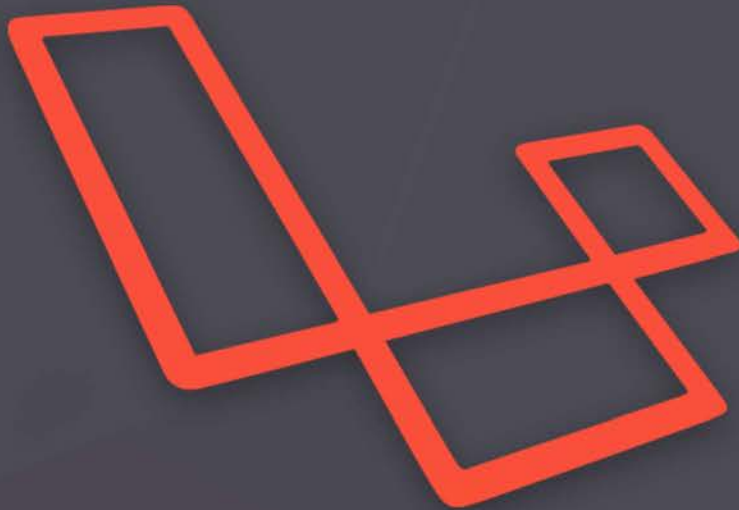


GRATIS

PANDUAN MEMULAI PROJECT **LARAVEL** 5.4



SW TEAM

Panduan Memulai Project Laravel 5.4

Step by Step bagaimana memulai project-mu dengan Laravel

© 2017 SW TEAM

*Karya kecil ini didedikasikan untuk para coder yang masih 'merem'
dengan Laravel agar sedikit lebih 'melek' dengan "The PHP
Framework For Web Artisans".*

*Kami bukan ahlinya, tapi kami mencoba untuk membuatnya lebih
mudah.*

- Tim Penyusun -

Contents

Kata Pengantar.....	i
Changelog.....	ii
Source Code	ii
Menginstall Laravel.....	3
Persiapan.....	3
Instalasi.....	2
Melalui Laravel Installer	2
Menginstall melalui Composer Create-Project	2
Konfigurasi	2
Konfigurasi file .env	2
Konfigurasi app.php	3
Konfigurasi AppServiceProvider.php	3
Menjalankan di Browser	4
User Aplikasi	6
Membuat Authentication.....	6
Menjalankan Migrate Table users dan password_resets	6
Membuat Role (Level User)	7
Instalasi Laratrust	7
Setup Laratrust.....	9
Membuat Sample User	10
Konfigurasi Register	12
Membuat Route Group Admin.....	13
Menginstall Package/Library.....	14

Bootstrap.....	14
JQuery.....	15
Font Awesome	15
DataTables	15
Laravel DataTables	16
Laravel Collective	19
Installasi Laravel Collective	19
Membuat Form Menggunakan Collective.....	20
Fancybox.....	22
Responsive File Manager	23
TinyMCE.....	27
Membuat Halaman Baru	29
Merancang Table.....	29
Models.....	30
Pengenalan	30
Membuat Model Table Posts.....	30
Migrations.....	33
Pengenalan	33
Membuat Migration Posts.....	36
Seeding.....	37
Pengenalan	37
Membuat Seeder Posts	43
Controllers	44
Pengenalan	44
Membuat PostController	44
Routing.....	47

Pengenalan	47
Membuat Routing Post.....	47
Menyiapkan Folder.....	48
Membuat File Index Post.....	49
Menyiapkan Interface	50
Menampilkan Data dengan DataTables.....	54
Membuat Halaman Form	57
Form Input.....	57
Menyiapkan File dan Controller	57
Membuat Form dengan laravelcollective/html	58
Menambahkan Responsive File Manager	63
Menambahkan TinyMCE	65
Menambahkan Flash Message.....	67
Form Edit	70
Proses Hapus Data.....	75
Penutup	77
Daftar Situs Belajar laravel.....	77
Jangan Berhenti di Ebook ini.....	78

Kata Pengantar

Ebook ini kami susun untuk Anda yang benar-benar pemula dan punya keinginan untuk mempelajari Laravel secara *step by step* dan *learning by doing*. Kami mencoba untuk meringkas materi dan mementingkan bahasa praktik dengan memberikan contoh langsung dalam bentuk script. Anda bisa membaca dokumentasi Laravel sebagai pelengkap ebook ini atau mencari referensi lainnya yang relevan dengan pembahasan.

“Apa yang akan saya pelajari di ebook ini ?”

Anda akan belajar hal-hal dasar bagaimana memulai membuat project Laravel :

1. Menginstall Laravel dan meng-*configurasi*-nya.
2. Membuat *Authentication*, Role, memasukkan data dengan Seeder.
3. Menginstall Package/Library yang dibutuhkan.
4. Membuat model, controller, dan Route.
5. Melakukan migration dan seeding.
6. Menyiapkan folder, file, dan interface.
7. Membuat halaman View untuk menampilkan data.
8. Menampilkan data dengan DataTables.
9. Membuat Form input dan edit beserta proses CRUD.

Ebook ini jelas jauh dari kesempurnaan, masih banyak kekurangan sebagai karya manusia yang tidak pernah lepas dari kesalahan. Jika Anda ingin berbagi masukan atau saran, atau tanggapan hubungi kami lewat kontak email **websiger@gmail.com**. Setiap *feedback* yang kami terima dari Anda akan membawa manfaat bagi pembaca lainnya.

Bandar Lampung, 12 Juni 2017

Tim Penyusun

Changelog

Rilis Versi Beta 1.0 160617 – 16 Juni 2017

- Menambahkan Responsive File Manager dan TinyMCE
- Memperbaiki kesalahan type

Rilis versi Alpha 1.0 – 12 Juni 2017

Source Code

Source code contoh project dalam ebook terdapat di Github, dapat Anda unduh dalam bentuk zip atau melakukan *clone repository* dengan HTTPS via git :

```
git clone https://github.com/sigerweb/larakod.git
```

Setelah diunduh atau *di-clone*, buka command line dan ikuti langkah-langkah berikut ini untuk melakukan penginstallan

1. Jalankan `composer install` untuk menginstall dependencies Laravel.
2. Jalankan perintah `cp .env.example .env` untuk generate file `.env`.
3. Jalankan perintah `php artisan key:generate` untuk generate key.
4. Buat database baru, kemudian buka file `.env`, atur username, password, dan nama database.
5. Jalankan perintah `php artisan migrate` untuk melakukan migration table.
6. Jalankan perintah `php artisan db:seed` untuk memasukkan data pada table.
7. Jalankan perintah `php artisan serve` untuk menjalankan server Laravel.
8. Buka Browser, ketik `localhost:8000` untuk menjalankan project.
9. Gunakan username: `admin@larakod.com` dan password: `admin123` untuk login sebagai Admin.

Menginstall Laravel

Persiapan

Sebelum menginstall Laravel, ada beberapa hal yang harus dipersiapkan.

1. Composer

Laravel memerlukan Composer untuk *manage* dependencies. Pastikan komputer kamu sudah terpasang Composer, jika belum segera install Composer. (Pengguna Windows dapat mengunduh Composer di [sini](https://getcomposer.org/Composer-Setup.exe)¹.)

2. PHP dengan versi minimal 5.6.4

Versi PHP yang umumnya digunakan pada saat ini sudah versi 5.6.4 ke atas, apalagi jika kamu menggunakan XAMPP yang tergolong baru. Jika XAMPP yang kamu gunakan sudah lawas dengan versi kurang dari 5.6.4 saatnya kamu upgrade XAMPP kamu. Kamu yang bukan pengguna XAMPP atau belum menggunakan XAMPP dan ingin menggunakan XAMPP, download di [sini](https://www.apachefriends.org/download.html)². Semua versi XAMPP pada halaman *download* sudah mendukung Laravel karena sudah melampaui versi minimalnya.

3. Lain-lain

Bekerja dengan Laravel tidak bisa dilakukan secara offline apalagi pada saat mulai men-*develop*, kamu harus terhubung dengan internet untuk melakukan instalasi package-package atau libray-library yang diperlukan. Hal berikutnya, jangan pernah tutup command line-mu, tetaplah selalu terbuka dan sudah pada lokasi direktori project. Command line akan banyak digunakan untuk menjalankan perintah Artisan Laravel, menjalankan server Laravel, dan lain-lain. Gunakan text editor kesayangan seperti Sublime Text, Atom, atau yang lain-lain.

¹ <https://getcomposer.org/Composer-Setup.exe>

² <https://www.apachefriends.org/download.html>

Instalasi

Kita bisa menginstall Laravel melalui Laravel Installer atau Composer Create-Project.

Melalui Laravel Installer

Download Laravel Installer melalui Composer :

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

lanjutkan perintah `laravel new` untuk membuat project aplikasi :

```
laravel new larakod
```

Menginstall melalui Composer Create-Project

Kita juga dapat langsung membuat project Laravel dengan menggunakan Composer.

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

Konfigurasi

Setelah selesai melakukan instalasi Laravel, kita perlu melakukan konfigurasi file `.env`, `app.php`, dan `AppServiceProvider.php`.

kita juga perlu untuk membuat **Authentication** dan menjalankan migrate table `users` dan table `password_resets`.

Konfigurasi file .env

Buat database baru bernama **dbkod**. Buka file `.env`, Pada isian `APP_NAME` ubah Laravel menjadi Larakod. Sesuaikan isian database, username, dan password MySQL.

.env

```
APP_NAME=Laravel Larakod
...
DB_DATABASE=homestead dbkod
DB_USERNAME=homestead root
DB_PASSWORD=secret
```

Konfigurasi app.php

Buka file `app.php` yang berada di dalam folder `config/`, ganti Laravel menjadi Larakod.

config/app.php

```
'name' = env('APP_NAME', 'Laravel Larakod')
```

Konfigurasi AppServiceProvider.php

Buka file `AppServiceProvider.php` yang berada di dalam `app/Providers`

Tambahkan `use Illuminate\Support\Facades\Schema;`

Masukkan `Schema::defaultStringLength(191);` pada method boot

app/Providers/AppServiceProvider.php

```
<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use Illuminate\Support\Facades\Schema;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        Schema::defaultStringLength(191);
    }

    ...
}
```

Menjalankan di Browser

Ketik di *address bar* browser :

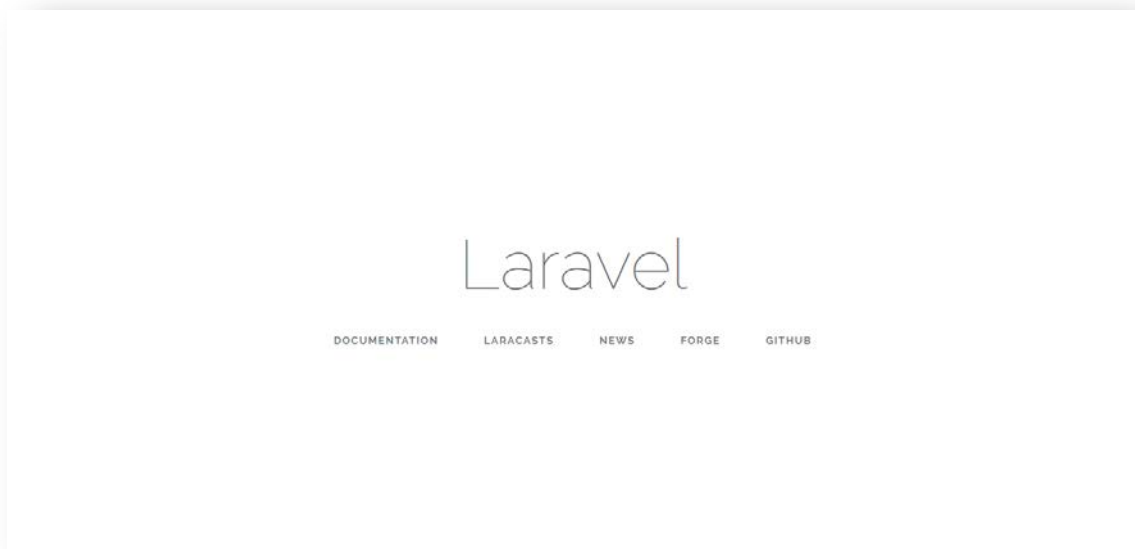
```
http://localhost/larakod/public
```

Atau, menjalankan web server Laravel :

```
php artisan serve  
// Laravel development server started: <http://127.0.0.1:8000>
```

Buka di *address bar* browser :

```
http://localhost:8000
```



Gambar 1 Halaman depan aplikasi

Bagi yang baru pertama kali belajar Laravel akan bingung darimana tampilan ini berasal. Laravel merupakan framework PHP MVC, jadi yang mengatur tampilan sesuai dengan request user adalah tugas Controller. Controller di Laravel tidak bekerja sendiri, tetapi dibantu oleh satu komponen yang bernama Router. Router di Laravel terdapat di file `web.php` di dalam folder `routes`.

Kalau kita buka `web.php`, kita akan menjumpai script berikut:

routes/web.php

```
<?php

/*
|-----
| Web Routes
|-----
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Route::get('/', function () {
    return view('welcome');
});

...
```

Dapat diketahui dari script di atas jika mengakses root (/) maka akan dijalankan method `view('welcome')`. Apa itu “welcome” ? welcome adalah nama file. Laravel menggunakan sistem templating yang bernama blade sehingga file yang terkait dengan tampilan sebuah halaman web mendapatkan akhiran `.blade.php`, uniknya akhiran `.blade.php` tidak perlu dituliskan ketika memanggil file tersebut. Jadi halaman depan yang kita lihat berasal dari file `welcome.blade.php`. File ini ada di folder `resources/views/`. ☞

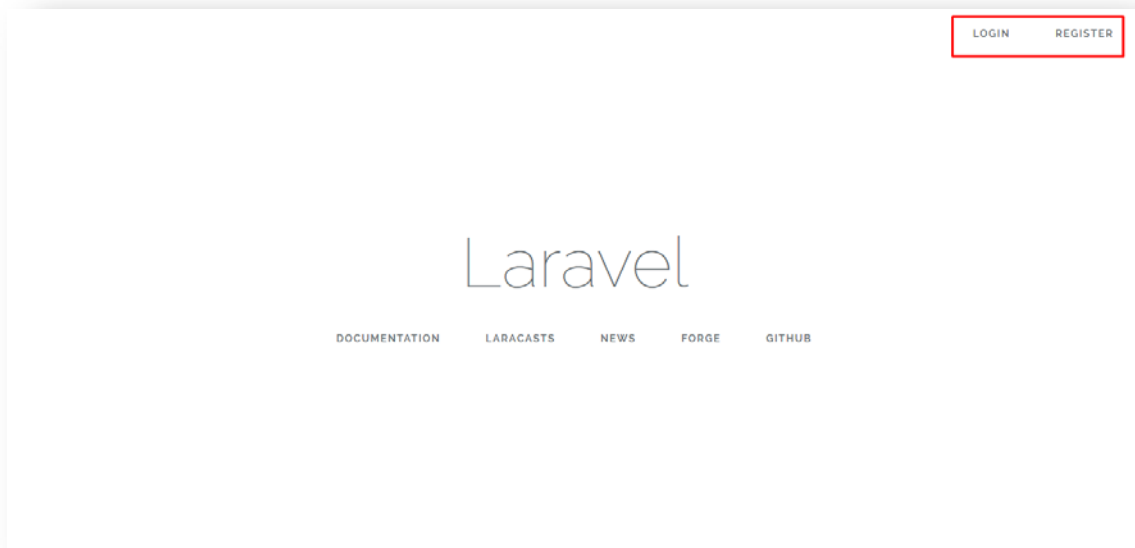
User Aplikasi

Membuat Authentication

```
php artisan make:auth
```

```
// Authentication scaffolding generated successfully
```

Setelah Authentication berhasil dibuat akan muncul link login dan register pada halaman Laravel.



Gambar 2 Muncul Link Login dan Register

Menjalankan Migrate Table users dan password_resets

Lakukan migrate dengan menggunakan perintah Artisan:

```
php artisan migrate
```

```
//Migration table created successfully.
```

```
//Migrating: 2014_10_12_000000_create_users_table
```

```
//Migrated: 2014_10_12_000000_create_users_table
```

```
//Migrating: 2014_10_12_100000_create_password_resets_table
```

```
//Migrated: 2014_10_12_100000_create_password_resets_table
```

Membuat Role (Level User)

Kita akan menggunakan Package [Santigarcor/Laratrust](https://github.com/santigarcor/laratrust)³ untuk membuat Role (level user) di project Laravel kita.

