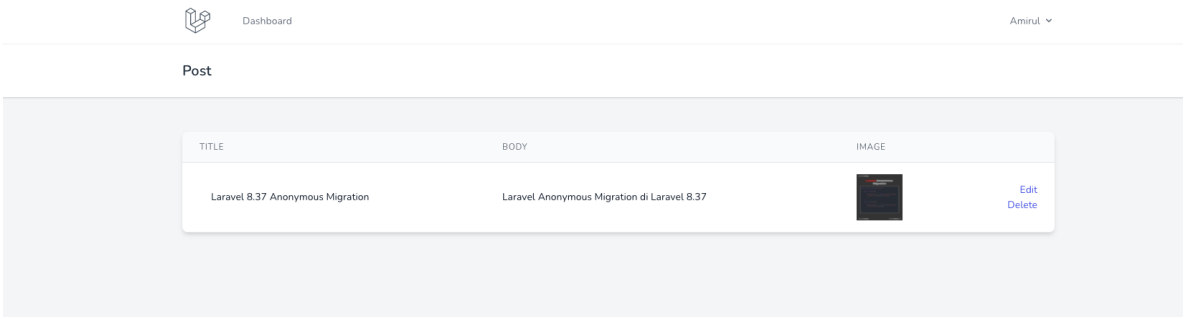
Sekarang jika kita klik "OK" tidak akan terjadi aksi apapun, karena kita belum membuat

fungsi pada *method* `destroy()`. Kita buat fungsi untuk hapus dan *redirect* ke halaman `post.index`.

```
public function destroy($id)
 {
     Post::find($id)->delete();

     return redirect()->route('post.index');
 }
```

Memoles Tampilan

Langkah akhir pada pembuatan fungsi CRUD ini adalah memoles tampilan agar lebih *user friendly* dengan menambahkan beberapa detail kecil yang belum kita tambahkan.



Jika kita perhatikan di sini ada beberapa kekurangan pada tampilan ini, pertama kita tidak punya navigasi untuk menuju halaman `/post` ini, kedua kita tidak memiliki navigasi untuk membuat sebuah post baru atau ke `/post/create`.

Membuat Navigasi Ke Halaman Post

Untuk menambahkan navigasi yang ada pada layout dari ***Breeze*** kita buka *file* `resources/views/layouts/navigation.blade.php` kemudian cari pada segmen ***Navigation Links*** tambahkan satu component `nav-link` dengan tag `<x-nav-link></x-nav-link>`.

```
<!-- Navigation Links -->
<div class="hidden space-x-8 sm:-my-px sm:ml-10 sm:flex">
    <x-nav-link href="{{ route('dashboard') }}"
:active="request()->routeIs('dashboard')">
        {{ __('Dashboard') }}
    </x-nav-link>
        <x-nav-link href="{{ route('post.index') }}"
:active="request()->routeIs('post.index')">
        {{ __('Post') }}
    </x-nav-link>
</div>
```

Sekarang kita bisa melihat ada tautan ke `/post` di sebelah `Dashboard` dan kondisinya akan aktif ketika kita sedang berada di halaman `/post`.

Menambahkan Navigasi Ke Halaman Buat Post

Kekurangan kedua tidak memiliki navigasi ke halaman buat *post*, akan kita tambahkan di atas daftar *post* yang sudah dibuat pada file `post/index.blade.php`, untuk class dari linknya kita ambil dari component `button` dari ***Breeze***.

```
<div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
        <a href="{{ route('post.create') }}" class="inline-flex items-
center px-4 py-2 bg-gray-800 border border-transparent rounded-md font-
semibold text-xs text-white uppercase tracking-widest hover:bg-gray-700
active:bg-gray-900 focus:outline-none focus:border-gray-900
focus:shadow-outline-gray disabled:opacity-25 transition ease-in-out
duration-150 mb-4">Create Post</a>
...
</div>
```

Sekarang setelah memperbaiki dua hal tersebut, tampilannya akan seperti ini.