Instalasi Laratrust

```
composer require "santigarcor/laratrust:3.2.*"

//./composer.json has been updated

//Loading composer repositories with package information

//Updating dependencies (including require-dev)

//Package operations: 2 installs, 0 updates, 0 removals

// - Installing kkszymanowski/traitor (0.2.3): Loading from cache

// - Installing santigarcor/laratrust (3.2.3): Loading from cache

//Writing lock file

//Generating optimized autoload files

//> Illuminate\Foundation\ComposerScripts::postUpdate

//> php artisan optimize

//Generating optimized class loader

//The compiled services file has been removed.
```

Buka file `app.php`, tambahkan pada array `providers` dan `aliases`.

config/app.php

```
<?php

/*
|-----
| Autoloaded Service Providers
|-----
|
| The service providers listed here will be automatically loaded on the
| request to your application. Feel free to add your own services to
| this array to grant expanded functionality to your applications.
|
*/

'providers' => [

    /*
```

³ <https://github.com/santigarcor/laratrust>

```

    * Package Service Providers...
    */
    ...

```

```

    Laratrust\LaratrustServiceProvider::class,

```

```

],

```

```

/*
|-----
| Class Aliases
|-----
|
| This array of class aliases will be registered when this application
| is started. However, feel free to register as many as you wish as
| the aliases are "lazy" loaded so they don't hinder performance.
|
*/

```

```

'aliases' => [

```

```

    ...
    'Laratrust' => Laratrust\LaratrustFacade::class,

```

Buka `Kernel.php` yang terdapat dalam direktori `app/Http/`.

Tambahkan pada `$routeMiddleware` tiga baris code berikut ini :

app/Http/Kernel.php

```

protected $routeMiddleware = [
    ...
    'role' => \Laratrust\Middleware\LaratrustRole::class,
    'permission' => \Laratrust\Middleware\LaratrustPermission::class,
    'ability' => \Laratrust\Middleware\LaratrustAbility::class,
];

```

Lakukan publish :

```

php artisan vendor:publish --tag="laratrust"

//Copied File [\vendor\santigarcor\laratrust\src\config\config.php] To
//[\config\laratrust.php]

//Copied File [\vendor\santigarcor\laratrust\src\config\laratrust_seeder.php] To
//[\config\laratrust_seeder.php]

//Publishing complete.

```


Setup Laratrust

Langkah berikutnya yaitu melakukan *setup* Laratrust.

```
php artisan laratrust:setup
```

Perintah ini akan meng-*generate* migrations, membuat model Role dan model Permission, serta menambahkan LaratrustUserTrait ke model User

Jika muncul pertanyaan:

```
Proceed with the migration creation ? (yes/no) [yes]
```

Ketik: yes

```
> yes
// Creating migration...
// Migration successfully created!
// Creating Role model
// Role model created successfully.
// Creating Permission model
// Permission model created successfully.
// Adding LaratrustUserTrait to User model
// LaratrustUserTrait added successfully to App\User
```

Hasilnya berupa file migration `<timestamp>_laratrust_setup_tables.php` , dua file model: `Permission.php` dan `Role.php`, dan menambahkan *trait* ke file model `User.php`.

app/User.php

```
<?php

namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Laratrust\Traits\LaratrustUserTrait;

class User extends Authenticatable
{
    use LaratrustUserTrait;
    use Notifiable;
    ...
}
```

Kemudian, jalankan perintah Composer dump-autoload.

```
composer dump-autoload  
// Generating optimized autoload files
```

Jalankan perintah Artisan migrate untuk meng-generate file migrations ke database

```
php artisan migrate  
// Migrating: 2017_05_31_160348_laratrust_setup_tables  
// Migrated: 2017_05_31_160348_laratrust_setup_tables
```

Setelah migration dijalankan, akan muncul lima table baru di database :

- **roles** - menyimpan data **role**
- **permissions** - menyimpan data **permission**
- **role_user** - menyimpan relasi *polymorphic* antara table **roles** table dan **users**
- **permission_role** - menyimpan relasi *many-to-many* antara table **roles** dan table **permissions**
- **permission_user** - menyimpan relasi *polymorphic* antara table **users** dan table **permission**

Membuat Sample User

Kita akan memasukkan data ke dalam table **users** dengan menggunakan file Seeder. File Seeder dapat dibuat dengan perintah Artisan **make:seeder** :

```
php artisan make:seeder UsersSeeder  
// Seeder created successfully.
```

Perintah di atas akan menghasilkan file **UsersSeeder.php** yang berada di dalam direktori **database/seeds**.

Buka file **UsersSeeder.php**, masukkan beberapa baris code ke dalamnya :

```
database/seeds/UserSeeder.php  
  
<?php  
use Illuminate\Database\Seeder;  
use App\Role;
```

```

use App\User;

class UsersSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        //Membuat role admin
        $adminRole = new Role();
        $adminRole->name = "admin";
        $adminRole->display_name = "Admin";
        $adminRole->save();

        //Membuat role member
        $memberRole = new Role();
        $memberRole->name = "member";
        $memberRole->display_name = "Member";
        $memberRole->save();

        //Membuat sample admin
        $admin = new User();
        $admin->name = 'Administrator';
        $admin->email = 'admin@larakod.com';
        $admin->password = bcrypt('admin123');
        $admin->save();
        $admin->attachRole($adminRole);

        //Membuat sample user
        $member = new User();
        $member->name = "Member Larakod";
        $member->email = "member@larakod.com";
        $member->password = bcrypt('member123');
        $member->save();
        $member->attachRole($memberRole);
    }
}

```



`bcrypt()` adalah method untuk mengenkripsi *password* yang dibuat dengan jenis enkripsi `AES-256-CBC` dan `APP_KEY` sebagai Salt-nya

`attachRole()` adalah method untuk menambahkan Role

Masukkan `UsersSeeder` ke method `run` pada file `DatabaseSeeder.php`

database/seeds/DatabaseSeeder.php

```

<?php
use Illuminate\Database\Seeder;

```

```

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        // $this->call(UsersTableSeeder::class);
        $this->call(UsersSeeder::class);
    }
}

```

Jalankan perintah `db:seed`

```

php artisan db:seed
// seeding: UsersSeeder

```

Pada table `users`, `roles`, dan `role_user` akan terisi data yang sudah kita buat di file `UsersSeeder.php`

Konfigurasi Register

Kita akan membuat sedikit perubahan pada file `RegisterController.php` agar user baru yang mendaftar dari halaman register dapat secara otomatis menjadi level member.

Tambahkan *syntax* berikut

app/Http/Controllers/Auth/RegisterController.php

```

<?php

...
use App\Role;

class RegisterController extends Controller
{
    /**
     * -----
     * Register Controller
     * -----
     *
     * This controller handles the registration of new users as well as their
     * validation and creation. By default this controller uses a trait to
     * provide this functionality without requiring any additional code.
     */
}

```

```

/**
 * Create a new user instance after a valid registration.
 *
 * @param array $data
 * @return User
 */
protected function create(array $data)
{
    $user = User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => bcrypt($data['password']),
    ]);
    $memberRole = Role::where('name', 'member')->first();
    $user->attachRole($memberRole);
    return $user;
}
}

```

Membuat Route Group Admin

Buka file `routes/web.php`, tambahkan `Route::group()` :

routes/web.php

```

<?php

/*
|-----
| Web Routes
|-----
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

...
Route::group(['prefix' => 'admin', 'middleware' => ['auth']], function () {
    //
});

```



Route Group yang memiliki prefix admin akan diarahkan untuk Role admin.

Menginstall Package/Library

Bootstrap

Meskipun Laravel sudah menggunakan Bootstrap, kita akan mencoba untuk memasangnya secara manual. Dapatkan Bootstrap di [sini](http://getbootstrap.com/)⁴. Ekstrak zip hasil download, kemudian salin :

- `bootstrap.min.css` ke folder `public/css/`
- `bootstrap.min.js` ke folder `public/js/`
- `fonts/` ke folder `public/`

Buka `app.blade.php` yang terdapat di `resources/views/layouts/`, include file css dan js Bootstrap :

resources/views/layouts/app.blade.php

```
<!DOCTYPE html>
<html lang="{{ app()->getLocale() }}">
<head>
    ...
    <!-- Styles -->
    <link href="{{ asset('css/bootstrap.min.css') }}" rel="stylesheet">
</head>
<body>
    ...
    <!-- Scripts -->
    <script src="{{ asset('js/bootstrap.min.js') }}"></script>
</body>
</html>
```

Agar tampilan Bootstrap tidak terlalu mainstream, kita dapat menggunakan css dari [Bootswatch](https://bootswatch.com/)⁵. Saya memilih theme [Cosmo](https://bootswatch.com/cosmo/bootstrap.min.css)⁶. Download css-nya, dan masukkan ke dalam folder `public/css`. Jika muncul notifikasi, pilih saja *replace* untuk menimpa ulang nama file css yang sama.

⁴ <http://getbootstrap.com/getting-started/#download>

⁵ <https://bootswatch.com/>

⁶ <https://bootswatch.com/cosmo/bootstrap.min.css>

JQuery

Download jQuery di [sini](#)⁷. Simpan dan salin ke folder `public/js`. Jangan lupa load js-nya :

resources/views/layouts/app.blade.php

```
...
<!-- Scripts -->
<script src="{{ asset('js/jquery-3.2.1.min.js') }}"></script>
<script src="{{ asset('js/bootstrap.min.js') }}"></script>
...
```

Font Awesome

Kita akan menambahkan Font Awesome ke dalam Larakod. Download Font Awesome di halaman resmi Font Awesome yang saat ini sudah mencapai versi 4.7 atau download di [sini](#)⁸ kemudian ekstrak hasilnya. Salin (Copy) folder `font`, letakkan di dalam folder `public`. Salin file `font-awesome.min.css`, letakkan di folder `css`.

Setelah itu tambahkan di dalam `app.blade.php` script untuk me-load `font-awesome.min.css`

resources/views/layouts/app.blade.php

```
<!-- Styles -->
<link href="{{ asset('css/bootstrap.min.css') }}" rel="stylesheet">
<link href="{{ asset('css/font-awesome.min.css') }}" rel="stylesheet">
```

DataTables

Sebelumnya Anda harus memiliki DataTables terlebih dahulu. Jika belum memilikinya, silahkan download di [sini](#)⁹ atau mengunjungi halaman [download](#)¹⁰ di situs resminya.

⁷ <https://code.jquery.com/jquery-3.2.1.min.js>

⁸ <http://fontawesome.io/assets/font-awesome-4.7.0.zip>

⁹ <https://datatables.net/download/download>

¹⁰ <https://datatables.net/download>



Ekstrak file hasil download, buka folder media, di dalamnya terdapat folder css, images, dan js yang akan digunakan untuk DataTables

Ada empat file dan satu folder yang akan kita salin ke dalam folder `public/`

- `dataTables.bootstrap.min.css` salin ke dalam folder `public/css/`
- `dataTables.bootstrap.min.js` dan `jquery.dataTables.min.js` salin ke dalam folder `public/js/`

Buka file `app.blade.php` yang terletak di dalam folder `resources/views/layouts/` kemudian tambahkan script HTML untuk me-load file CSS dan JS.

resources/views/layouts/app.blade.php

```
<!-- Styles -->
<link href="{{ asset('css/bootstrap.min.css') }}" rel="stylesheet">
<link href="{{ assets('css/font-awesome.min.css') }}" rel="stylesheet">
<link href="{{ asset('css/dataTables.bootstrap.min.css') }}"
rel="stylesheet">

<!-- Scripts -->
<script src="{{ asset('js/jquery-3.2.1.min.js') }}"></script>
<script src="{{ asset('js/bootstrap.min.js') }}"></script>
<script src="{{ asset('js/jquery.dataTables.min.js') }}"></script>
<script src="{{ asset('js/dataTables.bootstrap.min.js') }}"></script>
```

Laravel DataTables

Setelah berhasil menginstall DataTables, selanjutnya kita akan meng-install Package [yajra/laravel-datatables](https://github.com/yajra/laravel-datatables)¹¹ dengan menggunakan Composer. Package ini memudahkan kita untuk menggunakan DataTables Server Side. DataTables Server Side merupakan cara untuk me-load isi data table database dengan lebih cepat dan ringan sehingga tidak membebani server saat menangani ribuan data untuk ditampilkan pada table.

Ketik pada command line :

```
composer require yajra/laravel-datatables-oracle:^7.5
//./composer.json has been updated
```

¹¹ <https://github.com/yajra/laravel-datatables>


```
//Loading composer repositories with package information
//Updating dependencies (including require-dev)
//Package operations: 1 install, 0 updates, 0 removals
- Installing yajra/laravel-datatables-oracle (v7.5.0): Downloading (100%)
//yajra/laravel-datatables-oracle suggests installing yajra/laravel-datatables-buttons
(Plugin for server-side exporting of dataTable.)
//yajra/laravel-datatables-oracle suggests installing yajra/laravel-datatables-html
(Plugin for server-side HTML builder of dataTable.)
//yajra/laravel-datatables-oracle suggests installing league/fractal (Transform your api
response into something you won't hate.)
//Writing lock file
//Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postUpdate
> php artisan optimize
//Generating optimized class loader
//The compiled services file has been removed.
```

Kemudian, buka file `app.php`, tambahkan pada array `providers`.

config/app.php

```
/*
|-----
| Autoloaded Service Providers
|-----
|
| The service providers listed here will be automatically loaded on the
| request to your application. Feel free to add your own services to
| this array to grant expanded functionality to your applications.
|
*/

'providers' => [
    ...

    /*
     * Package Service Providers...
     */
    Laravel\Tinker\TinkerServiceProvider::class,
    Laratrust\LaratrustServiceProvider::class,
    Yajra\Datatables\DatatablesServiceProvider::class,
],
```

Buka command line, jalankan perintah Artisan berikut untuk mem-publish file konfigurasi

```
php artisan vendor:publish --tag=datatables
//Copied File [\vendor\yajra\laravel-datatables-oracle\src\config\datatables.php]
To [\config\datatables.php]
Publishing complete.
```

Selanjutnya, kita install `yajra/laravel-datatables-html` yaitu plugin untuk HTML Builder DataTables laravel untuk Laravel 5.4+

```
composer require yajra/laravel-datatables-html:^1.0
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Installing yajra/laravel-datatables-html (v1.0.0): Loading from cache
Writing lock file
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postUpdate
> php artisan optimize
Generating optimized class loader
The compiled services file has been removed.
```

Jangan lupa untuk menambahkannya ke `providers`

config/app.php

```
/*
|-----
| Autoloaded Service Providers
|-----
|
| The service providers listed here will be automatically loaded on the
| request to your application. Feel free to add your own services to
| this array to grant expanded functionality to your applications.
|
*/

'providers' => [

    /*
     * Package Service Providers...
     */
    Laravel\Tinker\TinkerServiceProvider::class,
    Laratrust\LaratrustServiceProvider::class,
    Yajra\Datatables\DatatablesServiceProvider::class,
    Yajra\Datatables\HtmlServiceProvider::class,
],
```

Terakhir, publish asset

```
php artisan vendor:publish --tag=datatables-html
// Copied Directory [\vendor\yajra\laravel-datatables-html\src\resources\views] To
[\resources\views\vendor\datatables]
// Publishing complete.
```

Laravel Collective

Laravel Collective merupakan package yang membantu kita untuk membuat form dan html dengan lebih cepat dan mudah.

Instalasi Laravel Collective

Jalankan perintah Composer berikut ini untuk menambahkan *require*

`laravelcollective/html` pada `composer.json` :

```
composer require "laravelcollective/html":"^5.4.0"
//./composer.json has been updated
//Loading composer repositories with package information
//Updating dependencies (including require-dev)
//Package operations: 1 install, 0 updates, 0 removals
- Installing laravelcollective/html (v5.4.8): Loading from cache
//Writing lock file
//Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postUpdate
> php artisan optimize
//Generating optimized class loader
//The compiled services file has been removed.
```

Kemudian, tambahkan provider baru ke array `providers` dan `aliases` di `config/app.php` :

config/app.php

```
/*
|-----
| Autoloaded Service Providers
|-----
|
| The service providers listed here will be automatically loaded on the
| request to your application. Feel free to add your own services to
| this array to grant expanded functionality to your applications.
|
*/

'providers' => [

    ...

    /*
     * Package Service Providers...
     */
    Laravel\Tinker\TinkerServiceProvider::class,
    Laratrust\LaratrustServiceProvider::class,
    Yajra\DataTables\DataTablesServiceProvider::class,
```

```

        Yajra\DataTables\HtmlServiceProvider::class,
        Collective\Html\HtmlServiceProvider::class,

        ...

    ],

    /*
    /-----
    / Class Aliases
    /-----
    /
    / This array of class aliases will be registered when this application
    / is started. However, feel free to register as many as you wish as
    / the aliases are "lazy" loaded so they don't hinder performance.
    /
    */
    'aliases' => [

        ...
        'Laratrust' => Laratrust\LaratrustFacade::class,
        'Form' => Collective\Html\FormFacade::class,
        'Html' => Collective\Html\HtmlFacade::class,

    ],

```

Membuat Form Menggunakan Collective

Syntax dasar untuk membuat form HTML :

```

{!! Form::open(['url' => 'foo/bar']) !!}
//
{!! Form::close() !!}

```

Secara default menggunakan metode POST, jika ingin menggunakan metode lain selain POST, tambahkan isian array :

```

{!! Form::open(['url' => 'foo/bar', 'method' => 'put']) !!}
//
{!! Form::close() !!}

```

Syntax dasar Action Form yang menggunakan route dan controller :

```

// action menggunakan named route
{!! Form::open(['route' => 'route.name']) !!}
// named route dengan parameter
{!! Form::open(['route' => ['route.name', $user->id]]) !!}
//action menggunakan controller

```

```
{!! Form::open(['action' => 'Controller@method']) !!}  
// controller dengan parameter  
{!! Form::open(['action' => ['Controller@method', $user->id]]) !!}
```

Syntax dasar Form jika terdapat file upload :

```
{!! Form::open(['url' => 'foo/bar', 'files' => true]) !!}
```

Syntax dasar form input HTML :

1. Input Text :

```
{!! echo Form::text('username') !!}
```

2. Input Text dengan Value :

```
{!! echo Form::text('username', 'Mark Odo') !!}
```

3. Input Password :

```
{!! Form::password('password') !!}
```

4. Input Password dengan Class :

```
{!! Form::password('password', ['class' => 'awesome']) !!}
```

5. Input Email :

```
{!! Form::text('email') !!}
```

6. Checkboxes dengan checked :

```
{!! Form::checkbox('name', 'value', true) !!}
```

7. Radio buttons dengan checked :

```
{!! Form::radio('name', 'value', true) !!}
```

8. Input Drop-Down lists

```
{!! Form::select('gender', ['M' => 'Male', 'F' => 'Female'], 'M') !!}
```

9. Input Number :

```
{!! Form::number('name', 'value') !!}
```

10. Input Date

```
{!! Form::date('name', \Carbon\Carbon::now()) !!}
```

11. Input File image

```
{!! Form::file('image') !!}
```

12. Tombol Submit

```
{!! Form::submit('OK!') !!}
```

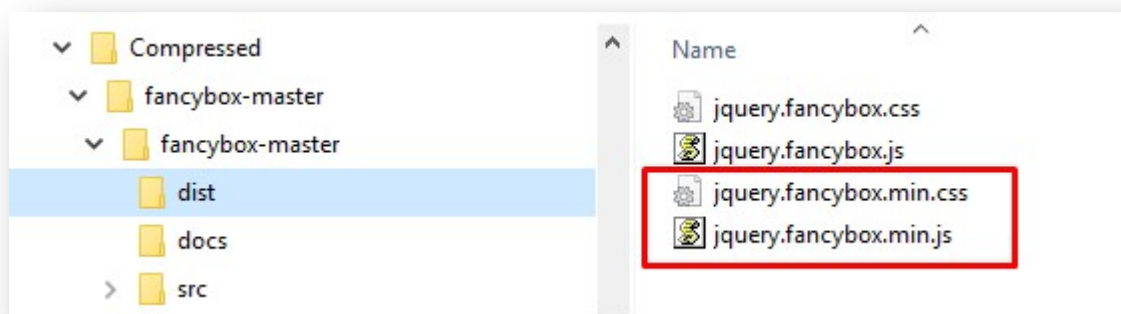
Silahkan baca [dokumentasi](#) ¹² lebih lengkap penggunaan Laravel Collective.

Fancybox

[Fancybox](#) ¹³ merupakan sebuah tool yang memberikan fungsi *zooming* dan *iframe* untuk gambar dan konten HTML. Klik di [sini](#) ¹⁴ untuk mendownload Fancybox.

Setelah Anda download, ekstrak `fancybox-master.zip`.

Copy `jquery.fancybox.min.css` dan `jquery.fancybox.min.js`, keduanya ada di dalam folder `dist` :



Gambar 3 Copy css dan js fancybox

¹² <https://laravelcollective.com/docs/5.4/html>

¹³ <http://fancyapps.com/fancybox/3/>

¹⁴ <https://github.com/fancyapps/fancybox/archive/master.zip>

Paste `jquery.fancybox.min.css` ke dalam `public/css`, dan `jquery.fancybox.min.js` ke dalam `public/js`

Kemudian panggil file css dan js fancybox ke dalam `app.blade.php`

resources/views/layouts/app.blade.php

```
<!-- Styles -->
<link href="{{ asset('css/app.css') }}" rel="stylesheet">
<link href="{{ asset('css/bootstrap.min.css') }}" rel="stylesheet">
<link href="{{ asset('css/font-awesome.min.css') }}" rel="stylesheet">
<link href="{{ asset('css/dataTables.bootstrap.min.css') }}" rel="stylesheet">
<link href="{{ asset('css/jquery.fancybox.min.css') }}" rel="stylesheet">

<!-- Scripts -->
<script src="{{ asset('js/jquery-3.2.1.min.js') }}"></script>
<script src="{{ asset('js/bootstrap.min.js') }}"></script>
<script src="{{ asset('js/jquery.dataTables.min.js') }}"></script>
<script src="{{ asset('js/dataTables.bootstrap.min.js') }}"></script>
<script src="{{ asset('js/jquery.fancybox.min.js') }}"></script>
```

Jika Anda ingin tahu lebih banyak tentang fancybox, Anda bisa baca [dokumentasi fancybox](#)¹⁵.

Responsive File Manager

[Responsive File Manager](#)¹⁶ merupakan plugin file manager yang digunakan untuk mengelola file yang diupload oleh user. Klik di [sini](#)¹⁷ untuk mengunduh Responsive File Manager.

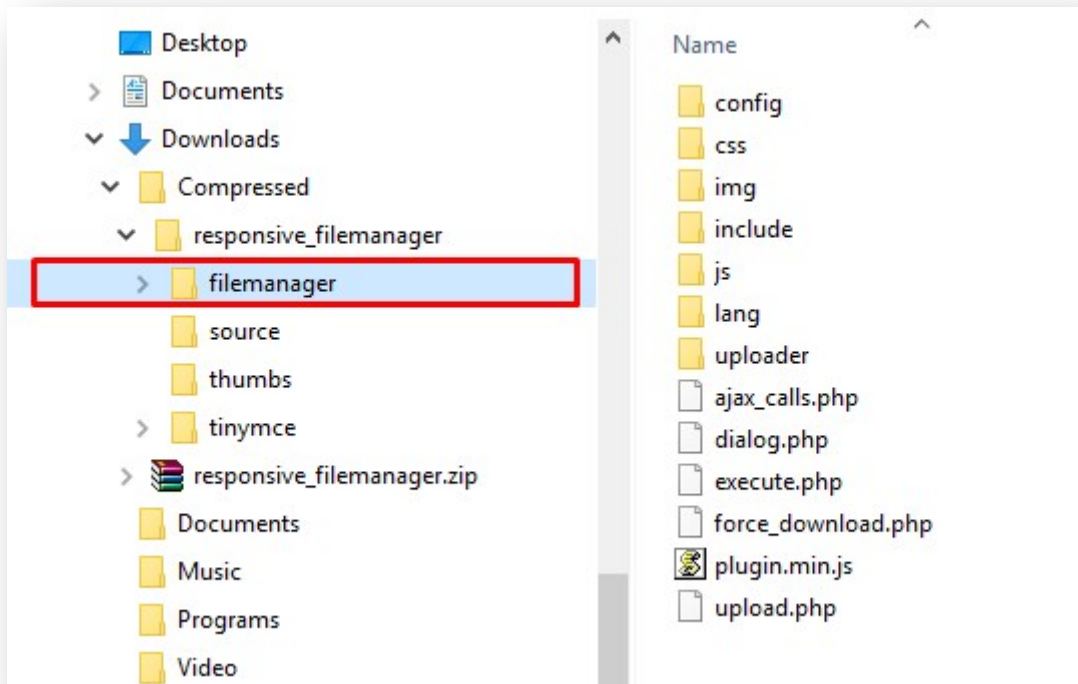
Setelah diekstrak, copy folder filemanager :

¹⁵ <http://fancyapps.com/fancybox/3/docs/>

¹⁶ <http://www.responsivefilemanager.com/>

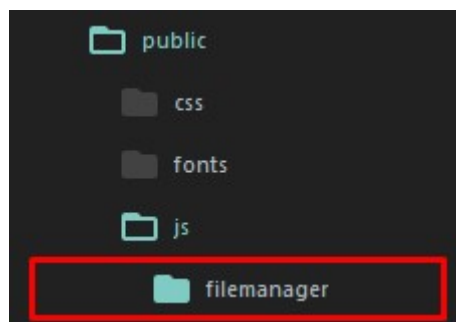
¹⁷

https://github.com/trippo/ResponsiveFilemanager/releases/download/v9.11.3/responsive_filemanager.zip



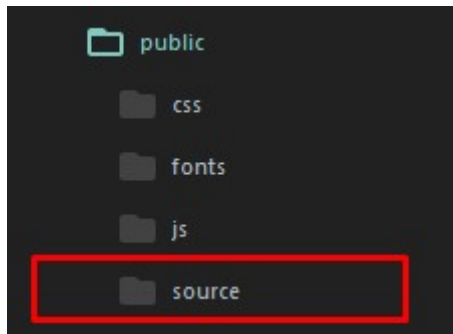
Gambar 4 copy folder filemanager

Masukkan dalam `public/js` :



Gambar 5 Paste filemanager ke dalam public/js

Buatlah folder baru di dalam folder `public` untuk menampung file upload, kita akan namakan `source`.



Gambar 6 Folder Source

Kemudian, buka file `config.php` dalam folder `config`, ganti isian berikut ini :

Public/js/filemanager/config/config.php

```
date_default_timezone_set('Europe/Rome Asia/Jakarta');  
'current_path' => '../source/ ../../source/',  
'thumbs_base_path' => '../thumbs/ ../../thumbs/',  
'default_language' => "en-EN-id",
```

`public/source` merupakan direktori tempat dimana file asli download berada, sementara `thumbs` merupakan direktori yang berisi file gambar thumbnail yang telah diperkecil.

Hapus file `include.js` yang terdapat di `public/js/filemanager/js` kemudian download `include.js` versi non minify, Anda bisa mengikuti tautan yang diberikan berikut ini untuk mendapatkan file tersebut.

```
https://raw.githubusercontent.com/trippo/ResponsiveFilemanager/master/resources/as  
sets/js/include.js
```

Kemudian Anda save, dan masukkan ke dalam `public/js/filemanager/js`.

Buka file `include.js` tersebut, cari dan hapus function `close_window` dan tambahkan function `close_window` baru :

Public/js/filemanager/js/include.js

```
function close_window()  
{  
    parent.jQuery.fancybox.getInstance().close();  
}
```

```

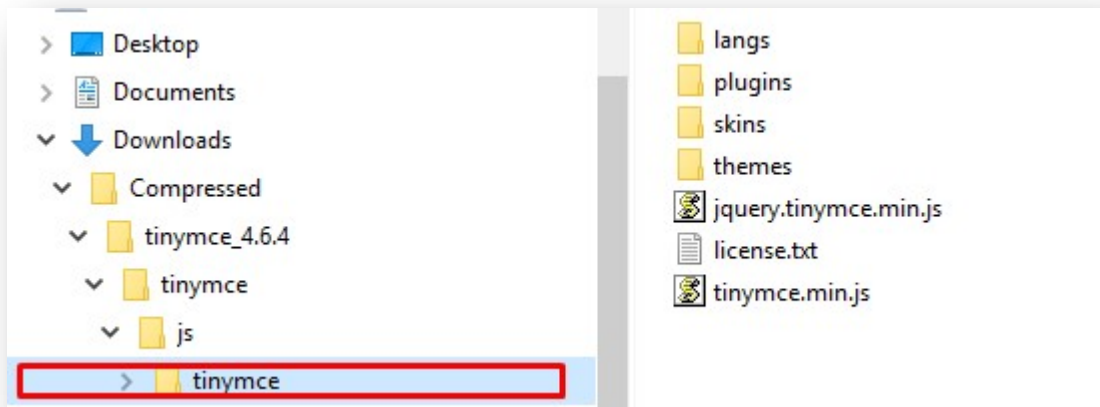
function close_window()
{
    if (jQuery('#popup').val() == 1)
    {
        window.close();
    }
    else
    {
        if (typeof parent.jQuery('.modal').modal == "function"){
            parent.jQuery('.modal').modal('hide');
        }
        if (typeof parent.jQuery != "undefined" && parent.jQuery)
        {
            if(typeof parent.jQuery.fancybox == 'function'){
                parent.jQuery.fancybox.close();
            }
        }
    }
    else
    {
        if(typeof parent.$.fancybox == 'function'){
            parent.$.fancybox.close();
        }
    }
}

function close_window()
{
    if (jQuery('#popup').val() == 1)
    {
        window.close();
    }
    else
    {
        if (typeof
parent.jQuery(".modal:has(iframe[src*=filemanager])").modal == "function"){
            parent.jQuery(".modal:has(iframe[src*=filemanager])").modal("hide");
        }
        if (typeof parent.jQuery != "undefined" && parent.jQuery)
        {
            if(typeof parent.jQuery.fancybox == 'function'){
                parent.jQuery.fancybox.close();
            }
        }
    }
    else
    {
        if(typeof parent.$.fancybox == 'function'){
            parent.$.fancybox.close();
        }
    }
}

```

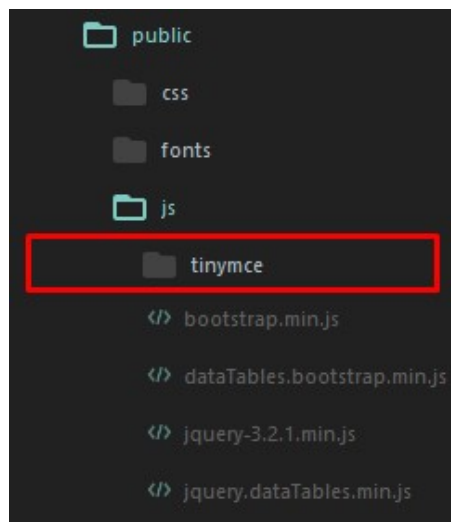
TinyMCE

[TinyMCE](https://www.tinymce.com)¹⁸ merupakan plugin yang berfungsi sebagai teks editor pada form di halaman web. Kita bisa mendapatkannya dengan cara mengunduhnya di situs resminya. Klik di [sini](http://download.ephox.com/tinymce/community/tinymce_4.6.4.zip)¹⁹ untuk mengunduh TinyMCE untuk tipe Community. Setelah kita dapatkan `tinymce_4.6.4.zip`, ekstrak, kemudian copy folder `tinymce` yang berada di dalam `tinymce_4.6.4/js`.



Gambar 7 copy folder tinymce

Kemudian, paste ke dalam folder `public/js`.



Gambar 8 tinymce dalam folder js

¹⁸ <https://www.tinymce.com>

¹⁹ http://download.ephox.com/tinymce/community/tinymce_4.6.4.zip

Tambahkan baris berikut untuk memanggil **js** di **app.blade.php** :

resources/views/layouts/app.blade.php

```
<!-- Scripts -->
<script src="{{ asset('js/jquery-3.2.1.min.js') }}"></script>
<script src="{{ asset('js/bootstrap.min.js') }}"></script>
<script src="{{ asset('js/jquery.dataTables.min.js') }}"></script>
<script src="{{ asset('js/dataTables.bootstrap.min.js') }}"></script>
<script src="{{ asset('js/jquery.fancybox.min.js') }}"></script>
<script src="{{ asset('js/tinymce/jquery.tinymce.min.js') }}"></script>
<script src="{{ asset('js/tinymce/tinymce.min.js') }}"></script>
```

Membuat Halaman Baru

Kita akan membuat halaman baru di dalam halaman Admin. Halaman yang akan saya buat adalah halaman view untuk menampilkan data pada table database. Saya akan buat table **posts** yang berisi data tentang artikel berita atau blog.

Merancang Table

Kita akan membuat struktur *table* **posts** sebagai berikut

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO	PRI	NULL	auto_inc
title	varchar(255)	NO		NULL	
seotitle	varchar(255)	NO		NULL	
author	varchar(191)	NO		NULL	
content	text	NO		NULL	
image	varchar(255)	NO		Noimage.jpg	
hits	int(11)	NO		0	
headline	enum('y', 'n')	NO		N	
active	enum('y', 'n')	NO		Y	
status	enum('publish', 'draft')	NO		publish	
created_at	timestamp	YES		NULL	
updated_at	timestamp	YES		NULL	

Models

Pengenalan

Model dalam Laravel masuk dalam pembahasan ORM (Object-Relational Mapping) Eloquent. Eloquent sendiri merupakan fitur yang sangat berguna dan mudah digunakan saat bekerja dengan database, seperti melakukan create, read, update, dan delete. Setiap table database memiliki model yang saling terhubung dan berinteraksi.

Membuat Model Table Posts

Meski pun Laravel tidak menyediaka *folder* `models`, kita dapat bebas menentukan sendiri dimana kita akan meletakannya. Secara *default file model* ditempatkan di dalam *folder* `app`.

Model Post akan digunakan untuk memproses informasi dari dan ke *table* posts di *database*.

Cara untuk membuat model adalah menggunakan perintah Artisan `make:model`

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

Jika kita ingin sekaligus membuat **database migration** pada saat membuat **model**, kita dapat menambahkan `--migration` atau `-m`

```
php artisan make:model Post -migration
// atau
php artisan make:model Post -m
```

Nama model diawali dengan huruf kapital. Nama yang kita buat akan menjadi nama class dan file model. Model `Post` adalah model dari table `posts`.

Nama table database bersifat plural (berakhiran s/es) dan nama model bersifat singular. Contohnya untuk membuat model dari tabel posts, maka nama model yang digunakan adalah Post.

Sebenarnya nama model tidak harus mengikuti nama table, bisa saja kita membuat nama model-nya adalah `Post`, sementara nama table-nya adalah

`artikel`, namun kita perlu menambahkan `protected $table = 'nama_table';`, agar antara Model dan table database saling terhubung.

app/Post.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    protected $table = 'artikel';
}
```

Supaya kita bekerja dengan lebih efektif, ikuti saja aturan penamaan yang sudah dikenal, hal ini akan membuat kita lebih konsisten, teratur, dan *friendly*. Jadi kita gunakan saja untuk table `posts`, memakai model dengan nama `Post`.

app/Post.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    //
}
```

Ada dua hal setidaknya yang harus kita perhatikan, yaitu masalah `primary key` dan `timestamps`. Default-nya Laravel menjadikan setiap table memiliki kolom `id` yang dijadikan sebagai `primary key`, sehingga jika kita memiliki kolom `primary key` tapi bukan dengan nama `id`, misalnya `post_id`, maka kita harus menentukan sendiri secara manual properti `$primaryKey`.

app/Post.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;
```

```
class Post extends Model
{
    protected $primaryKey = 'post_id';
}
```

Laravel juga secara default memperkirakan table yang kita buat terdapat kolom `created_at` dan `updated_at`. Jika ternyata table kita tidak memiliki kedua kolom tersebut, kita harus men-set **FALSE** properti `$timestamps`.

app/Post.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    protected $timestamps = false;
}
```

Kita juga dapat menentukan sendiri nama kolom table untuk `created_at` dan `updated_at`, jika kita ingin menamai kolom dengan nama yang berbeda.

app/Post.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    const CREATED_AT = 'tgl_buat';
    const UPDATED_AT = 'tgl_sunting';
}
```

Setelah kita berhasil membuat model Post, tambahkan properti `$fillable` untuk mengatur kolom mana saja yang akan diisi pada saat insert data.

app/Post.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
```



```
{  
    protected $fillable = ['title', 'seotitle', 'author', 'content',  
        'image', 'hits', 'headline', 'active', 'status'];  
}
```

Hati-hati saat mengatur properti `$fillable`, karena jika kolom database diatur NOT NULL sementara kita tidak memasukkannya ke dalam `$fillable`, maka akan terjadi error.

Migrations

Pengenalan

Migrations seperti *version control* untuk *database*. Migrations sangat bermanfaat bagi para developer yang bekerja secara tim dimana mereka dengan mudah dapat memodifikasi skema table pada database. Anggota tim mereka hanya cukup melakukan migrate untuk menerima hasil perubahan table.

Membuat migrations dilakukan dengan perintah Artisan `make:migration`. Aturan penamaan agar mudah dikenali diawali dengan kata **create**, **nama table**, dan diakhiri dengan kata **table**. Ditulis dengan huruf kecil, dan masing-masing kata dipisahkan dengan tanda underscore.

Contoh :

```
php artisan make:migration create_test_table
```

Hasil *generate* migrations dapat dilihat di dalam `database/migrations`. Contoh hasil file migration yang dibuat `2017_06_01_011420_create_test_table.php`. Perintah Artisan `make:migration` juga dapat ditambahkan dengan opsi `--create` atau `--table`. Opsi `--create` akan memudahkan dalam pembuatan struktur *table database* atau skema *database* karena akan dibuatkan skema buildernya.

```
php artisan make:migration create_test_table --create=tests
```

Opsi `--table` sendiri adalah opsi yang memudahkan kita untuk melakukan *modifikasi* table seperti menambahkan kolom, menghapus kolom, mengubah kolom, dll.

```
php artisan make:migration add_votes_to_test_table --table=tests
```

Keterangan :

- `make:migration` adalah perintah Artisan untuk membuat **migration**
- `create_tests_table` adalah **nama migration**-nya
- `add_vote_to_tests_table` adalah **nama migration**
- `create=tests` adalah **nama untuk *table*** yang akan dibuat
- `table=tests` adalah **nama *table*** yang akan dilakukan modifikasi

Isi file migration default :

database/migrations/2017_06_01_011420_create_test_table.php

```
<?php

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

class CreateTestTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        //
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        //
    }
}
```

Isi file migration dengan opsi `--create` :

database/migrations/2017_06_01_011420_create_test_table.php

```
<?php

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

```

use Illuminate\Database\Migrations\Migration;

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

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

```

File migration dengan opsi `--table` :

database/migrations/2017_06_01_020554_add_votes_to_test_table.php

```

<?php

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

class AddVotesToTestTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('tests', function (Blueprint $table) {
            $table->string('name')->after('id');
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
}

```

```

    */
    public function down()
    {
        Schema::table('tests', function (Blueprint $table) {
            //
        });
    }
}

```

Method `up()` berfungsi untuk membuat table, sementara *method* `down()` digunakan untuk menghapus table. Pada file migration yang dibuat dengan opsi `--create` sudah terdapat schema pembuatan table tinggal ditambahkan untuk membuat kolom baru. Kita juga dapat menghapus table yang sudah dibuat terakhir kali dengan perintah **Artisan** `migrate:rollback`.

```
php artisan migrate:rollback
```

Jika ingin mengembalikan table yang kita hapus, gunakan perintah **Artisan** `migrate`.



Jika kita tanpa sengaja menghapus file **migration**, kita harus melakukan perintah `composer dump-autoload` agar dapat membuat file **migration** dengan nama table yang sama jika table di database juga terhapus.

Membuat Migration Posts

Saya akan membuat migration Posts dengan opsi `--create` agar dapat menghasilkan skema pembuatan table. Jika pada saat Anda membuat model menyertakan pembuatan migration `--migration` maka Anda bisa melewati perintah `make:migration` dan membuka file migration posts yang sudah dibuat.

```

php artisan make:migration create_posts_table --create=posts
// Created Migration: 2017_06_01_005044_create_posts_table

```

Buka file `2017_06_01_005044_create_posts_table.php`.

Tambahkan di dalam *method* `up` :

Database/migrations/2017_06_01_005044_create_posts_table.php

```

<?php

use Illuminate\Support\Facades\Schema;

```

```

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

class CreatePostsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->increments('id');
            $table->string('title', 255);
            $table->string('seotitle', 255);
            $table->string('author');
            $table->text('content');
            $table->string('image', 255)->default('noimage.jpg');
            $table->integer('hits')->default('0');
            $table->enum('headline', ['Y', 'N'])->default('N');
            $table->enum('active', ['Y', 'N'])->default('Y');
            $table->enum('status', ['publish', 'draft'])->
            >default('publish');
            $table->timestamps();
        });
    }
    ...
}

```

Setelah itu jalankan perintah Artisan `migrate` untuk mengeksekusi file migration yang dibuat :

```

php artisan migrate

// Migrating: 2017_06_01_005044_create_posts_table
// Migrated: 2017_06_01_005044_create_posts_table

```

Seeding

Pengenalan

Apa itu Seeding ?

Seeding adalah cara yang mudah untuk melakukan insert data ke database dengan menggunakan class **Seed**. Semua class **Seed** ini disimpan di dalam direktori `database/seeds`. Class Seed sendiri bisa kita namai apa saja, nama class akan menjadi nama file-nya. Biasanya dan ini diikuti oleh banyak orang, class Seed diberi nama dengan 2 atau 3 suku kata yang diawali masing-masing dengan

huruf kapital yang digabungkan menjadi satu. Suku kata pertama adalah nama table yang ingin kita buat class Seeder-nya contohnya table **users**, suku kata kedua adalah **Table**, dan yang terakhir adalah **Seeder**, maka jadinya: **UsersTableSeeder**. Ada satu buah class yang sudah ada di dalam direktori **database/seeds**. Class tersebut adalah **DatabaseSeeder**, class ini merupakan class default yang sudah ditetapkan, dimana dengan class ini kita dapat menggunakan method **call** yang memanggil semua class-class Seed lainnya sehingga dapat dieksekusi bersama-sama. Kita juga dapat mengatur urutannya table mana dulu yang mau kita isi. Menulis statement insert data ke database bisa dilakukan langsung di method **run** class **DatabaseSeeder**.

Membuat Seeders

Bagaimana membuat class **Seed** ? kita buat dengan perintah Artisan **make:seeder**.

Contoh :

```
php artisan make:seeder TestsTableSeeder  
  
// Seeder created successfully
```

Setelah perintah ini dijalankan, akan muncul **TestsTableSeeder.php** di dalam direktori **database/seeds**.

database/seeds/TestsTableSeeder.php

```
<?php  
  
use Illuminate\Database\Seeder;  
  
class TestsTableSeeder extends Seeder  
{  
    /**  
     * Run the database seeds.  
     *  
     * @return void  
     */  
    public function run()  
    {  
        //  
    }  
}
```

Class **TestsTableSeeder** hanya punya satu method **run**. Membuat statemen untuk insert data dilakukan di method **run**. Kita dapat menggunakan **query builder**,

atau `Eloquent model factories`. Kita bisa menulis query pada `TestsTableSeeder` atau secara langsung ke `DatabaseSeeder`. Namun, lebih rekomended jika dilakukan pada class seeder-nya masing-masing. Menulis query pada class `DatabaseSeeder` hanya untuk testing atau keperluan lain yang bersifat praktis.

Menggunakan Query Builder

Kita akan mencoba memasukkan data ke table tests dengan menggunakan seeding melalui class `DatabaseSeeder`. Statement insert data yang akan ditulis adalah dengan menggunakan query builder.

database/seeds/DatabaseSeeder.php

```
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        DB::table('tests')->insert(['name'=>'Andi Khan']);
    }
}
```

Menggunakan Model Factories

Model factories praktis digunakan untuk memasukkan data tanpa perlu satu-satu menentukan atribut pada setiap model seed. Kita dapat memasukkan lebih dari satu record dan membuat data secara otomatis. Model factories digunakan untuk tujuan testing atau demo dengan data-data sample.

Contoh berikut ini adalah membuat tiga record ke table tests :

Pertama, buka model `Test.php`, tambahkan script berikut :

app/Test.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;
```

```
class Test extends Model
{
    protected $fillable = [
        'name'
    ];
}
```

Selanjutnya buka `ModelFactory.php` yang berada di dalam direktori `database/factories`, tambahkan script berikut di dalamnya :

database/factories/ModelFactory.php

```
<?php

/*
|-----
| Model Factories
|-----
|
| Here you may define all of your model factories. Model factories give
| you a convenient way to create models for testing and seeding your
| database. Just tell the factory how a default model should look.
|
*/

...

$factory->define(App\Test::class, function (Faker\Generator $faker) {
    return [
        'name' => $faker->name
    ];
});
```

Setelah itu buka `DatabaseSeeder.php`, masukkan script berikut :

database/seeds/DatabaseSeeder.php

```
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        factory(App\Test::class, 3)->create();
    }
}
```


Setelah dieksekusi, hasilnya akan masuk tiga record dalam table `tests`

Menggunakan Mass Assignment

Kita bisa menggunakan mass assignment dalam insert data di seeder. Sebelumnya kita menggunakan *query builder* `DB::table('tests')->insert();` tapi dengan mass assignment kita menggunakan `Test::create();`, mirip dengan query yang digunakan dalam *model factory*.

Buka `TestsTableSeeder.php`, kali ini kita akan menulis query-nya di class `TestTableSeeder` yang kemudian dipanggil ke class `DatabaseSeeder` dengan method `call`. hapus semua baris *syntax* di dalam method `run()`, kemudian masukkan *syntax* berikut:

database/seeds/TestsTableSeeder.php

```
<?php
use Illuminate\Database\Seeder;
use App\Test;

class TestsTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        Test::create(['name'=>'Mike Jacobus']);
    }
}
```

Setelah itu buka `DatabaseSeeder.php`, panggil class `TestsTableSeeder`.

database/seeds/DatabaseSeeder.php

```
<?php
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
```

```

    {
        $this->call(TestsTableSeeder);
    }
}

```

Jalankan `db:seed` :

```

php artisan db:seed

// Seeding: TestsTableSeeder

```

Memanggil Class Seeder yang lain

Class `DatabaseSeeder` dapat menjalankan class seed yang lain dengan cara memanggil class-class tersebut secara berurutan jika ingin melakukan insert data multi table atau hanya satu table. Kita cukup menambahkan method `call` :

database/seeds/DatabaseSeeder.php

```

<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $this->call(UsersSeeder::class);
        $this->call(PostsTableSeeder::class);
    }
}

```

Menjalankan Seeders

Setelah kita membuat class seeder, jalankan dengan perintah Artisan `db:seed` untuk memasukkan data ke database. Secara defaultnya, perintah `db:seed` menjalankan class `DatabaseSeeder`, yang mana juga digunakan untuk memanggil class-class lainnya untuk dijalankan bersama.

```

php artisan db:seed

```

Jika kita tidak ingin menjalankan class `DatabaseSeeder`, kita bisa menambahkan opsi `--class` untuk memilih satu class tertentu untuk dijalankan.

```

php artisan db:seed --class=TestsTableSeeder

```

Selain dengan perintah `db:seed`, kita juga dapat menggunakan perintah `migrate:refresh` dengan opsi `--seed`, perintah ini akan melakukan rollback dan menjalankan kembali semua data migration.

```
php artisan migrate:refresh --seed
```

Membuat Seeder Posts

Setelah mengetahui sekilas tentang seeder, saatnya kita membuat seeder Posts. Jalankan perintah Artisan berikut ini :

```
php artisan make:seeder PostsTableSeeder  
  
// Seeder created successfully
```

Buka dan modifikasi `PostsTableSeeder.php`, kita akan menggunakan query builder untuk insert datanya :

database/seeds/PostsTableSeeder.php

```
<?php  
  
use Illuminate\Database\Seeder;  
  
class PostsTableSeeder extends Seeder  
{  
    /**  
     * Run the database seeds.  
     *  
     * @return void  
     */  
    public function run()  
    {  
        // masukkan data ke database  
        DB::table('posts')->insert([  
            'title' => 'Panduan Memulai Laravel',  
            'seotitle' => 'panduan-memulai-laravel',  
            'author' => 'Sigerweb',  
            'content' => 'Lorem ipsum excepteur aliquip ad nulla amet dui consectetur  
eu ullamco aliquip aute dolor est adipisicing ut consectetur voluptate ut tempor  
ex velit cupidatat id laborum occaecat sunt magna ullamco id fugiat cillum elit.',  
            'image' => 'laravel.jpg',  
            'hits' => 100  
        ]);  
    }  
}
```

Setelah itu buka class `DatabaseSeeder` , panggil class `PostsTableSeeder`:

```
<?php
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $this->call(PostsTableSeeder::class);
    }
}
```

Terakhir, jalankan dengan command prompt :

```
php artisan db:seed
// Seeding: PostsTableSeeder
```

Lihat pada table posts akan terlihat satu data baru yang ditambahkan sesuai dengan apa yang kita masukkan ke dalam class `PostsTableSeeder`. Tanpa `PostsTableSeeder`, kita bisa menuliskannya secara langsung ke `DatabaseSeeder`, atau alternatif lain tanpa `DatabaseSeeder` kita bisa langsung menjalankan `PostsTableSeeder` dengan opsi `--class=PostsTableSeeder` pada perintah `db:seed`.

Controllers

Pengenalan

Controllers adalah class yang mengatur penanganan berbagai request dan mengelompokkannya menjadi satu class. Controllers tersimpan di `app/Http/Controllers`.

Membuat PostController

Kita akan membuat `PostController` yang akan menangani berbagai request yang terkait dengan table `post`.

Sebelum membuat Controller ada yang harus kita perhatikan terdapat perbedaan ketika perintah `make:controller` dijalankan tanpa opsi `--resource` dengan

memakai opsi `--resource`. Jika kita ingin menangani proses create, read, update, dan delete, agar lebih mudah gunakan opsi `--resource`.

Membuat `PostController` default, tanpa opsi `--resource` :

```
php artisan make:controller PostController
// controller created successfully
```

Hasil dari perintah di atas :

app/Http/Controllers/PostController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PostController extends Controller
{
    //
}
```

Membuat `PostController` dengan opsi `--resource`

```
php artisan make:controller PostController --resource
// controller created successfully
```

Hasil dari perintah di atas :

app/Http/Controllers/PostController.php

```
<?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)
        {
            //
        }
    }
}

```

Setelah mengetahui perbedaannya, kita akan membuat `PostController` menggunakan perintah Artisan `make:controller` dengan opsi `--resource`.

Routing

Pengenalan

Routing merupakan proses yang mengatur request user terhadap suatu sistem kemudian menampilkan hasil dari proses tersebut. Routing bekerjasama dengan controller dan view dalam setiap proses yang diinginkan user.

Membuat Routing Post

Buka kembali `web.php`, sebelumnya kita sudah membuat `Route::group`, sekarang tambahkan ke dalamnya, `Route::resource()` :

routes/web.php

```

<?php

/*
|-----
| Web Routes
|-----
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

...

Route::group(['prefix' => 'admin', 'middleware' => ['auth']], function () {
    Route::resource('posts', 'PostController');
});

```

`Route::resource` sangat bermanfaat untuk membuat routing secara otomatis tanpa menuliskannya secara manual satu persatu untuk masing-masing method create, read, update, dan delete. Gunakan perintah Artisan `route:list` untuk melihat list route yang tersedia.

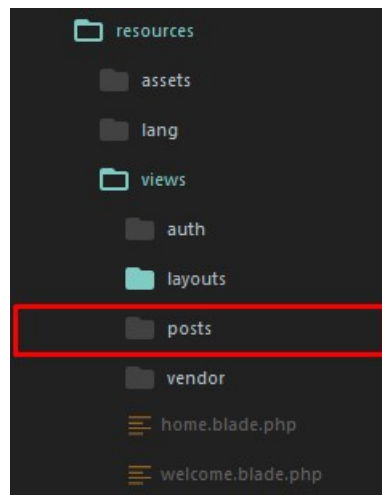
Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/		Closure	web
	POST	admin/posts	posts.store	App\Http\Controllers\PostController@store	web,auth
	GET HEAD	admin/posts	posts.index	App\Http\Controllers\PostController@index	web,auth
	GET HEAD	admin/posts/create	posts.create	App\Http\Controllers\PostController@create	web,auth
	DELETE	admin/posts/{post}	posts.destroy	App\Http\Controllers\PostController@destroy	web,auth
	PUT PATCH	admin/posts/{post}	posts.update	App\Http\Controllers\PostController@update	web,auth
	GET HEAD	admin/posts/{post}	posts.show	App\Http\Controllers\PostController@show	web,auth
	GET HEAD	admin/posts/{post}/edit	posts.edit	App\Http\Controllers\PostController@edit	web,auth
	GET HEAD	api/user		Closure	api,auth:api
	GET HEAD	home	home	App\Http\Controllers\HomeController@index	web,auth
	POST	login		App\Http\Controllers\Auth\LoginController@login	web,guest
	GET HEAD	login	login	App\Http\Controllers\Auth\LoginController@showLoginForm	web,guest
	POST	logout	logout	App\Http\Controllers\Auth\LoginController@logout	web
	POST	password/email	password.email	App\Http\Controllers\Auth\ForgotPasswordController@sendResetLinkEmail	web,guest
	GET HEAD	password/reset	password.request	App\Http\Controllers\Auth\ForgotPasswordController@showLinkRequestForm	web,guest
	POST	password/reset		App\Http\Controllers\Auth\ResetPasswordController@reset	web,guest
	GET HEAD	password/reset/{token}	password.reset	App\Http\Controllers\Auth\ResetPasswordController@showResetForm	web,guest
	POST	register		App\Http\Controllers\Auth\RegisterController@register	web,guest
	GET HEAD	register	register	App\Http\Controllers\Auth\RegisterController@showRegistrationForm	web,guest
	GET HEAD	test		Closure	web

Gambar 9 List Route pada Posts

Setelah kita menambahkan `Route:resource` dalam list route muncul tujuh route yang bisa kita gunakan untuk merouting halaman Post, mulai dari index sampai dengan delete.

Menyiapkan Folder

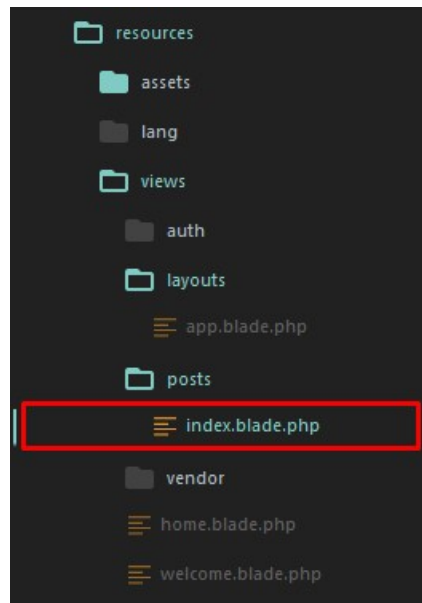
Kita akan membuat folder `posts` di dalam `resources/views` sebagai wadah untuk file-file yang terkait dengan `Posts`.



Gambar 10 folder post

Membuat File Index Post

Buatlah file `index.blade.php` di dalam folder `posts`. File `index.blade.php` digunakan untuk menampilkan view `posts`.



Gambar 11 `index.blade.php`

Masukkan dalam `index.blade.php`, syntax berikut :

resources/views/posts/index.blade.php

```
@extends('layouts.app')

@section('content')
<div class="container">
  <div class="row">
    <div class="col-md-12">
      <ul class="breadcrumb">
        <li><a href="{{ url('/home') }}">Dashboard</a></li>
        <li class="active">Posts</li>
      </ul>

      <div class="panel panel-default">
        <div class="panel-heading">
          <h2 class="panel-title">Posts <span class="pull-right"><a
class="btn btn-xs btn-default" href="{{ route('posts.create')
}}">Tambah</a></span>
        </div></h2>

        <div class="panel-body">

      </div>
    </div>
  </div>
</div>
```

```
</div>
</div>
@endsection
```

Agar dapat ditampilkan, buka `PostController.php`, tambahkan `return`
`view('posts.index');`.

app/Http/Controller/PostController.php

```
<?php

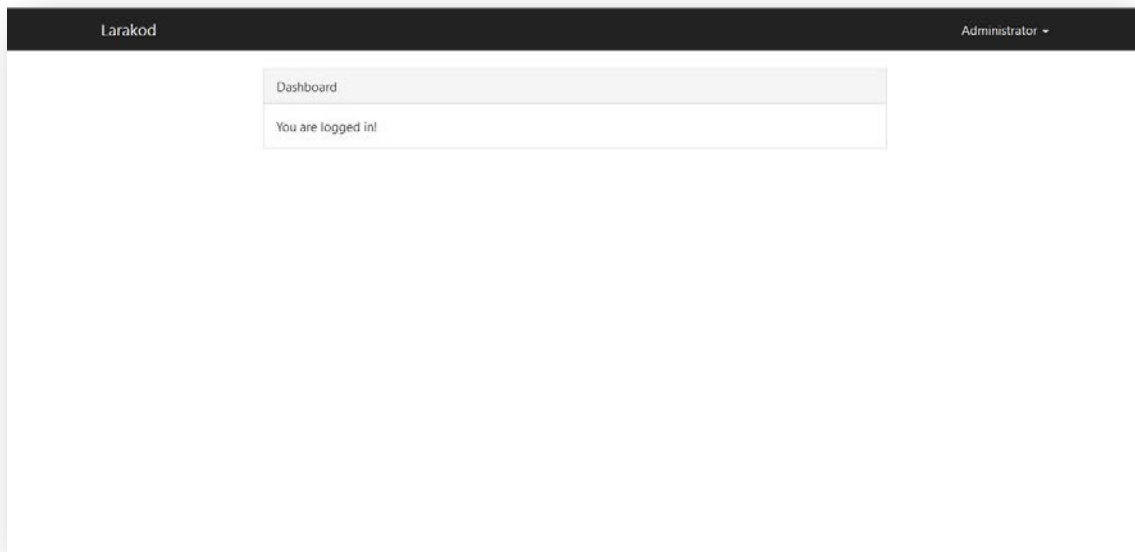
namespace App\Http\Controllers;

use Illuminate\Http\Request;

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

Menyiapkan Interface

Jalankan `localhost:8000`. Klik login, masukkan username dan password sebagai Admin. Jika username dan password benar, kita akan masuk ke halaman Dashboard Admin.



Gambar 12 Tampilan halaman Dashboard

Menambahkan link menu Dashboard dan Posts

app/Http/Controller/PostController.php

```
<?php
<!DOCTYPE html>
<html lang="{{ app()->getLocale() }}">
<head>
...
</head>
<body>
    <div id="app">
        <nav class="navbar navbar-default navbar-static-top">
            <div class="container">
                <div class="navbar-header">

                    <!-- Collapsed Hamburger -->
                    ...
                    <!-- Branding Image -->
                    <a class="navbar-brand" href="{{ url('/') }}">
                        {{ config('app.name', 'Laravel') }}
                    </a>
                </div>

                <div class="collapse navbar-collapse" id="app-navbar-collapse">
                    <!-- Left Side Of Navbar -->
                    <ul class="nav navbar-nav">
                        @if (Auth::check())
                            <li><a href="{{ url('/home') }}">Dashboard</a></li>
                            <li><a href="{{ route('posts.index') }}">Post</a></li>
                        @endif
                    </ul>

                    <!-- Right Side Of Navbar -->
                    ...
                </div>
            </div>
        </nav>
```

```
@yield('content')
</div>
...
</body>
</html>
```



Method `check()` akan menghasilkan nilai `true` jika `user` sudah `login` sehingga `link` menu akan terlihat atau tampil jika `user` sudah `login`. Berbeda dengan method `guest()` dimana akan menghasilkan nilai `true` jika `user` belum `login`.

Secara default, Laravel menggunakan `url()` dimana isi parameternya diambil dari `URI`, kali ini kita akan mencoba menggunakan `route()`, dimana isi parameternya diambil dari `Name`.

Anda bisa melihat berbagai nama route yang ada dengan menggunakan perintah:

```
php artisan route:list
```

Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/		Closure	web
	POST	admin/posts	posts.store	App\Http\Controllers\PostController@store	web,auth
	GET HEAD	admin/posts	posts.index	App\Http\Controllers\PostController@index	web,auth
	GET HEAD	admin/posts/create	posts.create	App\Http\Controllers\PostController@create	web,auth
	DELETE	admin/posts/{post}	posts.destroy	App\Http\Controllers\PostController@destroy	web,auth
	PUT PATCH	admin/posts/{post}	posts.update	App\Http\Controllers\PostController@update	web,auth
	GET HEAD	admin/posts/{post}	posts.show	App\Http\Controllers\PostController@show	web,auth
	GET HEAD	admin/posts/{post}/edit	posts.edit	App\Http\Controllers\PostController@edit	web,auth
	GET HEAD	api/user		Closure	api,auth:api
	GET HEAD	home	home	App\Http\Controllers\HomeController@index	web,auth
	POST	login		App\Http\Controllers\Auth\LoginController@login	web,guest
	GET HEAD	login	login	App\Http\Controllers\Auth\LoginController@showLoginForm	web,guest
	POST	logout	logout	App\Http\Controllers\Auth\LoginController@logout	web
	POST	password/email	password.email	App\Http\Controllers\Auth\ForgotPasswordController@sendResetLinkEmail	web,guest
	GET HEAD	password/reset	password.request	App\Http\Controllers\Auth\ForgotPasswordController@showLinkRequestForm	web,guest
	POST	password/reset		App\Http\Controllers\Auth\ResetPasswordController@reset	web,guest
	GET HEAD	password/reset/{token}	password.reset	App\Http\Controllers\Auth\ResetPasswordController@showResetForm	web,guest
	POST	register		App\Http\Controllers\Auth\RegisterController@register	web,guest
	GET HEAD	register	register	App\Http\Controllers\Auth\RegisterController@showRegistrationForm	web,guest
	GET HEAD	test		Closure	web

Gambar 13 Named routes

Kemudian tambahkan `@yield` untuk menampung script javascript.

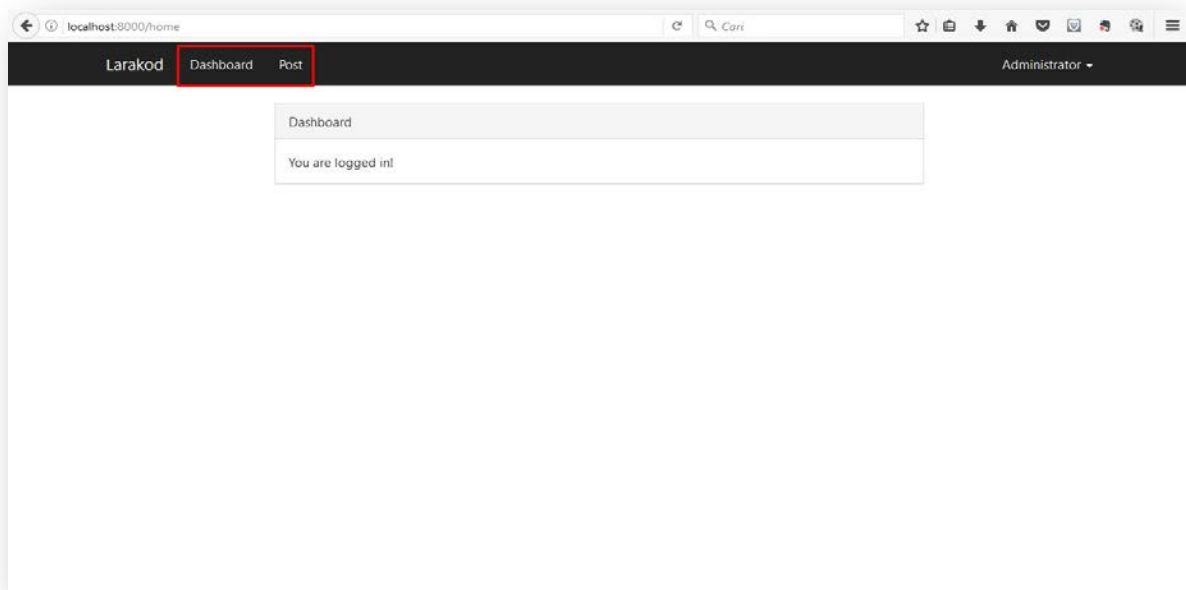
app/Http/Controller/PostController.php

```
<!DOCTYPE html>
<html lang="{{ app()->getLocale() }}">
<head>
    ...
</head>
<body>
    ...
    <!-- Scripts -->
    <script src="{{ asset('js/jquery-3.2.1.min.js') }}"></script>
    <script src="{{ asset('js/bootstrap.min.js') }}"></script>
    <script src="{{ asset('js/jquery.dataTables.min.js') }}"></script>
    <script src="{{ asset('js/dataTables.bootstrap.min.js') }}"></script>

    @yield('scripts')

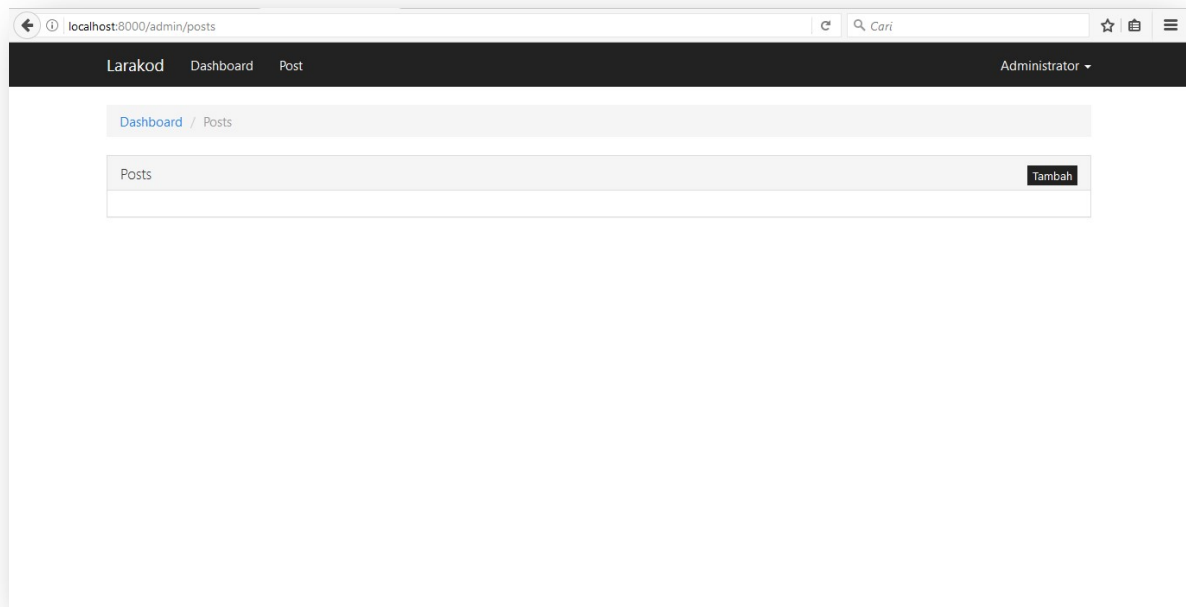
</body>
</html>
```

Sekarang pada navbar kita sudah melihat link menu Dashboard dan Post.



Gambar 14 Tampilan link pada navbar

Klik link **Post** pada navbar



Gambar 15 Tampilan halaman Post

Menampilkan Data dengan DataTables

Sekarang kita akan membuat table untuk menampilkan data pada table posts. Kita akan membuat table dengan menggunakan DataTables.

Sebelumnya, impor class **Posts**, **DataTables** dan **Builder** ke dalam `PostController.php`.

app/Http/Controller/PostController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Post;
use Yajra\DataTables\DataTables;
use Yajra\DataTables\Html\Builder;
```

Kemudian modifikasi method `index`, seperti berikut :

app/Http/Controller/PostController.php

```
public function index(Request $request, Builder $htmlBuilder)
{
    if ($request->ajax()) {
        $posts =
        Post::select(['id', 'title', 'seotitle', 'author', 'content', 'hits', 'status']);
        return DataTables::of($posts)->make(true);
    }

    $html = $htmlBuilder
        ->addColumn(['data'=>'title', 'name'=>'title', 'title'=>'Title'])
        ->addColumn(['data'=>'author', 'name'=>'author', 'title'=>'Author'])
        ->addColumn(['data'=>'content', 'name'=>'content',
        'title'=>'Content'])
        ->addColumn(['data'=>'hits', 'name'=>'hits', 'title'=>'Hits'])
        ->addColumn(['data'=>'status', 'name'=>'status', 'title'=>'Status']);

    return view('posts.index')->with(compact('html'));
}
```

Selanjutnya, buka `index.blade.php`

resources/views/posts/index.blade.php

```
@extends('layouts.app')

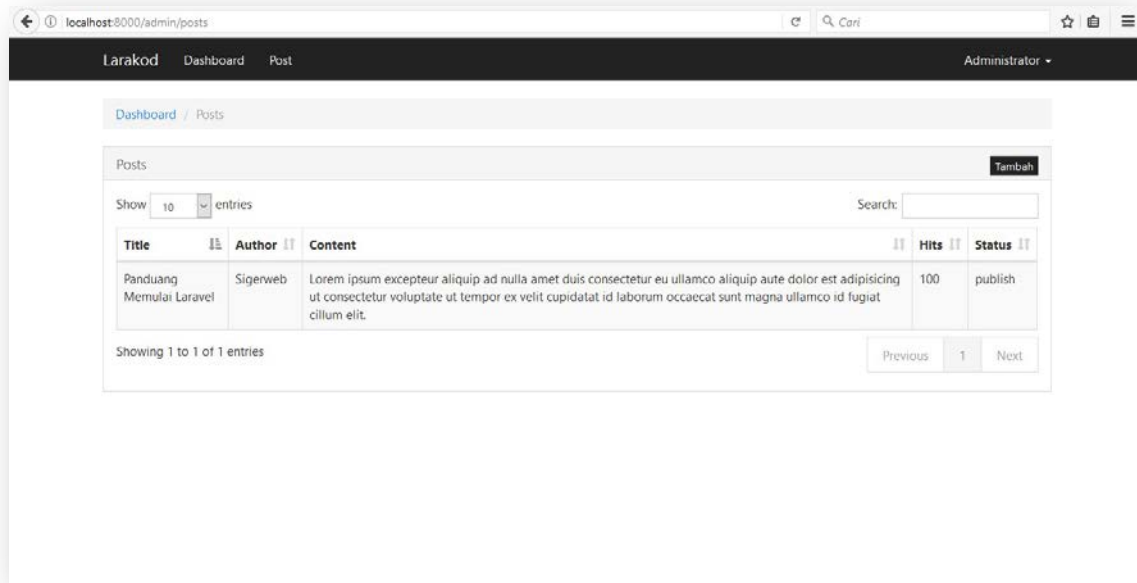
@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-12">
            <ul class="breadcrumb">
                <li><a href="{{ url('/home') }}">Dashboard</a></li>
                <li class="active">Posts</li>
            </ul>

            <div class="panel panel-default">
                <div class="panel-heading">
                    <h2 class="panel-title">Posts
                    <span class="pull-right">
                        <a class="btn btn-xs btn-default" href="{{
route('posts.create') }}">Tambah</a>
                    </span>
                </div>

                <div class="panel-body">
                    {!! $html->table(['class'=>'table table-striped table-
bordered']) !!}
                </div>
            </div>
        </div>
    </div>
</div>
@endsection

@section('scripts')
{!! $html->scripts() !!}
@endsection
```

Hasilnya :



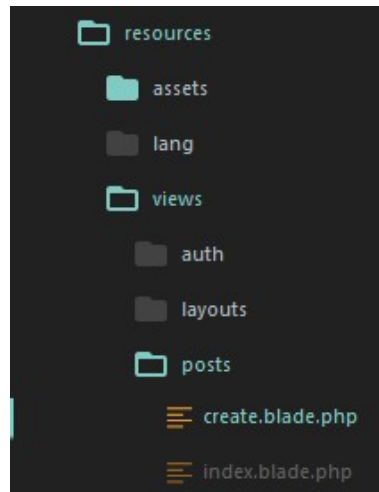
Gambar 16 Tampilan DataTables Post

Membuat Halaman Form

Form Input

Menyiapkan File dan Controller

Buatlah file baru di dalam folder `posts`, beri nama `create.blade.php`.



Gambar 17 File Create Blade.php

Masukkan script ke dalam `create.blade.php`, sehingga menjadi seperti berikut:

resources/views/posts/create.blade.php

```
@extends('layouts.app')

@section('content')
<div class="container">
  <div class="row">
    <div class="col-md-12">
      <ul class="breadcrumb">
        <li><a href="{{ url('/home') }}">Dashboard</a></li>
        <li><a href="{{ route('posts.index') }}">Posts</a></li>
        <li class="active">Tambah Posts</li>
      </ul>

      <div class="panel panel-default">
        <div class="panel-heading">Tambah Posts</div>

        <div class="panel-body">

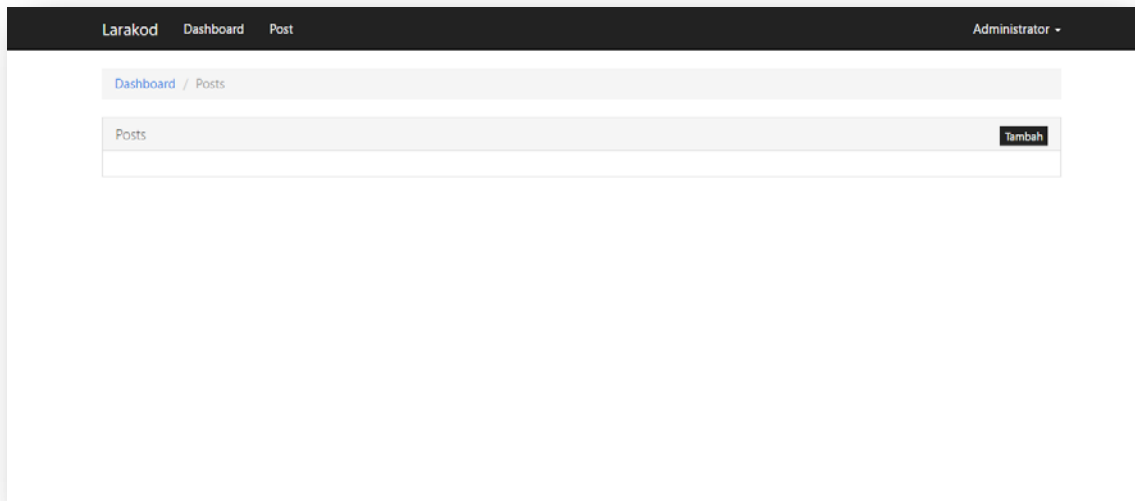
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
@endsection
```

Kemudian, buka `PostController.php`, tambahkan script berikut di dalam method `create`.

app/Http/Controller/PostController.php

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

Kemudian, jalankan `localhost:8000/admin/posts/create`.



Gambar 18 Tampilan Halaman Form sementara

Membuat Form dengan laravelcollective/html

Buka kembali `create.blade.php`. Salin script baru :

```
{!! Form::open(['url' => route('posts.store'), 'method' => 'post',
'class'=>'form-horizontal']) !!}
    @include('posts._form')
{!! Form::close() !!}
```

Sisipkan di dalam `body panel` , sehingga hasilnya akan seperti berikut:

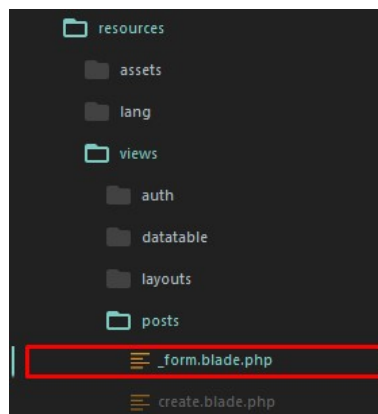
resources/views/posts/create.blade.php

```
@extends('layouts.app')

@section('content')
<div class="container">
  <div class="row">
    <div class="col-md-12">
      <ul class="breadcrumb">
        <li><a href="{{ url('/home') }}">Dashboard</a></li>
        <li><a href="{{ route('posts.index') }}">Posts</a></li>
        <li class="active">Tambah Posts</li>
      </ul>
      <div class="panel panel-default">
        <div class="panel-heading">Posts</div>
        <div class="panel-body">
          {!! Form::open(['url' => route('posts.store'),
            'method' => 'post', 'class' => 'form-horizontal']) !!}
          @include('posts._form')
          {!! Form::close() !!}
        </div>
      </div>
    </div>
  </div>
</div>
@endsection
```

Isi form akan di-include dari file `_form.blade.php`.

Buat file baru `_form.blade.php`.



Gambar 19 File `_form.blade.php`

Masukkan script di bawah ini ke `_form.blade.php`

resources/views/posts/_form.blade.php

```
<div class="form-group{{ $errors->has('name') ? ' has-error' : '' }}">
  <div class="form-group">
    {!! Form::label('title', 'Title', ['class' => 'col-sm-2 control-
```

```

label']) !!}
    <div class="col-sm-8">
        {!! Form::text('title', null, ['class'=>'form-control']) !!}
        {!! $errors->first('title', '<p class="help-
block">:message</p>') !!}
    </div>
</div>
<div class="form-group">
    {!! Form::label('headline', 'Headline', ['class'=>'col-sm-2
control-label']) !!}
    <div class="col-sm-8">
        <div class="radio">
            <label>
                {!! Form::radio('headline', 'Y') !!} Ya
            </label>
        </div>
        <div class="radio">
            <label>
                {!! Form::radio('headline', 'N', true) !!} Tidak
            </label>
        </div>
    </div>
</div>
<div class="form-group">
    {!! Form::label('active', 'Active', ['class'=>'col-sm-2
control-label']) !!}
    <div class="col-sm-8">
        <div class="radio">
            <label>
                {!! Form::radio('active', 'Y', true) !!} Ya
            </label>
        </div>
        <div class="radio">
            <label>
                {!! Form::radio('active', 'N') !!} Tidak
            </label>
        </div>
    </div>
</div>
<div class="form-group">
    {!! Form::label('image', 'Image', ['class'=>'col-sm-2 control-
label']) !!}
    <div class="col-sm-8">
        <div class="input-group">
            {!! Form::text('fupload', null, ['class'=>'form-control',
'id'=>'fupload']) !!}
            <span class="input-group-btn">
                <a href="#" class="btn btn-success">Pilih Gambar</a>
            </span>
        </div>
    </div>
</div>
<div class="form-group">
    {!! Form::label('content', 'Content', ['class'=>'col-sm-2
control-label']) !!}
    <div class="col-sm-8">
        {!! Form::textarea('content', null, ['class'=>'form-control'])
!!}

```

```

        {!! $errors->first('content', '<p class="help-
block">:message</p>') !!}
    </div>
</div>
<div class="form-group">
    {!! Form::label('status', 'Status', ['class'=>'col-sm-2
control-label']) !!}
    <div class="col-sm-8">
        <div class="radio">
            <label>
                {!! Form::radio('status', 'publish') !!} Publish
            </label>
        </div>
        <div class="radio">
            <label>
                {!! Form::radio('status', 'draft', true) !!} Draft
            </label>
        </div>
    </div>
</div>
</div>
{!! Form::hidden('author', Auth::user()->name) !!}
<div class="form-group">
    <div class="col-sm-offset-2 col-sm-8">
        {!! Form::submit('Simpan', ['class'=>'btn btn-primary']) !!}
        {!! Form::button('Batal', ['class'=>'btn btn-warning']) !!}
    </div>
</div>
</div>

```

Buka `PostController.php`. Salin script berikut :

```

$this->validate($request, ['title' => 'required', 'content'=>'required']);
$posts = Post::create(['title' => $request->title, 'seotitle' =>
str_slug($request->title, '-'), 'author' => $request->author, 'headline'
=> $request->headline, 'active' => $request->active, 'image' => $request-
>fupload, 'content' => $request->content, 'status' => $request->status]);
return redirect()->route('posts.index');

```

Tambahkan ke dalam method `store` :

app/Http/Controller/PostController.php

```

/**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    $this->validate($request, ['title' =>

```

```

        'required', 'content' => 'required']);

        $posts = Post::create(['title' => $request->title, 'seotitle' =>
str_slug($request->title, '-'), 'author' => $request->author, 'headline'
=> $request->headline, 'active' => $request->active, 'image' => $request-
>fupload, 'content' => $request->content, 'status' => $request->status]);
        return redirect()->route('posts.index');
    }

```

Buka model `Post.php`, tambahkan :

```

protected $fillable = ['title', 'seotitle', 'author', 'content', 'image', 'hits',
'headline', 'active', 'status'];

```

Masukkan ke dalam class `Post`

app/Post.php

```

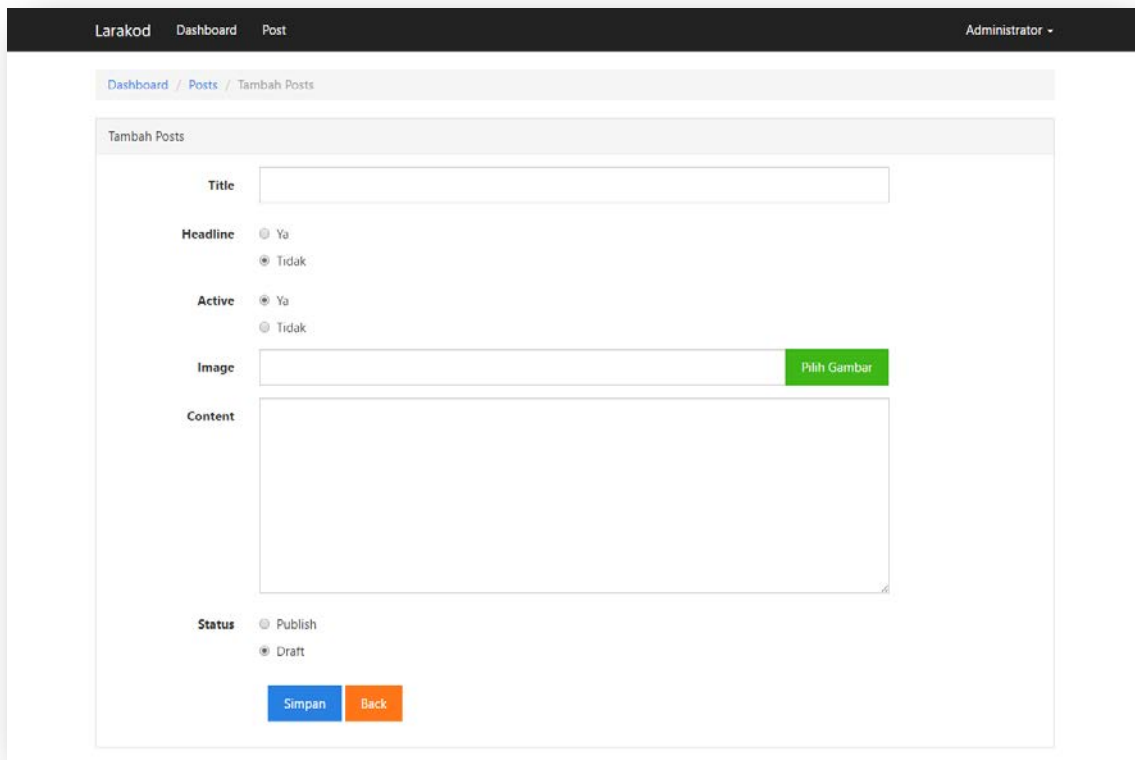
<?php
namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    protected $fillable = ['title', 'seotitle', 'author', 'content',
'image', 'hits', 'headline', 'active', 'status'];
}

```

Hasilnya form input akan tampak sebagai berikut:



The screenshot shows a web application interface for adding a new post. The top navigation bar includes 'Larakod', 'Dashboard', 'Post', and a user profile 'Administrator'. The breadcrumb trail is 'Dashboard / Posts / Tambah Posts'. The form itself is titled 'Tambah Posts' and contains the following elements:

- Title:** A text input field.
- Headline:** Radio buttons for 'Ya' and 'Tidak', with 'Tidak' selected.
- Active:** Radio buttons for 'Ya' and 'Tidak', with 'Ya' selected.
- Image:** A text input field and a green button labeled 'Pilih Gambar'.
- Content:** A large text area for the post content.
- Status:** Radio buttons for 'Publish' and 'Draft', with 'Draft' selected.
- Buttons:** A blue 'Simpan' button and an orange 'Back' button.

Gambar 20 Tampilan form input Post

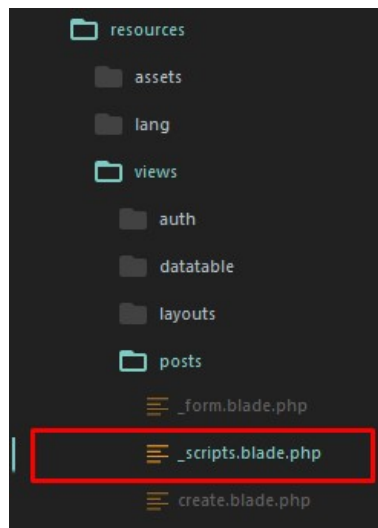
Menambahkan Responsive File Manager

Tambahkan script berikut pada tag a :

resources/views/posts/_form.blade.php

```
...
<div class="form-group">
    {!! Form::label('image', 'Image', ['class'=>'col-sm-2 control-label']) !!}
    <div class="col-sm-8">
        <div class="input-group">
            {!! Form::text('fupload', null, ['class'=>'form-control', 'id'=>'fupload']) !!}
            <span class="input-group-btn">
                <a data-fancybox data-type="iframe" href="{{
asset('js/filemanager/dialog.php?type=1&field_id=fupload&relative_url=1')
}}" class="btn btn-success">Pilih Gambar</a>
            </span>
        </div>
    </div>
</div>
...
```

Buat file baru `_scripts.blade.php` untuk menampung script js :



Gambar 21 file `_scripts.blade.php`

Buka `_scripts.blade.php`, tambahkan script sehingga menjadi seperti berikut :

resources/views/posts/_scripts.blade.php

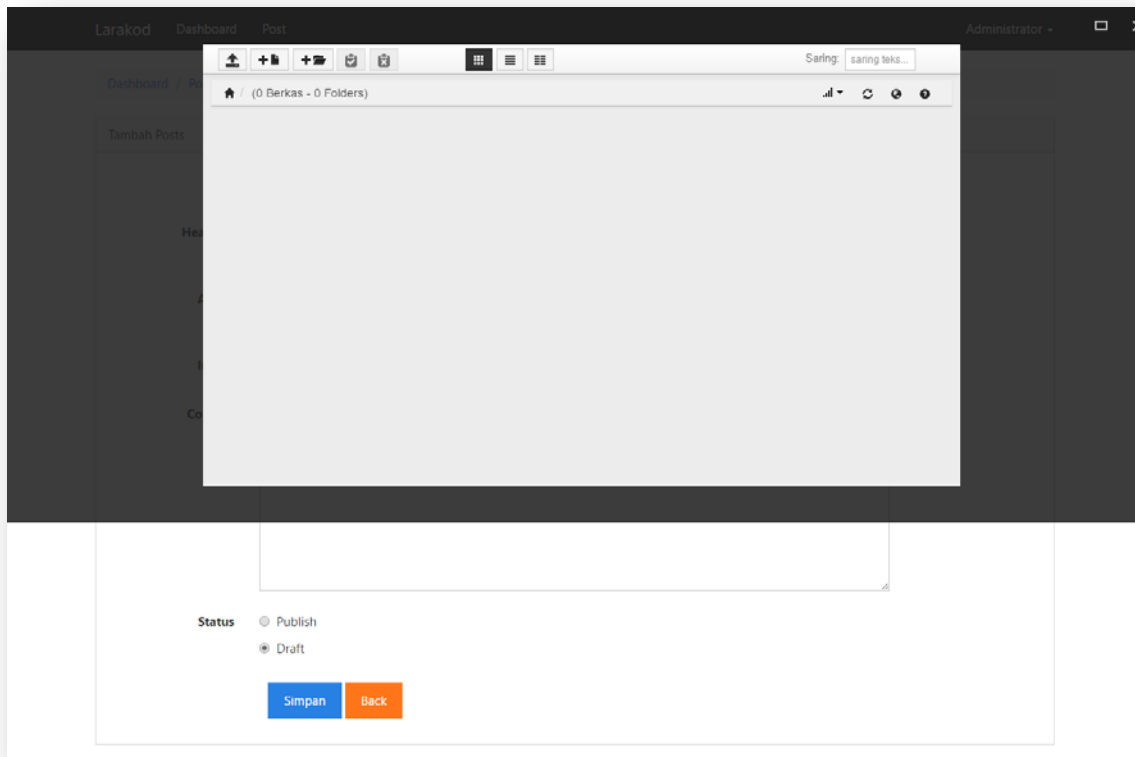
```
<script>
$(function() {
$('[data-fancybox]').fancybox({
    iframe : {
        preload : false,
        css : {
            width : '900px',
            height: '600px',
        }
    }
});
});
</script>
```

Terakhir, buka `create.blade.php`, include `_script.blade.php` :

resources/views/posts/create.blade.php

```
.....
@endsection
@section('scripts')
@include('posts._scripts')
@endsection
```


Hasilnya jika “Pilih Gambar” diklik akan tampil iframe yang berisi konten Responsive File Manager:



Gambar 22 Responsive File Manager

Menambahkan TinyMCE

Kita akan mengubah tampilan textarea biasa menjadi text editor dengan TinyMCE.

Buka `_script.blade.php`, tambahkan script seperti di bawah ini :

resources/views/posts/_scripts.blade.php

```
<script>
$(function() {
...
tinymce.init({
  selector: '#content',
  theme: 'modern',
  branding: false,
  menubar: false,
  language: "id",
  skin: "lightgray",
  plugins: [
    "advlist autolink link image lists charmap print preview hr
```

```

anchor pagebreak",
    "searchreplace wordcount visualblocks visualchars
insertdatetime media nonbreaking",
    "table contextmenu directionality emoticons paste textcolor
responsivefilemanager code"
],
    toolbar1: "undo redo | bold italic underline | alignleft aligncenter
alignright alignjustify | bullist numlist outdent indent | styleselect",
    toolbar2: "| responsivefilemanager | link unlink anchor | image media
| forecolor backcolor | print preview code ",
    image_advtab: true,
    relative_urls: true,
    external_filemanager_path: "{!!
str_finish(asset('js/filemanager'),'/') !!}",
    external_filemanager_path: "/js/filemanager/",
    filemanager_title: "Responsive Filemanager" ,
    external_plugins: { "filemanager" : "/js/filemanager/plugin.min.js"}
});
});
</script>

```

Terakhir, buka `PostController.php`, tambahkan pengaturan untuk tampilan kolom content pada view Posts agar menghilangkan tag-tag html dengan function `strip_tags` :

app/Http/Controller/PostController.php

```

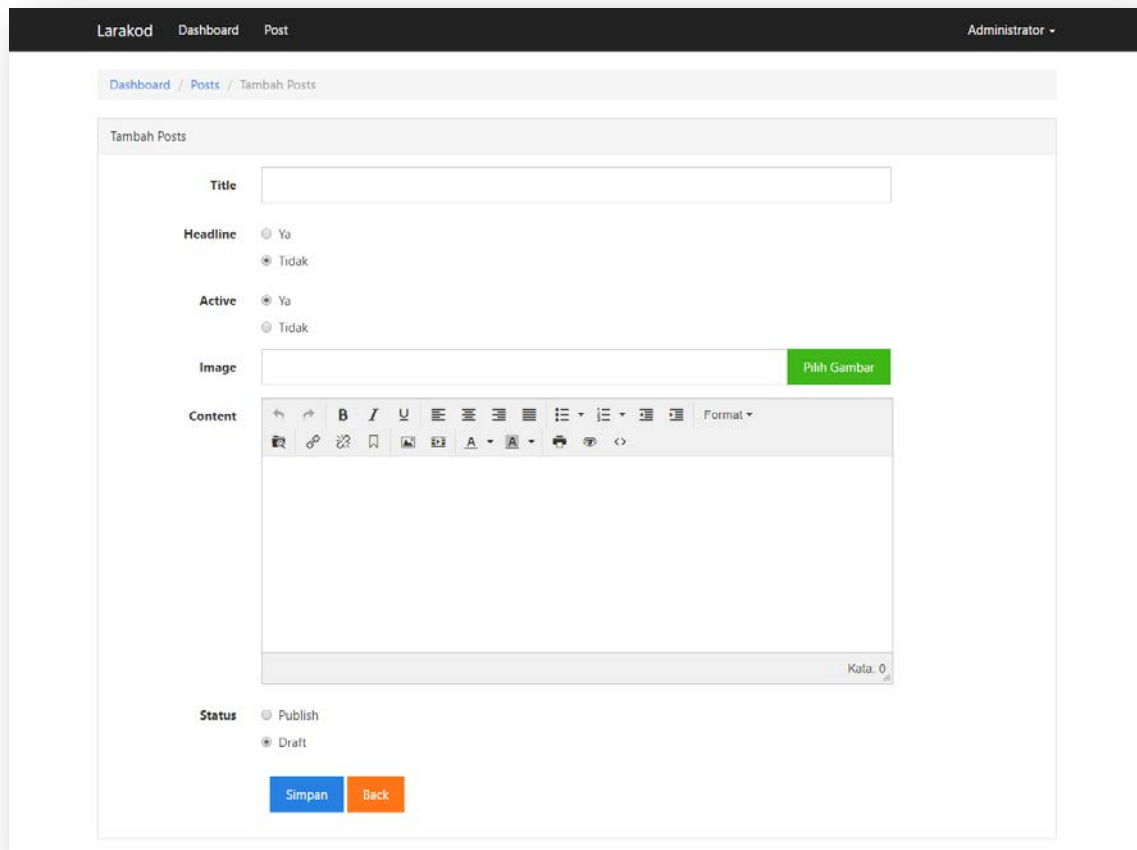
public function index(Request $request, Builder $htmlBuilder)
{
    if ($request->ajax()) {
        $posts =
Post::select(['id', 'title', 'seotitle', 'author', 'content', 'hits', 'status']);
        return DataTables::of($posts)
            ->addColumn('content', function($posts){
                return strip_tags($posts->content);
            })
            ->make(true);
    }

    $html = $htmlBuilder
        ->addColumn(['data'=>'title', 'name'=>'title', 'title'=>'Title'])
        ->addColumn(['data'=>'author', 'name'=>'author', 'title'=>'Author'])
        ->addColumn(['data'=>'content', 'name'=>'content',
'title'=>'Content'])
        ->addColumn(['data'=>'hits', 'name'=>'hits', 'title'=>'Hits'])
        ->addColumn(['data'=>'status', 'name'=>'status', 'title'=>'Status']);

    return view('posts.index')->with(compact('html'));
}

```

Hasilnya form input Posts akan terlihat sebagai berikut :



The screenshot shows a web interface for adding a new post. The top navigation bar includes 'Larakod', 'Dashboard', 'Post', and a user profile 'Administrator'. The breadcrumb trail is 'Dashboard / Posts / Tambah Posts'. The form itself is titled 'Tambah Posts' and contains several input fields: 'Title' (a text box), 'Headline' (radio buttons for 'Ya' and 'Tidak', with 'Tidak' selected), 'Active' (radio buttons for 'Ya' and 'Tidak', with 'Tidak' selected), 'Image' (a text box and a green 'Pilih Gambar' button), 'Content' (a TinyMCE editor with a toolbar and a 'Kata: 0' counter), and 'Status' (radio buttons for 'Publish' and 'Draft', with 'Draft' selected). At the bottom of the form are two buttons: 'Simpan' (blue) and 'Back' (orange).

Gambar 23 TinyMCE pada textarea

Menambahkan Flash Message

Flash Message merupakan pesan notifikasi jika suatu proses create, update, dan delete berhasil dilakukan. Flash Message akan ditampilkan di halaman view.

Buka `PostController.php`. Impor class `Session` dan tambahkan `session::flash()` :

App\Http\Controllers\PostController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Post;
use Yajra\DataTables\DataTables;
use Yajra\DataTables\Html\Builder;
use Session;

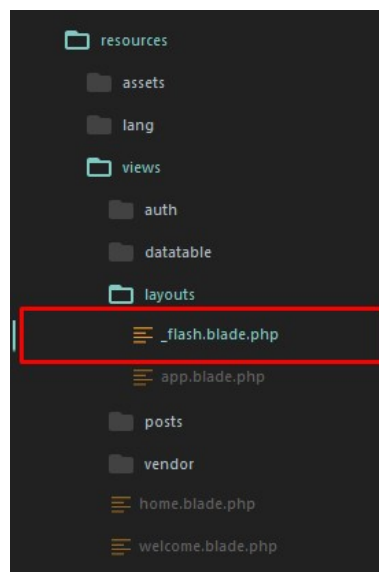
class PostController extends Controller
{
```

```

...
/**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    $this->validate($request, ['title' =>
    'required', 'content'=>'required']);
    $posts = Post::create(['title' => $request->title, 'seotitle' =>
    str_slug($request->title, '-'), 'author' => $request->author, 'headline'
    => $request->headline, 'active' => $request->active, 'image' => $request-
    >fupload, 'content' => $request->content, 'status' => $request->status]);
    Session::flash("flash_notification", [
        "level"=>"success",
        "message"=>"Data Post Berhasil Tersimpan"
    ]);
    return redirect()->route('posts.index');
}
...

```

Selanjutnya, buat file baru bernama `_flash.blade.php` di dalam folder `layouts`



Gambar 24 File `_flash.blade.php`

Masukkan script berikut ke dalam `_flash.blade.php`

resources/views/layouts/_flash.blade.php

```

@if (session()->has('flash_notification.message'))
<div class="container">
<div class="alert alert-{{ session()->get('flash_notification.level')
}}">

```

```
<button type="button" class="close" data-dismiss="alert" aria-
hidden="true">&times;</button>
{!! session()->get('flash_notification.message') !!}
</div>
</div>
@endif
```

Terakhir, buka `app.blade.php`

Tambahkan :

```
@include('layouts._flash')
```

Tepat di atas `@yield('content')`

resources/views/layouts/_flash.blade.php

```
.....
@include('layouts._flash')
@yield('content')
.....
@yield('scripts')
.....
```

Sekarang kita akan coba masukkan data pada form input posts :

Larakod
Dashboard
Post
Administrator

Dashboard / Posts / Tambah Posts

Tambah Posts

Title
Mengenal Laravel 5.4

Headline
☐ Ya
☒ Tidak

Active
☒ Ya
☐ Tidak

Image
LaravelLogo.png
Pilih Gambar

Content

B
I
U
List
Link
Image
Table
A
A
Format

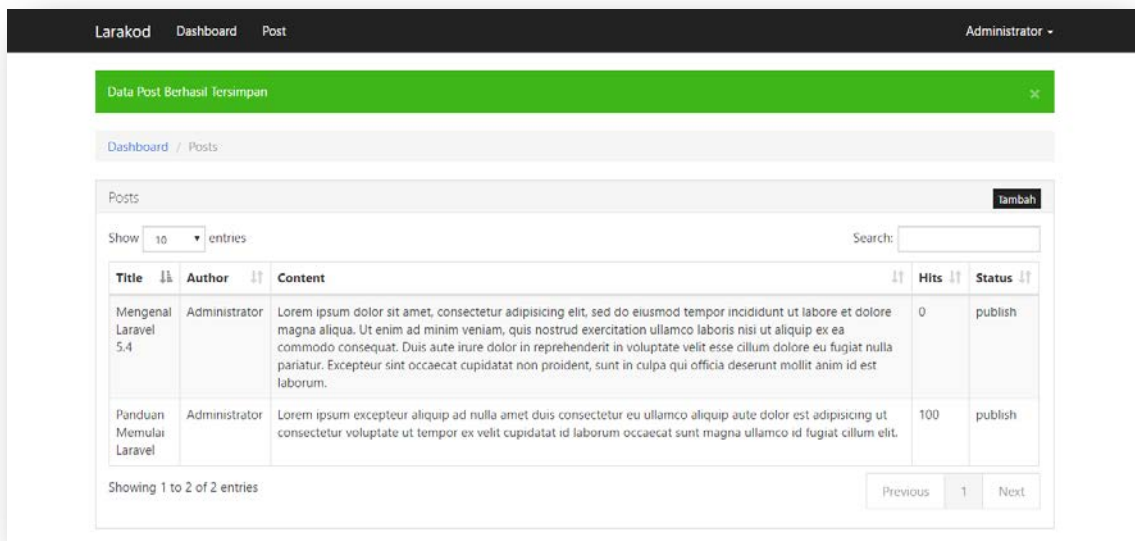
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

p
Kata: 69

Status
☒ Publish
☐ Draft

Simpan
Back

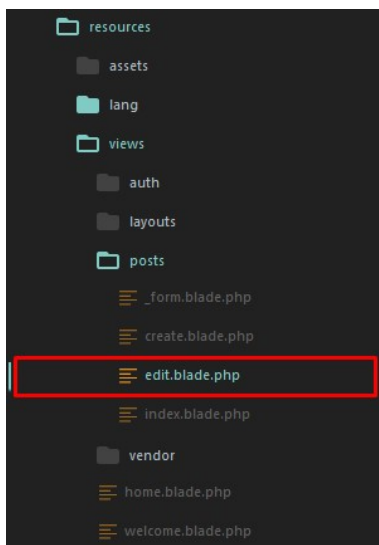
Klik tombol Simpan, hasilnya akan terlihat seperti berikut :



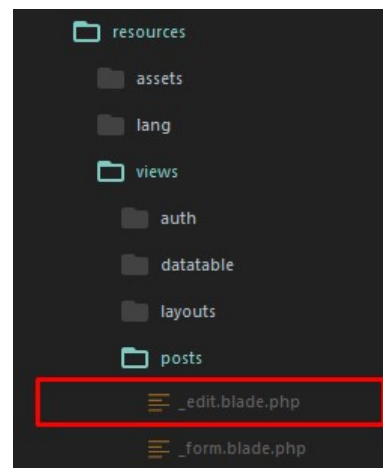
Gambar 25 Flash Message ketika berhasil input data

Form Edit

Buat file baru `edit.blade.php` dan `_edit.blade.php`



Gambar 26 File `edit.blade.php`



Gambar 27 File `_edit.blade.php`

Masukkan script di bawah ini ke dalam `edit.blade.php`

resources/views/posts/edit.blade.php

```
@extends('layouts.app')

@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-12">
            <ul class="breadcrumb">
                <li><a href="{{ url('/home') }}">Dashboard</a></li>
                <li><a href="{{ route('posts.index') }}">Posts</a></li>
                <li class="active">Edit Posts</li>
            </ul>

            <div class="panel panel-default">
                <div class="panel-heading">Posts</div>
                <div class="panel-body">
                    {!! Form::model($posts, ['url' => route('posts.update',
$posts->id),
                    'method' => 'put', 'class'=>'form-horizontal']) !!}
                    @include('posts._edit')
                    {!! Form::close() !!}
                </div>
            </div>
        </div>
    </div>
</div>
@endsection
```

Kemudian, buka file `_edit.blade.php`, isikan dengan script di bawah ini :

resources/views/posts/_edit.blade.php

```
<div class="form-group{{ $errors->has('name') ? ' has-error' : '' }}">
    <div class="form-group">
        {!! Form::label('title', 'Title', ['class'=>'col-sm-2 control-
label']) !!}
        <div class="col-sm-8">
            {!! Form::text('title', null, ['class'=>'form-control']) !!}
            {!! $errors->first('title', '<p class="help-
block">:message</p>') !!}
        </div>
    </div>
    <div class="form-group">
        {!! Form::label('headline', 'Headline', ['class'=>'col-sm-2
control-label']) !!}
        <div class="col-sm-8">
            <div class="radio">
                <label>
                    {!! Form::radio('headline', 'Y') !!} Ya
                </label>
            </div>
        </div>
    </div>
```

```

        <div class="radio">
            <label>
                {!! Form::radio('headline', 'N', true) !!} Tidak
            </label>
        </div>
    </div>
</div>
<div class="form-group">
    {!! Form::label('active', 'Active', ['class'=>'col-sm-2
control-label']) !!}
    <div class="col-sm-8">
        <div class="radio">
            <label>
                {!! Form::radio('active', 'Y', true) !!} Ya
            </label>
        </div>
        <div class="radio">
            <label>
                {!! Form::radio('active', 'N') !!} Tidak
            </label>
        </div>
    </div>
</div>
<!-- Images -->
<div class="form-group">
    {!! Form::label('image', 'Image', ['class'=>'col-sm-2
control-label']) !!}
    <div class="col-sm-8">
        image}}">
        <div class="input-group">
            {!! Form::text('image', null, ['class'=>'form-
control']) !!}
            <span class="input-group-btn">
                <a data-fancybox data-type="iframe" href="{{
asset('js/filemanager/dialog.php?type=1&field_id=image&relative_url=1')
}}" class="btn btn-success" type="button">Ganti Gambar</a>
            </span>
        </div>
    </div>
</div>
<div class="form-group">
    {!! Form::label('content', 'Content', ['class'=>'col-sm-2
control-label']) !!}
    <div class="col-sm-8">
        {!! Form::textarea('content', null, ['class'=>'form-control'])
!!}
        {!! $errors->first('content', '<p class="help-
block">:message</p>') !!}
    </div>
</div>
<div class="form-group">
    {!! Form::label('status', 'Status', ['class'=>'col-sm-2
control-label']) !!}
    <div class="col-sm-8">
        <div class="radio">
            <label>
                {!! Form::radio('status', 'publish') !!} Publish

```



```

        </label>
    </div>
    <div class="radio">
        <label>
            {!! Form::radio('status', 'draft', true) !!} Draft
        </label>
    </div>
</div>
</div>
</div>
{!! Form::hidden('author', Auth::user()->name) !!}
<div class="form-group">
    <div class="col-sm-offset-2 col-sm-8">
        {!! Form::submit('Ubah', ['class'=>'btn btn-primary']) !!}
        <a href="{{ url()->previous() }}" class="btn btn-warning"> Batal</a>
    </div>
</div>

```

Buka `PostController.php`.

Perhatikan method `index`, sisipkan script yang bercetak tebal berikut ini :

app/Http/Controllers/PostController.php

```

if ($request->ajax()) {
    $posts = Post::select(['id', 'title', 'seotitle', 'author',
        'content', 'hits', 'status']);
    return DataTables::of($posts)
        ->addColumn('action', function($posts){
            return view('datatable._action', [
                'edit_url' => route('posts.edit', $posts->id)]);
        })
        ->addColumn('content', function($posts){
            return strip_tags($posts->content);
        })
        ->make(true);
}
$html = $htmlBuilder
    ->addColumn(['data' => 'title', 'name' => 'title', 'title'
=> 'Title'])
    ->addColumn(['data' => 'author', 'name' => 'author', 'title'
=> 'Author'])
    ->addColumn(['data' => 'content', 'name' => 'content',
'title' => 'Content'])
    ->addColumn(['data' => 'hits', 'name' => 'hits', 'title' =>
'Hits'])
    ->addColumn(['data' => 'status', 'name' => 'status',
'title'=>'Status'])
    ->addColumn(['data' => 'action', 'name' => 'action', 'title'
=> 'Action', 'orderable' => false, 'searchable' => false]);

```

Masukkan script ini ke dalam method `edit`.

app/Http/Controllers/PostController.php

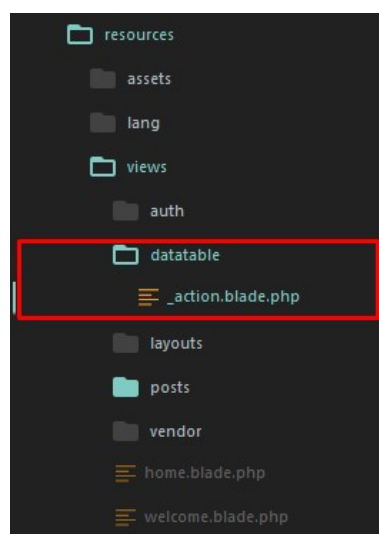
```
public function edit($id)
{
    $posts = Post::find($id);
    return view('posts.edit', ['posts' => $posts])->with(compact('posts'));
}
```

Masukkan script ini ke dalam method `update`.

app/Http/Controllers/PostController.php

```
public function update(Request $request, $id)
{
    $this->validate($request, ['title' => 'required', 'content' =>
    'required']);
    $posts = Post::find($id);
    $posts->update(['title' => $request->title, 'seotitle' =>
    str_slug($request->title, '-'), 'author' => $request->author, 'image' =>
    $request->image, 'content' => $request->content, 'status' => $request-
    >status]);
    Session::flash("flash_notification", ["level" => "success", "message" =>
    "Data Post berhasil diubah"]);
    return redirect()->route('posts.index');
}
```

Terakhir buatlah folder `datatable` dan file `_action.blade.php`



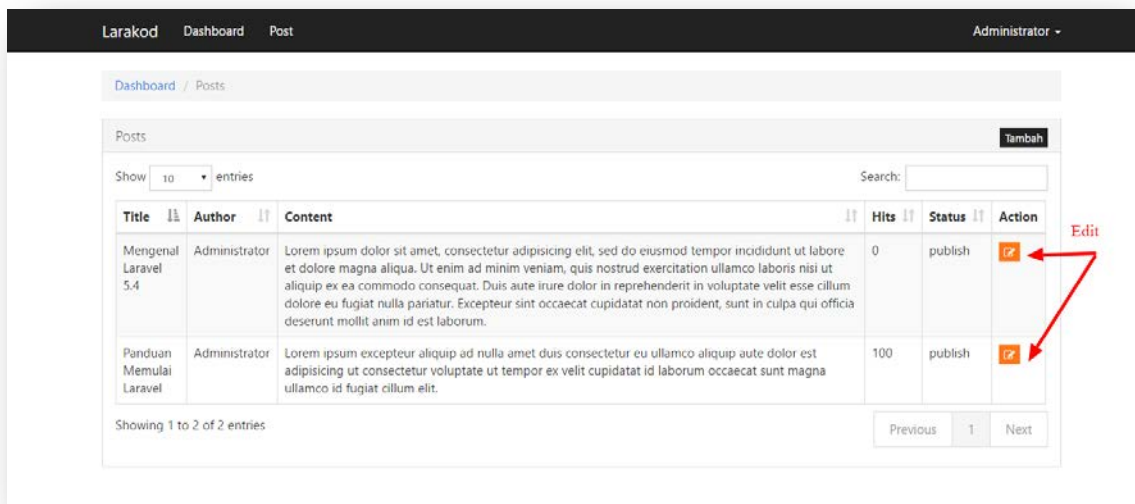
Gambar 28 Folder `datatable` dan file `_action.blade.php`

Isikan file `_action.blade.php` dengan syntax seperti berikut :

resources/views/datatables/_action.blade.php

```
<a href="{!! $edit_url !!}" class="btn btn-xs btn-warning"><i class="fa fa-edit"></i></a>
```

Hasilnya :



Gambar 29 Kolom action pada datatables

Proses Hapus Data

Buka `PostController.php`, edit bagian method `destroy`, masukkan script baru sehingga isinya menjadi seperti berikut :

app/Http/Controllers/PostController.php

```
...  
public function destroy($id)  
{  
    Post::destroy($id);  
    Session::flash("flash_notification", [  
        "level"=>"success",  
        "message"=>"Data Post berhasil dihapus"  
    ]);  
    return redirect()->route('posts.index');  
}
```

Tambahkan pengaturan route untuk tombol hapus di kolom Action pada DataTable. Tambahkan script berikut ke dalam method index:

```

...
public function index(Request $request, Builder $htmlBuilder)
{
    if ($request->ajax()) {
        $posts = Post::select( ['id', 'title', 'seotitle', 'author', 'content',
        'hits', 'status'] );
        return DataTables::of($posts)
        ->addColumn('action', function($posts){
            return view('datatable._action', [
                'model' => $posts,
                'form_url' => route('posts.destroy', $posts->id),
                'edit_url' => route('posts.edit', $posts->id)]);
        }->make(true);
    }
}

```

Buka file `_action.blade.php`, tambahkan script untuk membuat tombol hapus :

resources/views/datatable/_action.blade.php

```

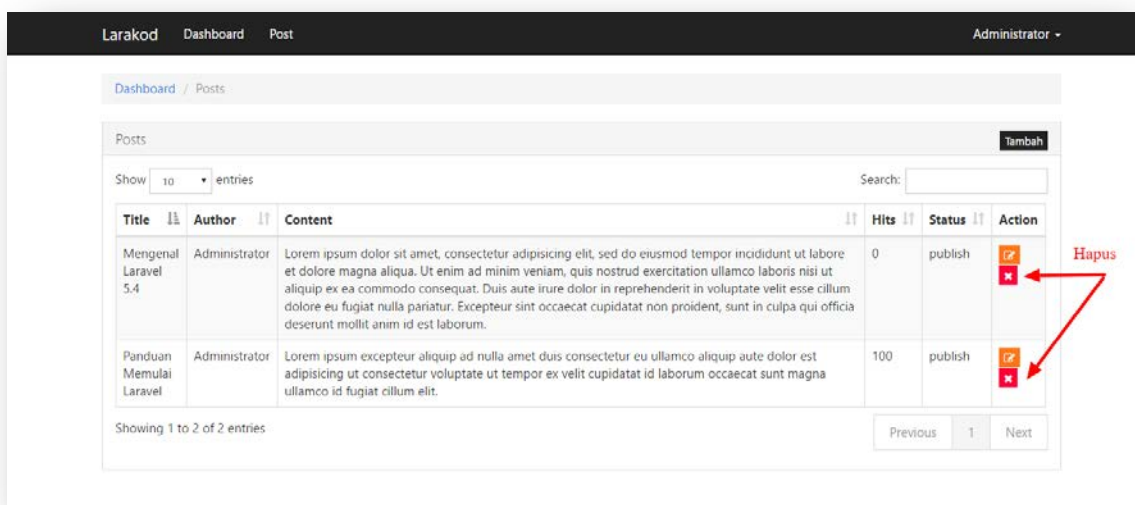
{!! Form::model($model, ['url'=>$form_url, 'method'=>'delete',
'class'=>'form-inline']) !!}

<a href="{!! $edit_url !!}" class="btn btn-xs btn-warning"><i class="fa
fa-edit"></i></a>

{!! Form::button('<i class="fa fa-remove"></i>',
['type'=>'submit', 'class'=>'btn btn-xs btn-danger']) !!}
{!! Form::close() !!}

```

Hasilnya akan menjadi seperti berikut :



Gambar 30 Tombol hapus

Penutup

Akhirnya kita sampai pada penghujung ebook ini. Kita sudah mempelajari sebagian kecil dari Laravel dengan cara praktik langsung dari mulai menginstall sampai dengan operasi CRUD. Kini, sebagai pemula Anda sudah memiliki kemampuan dasar yang dibutuhkan untuk membuat project aplikasi dengan Laravel.

Daftar Situs Belajar laravel

Berikut ini merupakan daftar situs yang kami rekomendasikan kepada Anda untuk mempelajari Laravel :

- ✓ <https://laravel.com/>
- ✓ <https://laracasts.com/>
- ✓ <http://laravel-tricks.com/>
- ✓ <http://laravel-recipes.com/>
- ✓ <https://malescast.com/topic/laravel/>
- ✓ <https://www.sekolahkoding.com/kelas/belajar-laravel-53-dasar>

Jika Anda kesulitan mengikuti pembahasan dalam situs berbahasa Inggris, Anda bisa memulai dengan situs lokal yang berbahasa Indonesia, seperti sekolahkoding.com dan malescasts.com, atau situs-situs lain yang memiliki pembahasan seputar Laravel. Anda juga bisa memanfaatkan facebook untuk belajar dan bertanya langsung tentang Laravel dengan join **Group Laravel Indonesia** (Unofficial) di <https://web.facebook.com/groups/laravel>.

Jangan Berhenti di Ebook ini

Pelajari terus, kembangkan kemampuan Anda, miliki rasa penasaran dan antusiasme yang besar untuk mempelajari framework ini setahap demi setahap. Kita tidak bisa mempelajari framework dari A sampai Z hanya dengan satu ebook dan kemudian membuat Anda langsung menguasai Laravel, Anda butuh proses, Anda butuh waktu. Jangan berhenti hanya di ebook ini, Anda bisa membeli buku-buku Laravel beserta contoh studi kasus membuat project aplikasi, atau membaca ebook-ebook lainnya seperti **Seminggu Belajar Laravel**²⁰ dan **Menyelami Framework Laravel**²¹ karya Rahmat Awaludin. Laravel akan terus berkembang dan akan segera merilis versi 5.5, jika Anda tidak mengikuti perkembangan Laravel Anda akan tertinggal, namun jika Anda sudah mempelajari Laravel pada versi-versi sebelumnya Anda masih dapat mengikuti perkembangannya karena Anda sudah memiliki *basic* atau kemampuan dasar Laravel. *Next*, kami akan memudahkan Anda untuk mempelajari Laravel secara lebih sederhana dengan metode belajar “*Learning by Doing*” beserta studi kasus membuat project aplikasi dari awal sampai akhir di ebook kedua kami. Penasaran seperti apa, tunggu ebook kami selanjutnya.

Akhir kata, terima kasih.. dan selamat belajar.

²⁰ <https://leanpub.com/seminggubelajarlaravel>

²¹ <https://leanpub.com/bukularavel>

♥ Happy coding with **Laravel** ♥