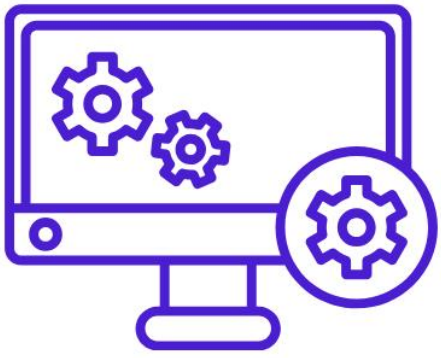
PEMROGRAMAN WEB

RPL

SMK NEGERI 2 TRENGGALEK



XI

KEGIATAN BELAJAR 6

- LARAVEL 10 -

- *LAYOUT*ING, *MODEL, CONTROLLER* DAN *MIGRATION* -

A. Capain Pembelajaran

1. Menerapkan teknologi *framework* dalam aplikasi web
2. Membuat aplikasi web menggunakan teknologi *framework*

B. TUJUAN

Setelah proses belajar, berdiskusi, praktik dan menggali informasi, peserta didik diharapkan mampu :

1. Memahami *layouting, model, controller* dan *migration* secara sederhana pada *framework* Laravel.
2. Membuat *layout*ing yang memanfaatkan *blade templating engine* pada *framework* Laravel.
3. Memahami pemanfaatan *model, controller* dan *migration* pada *framework* Laravel.
4. Menerapkan *model, controller* dan *migration* pada *framework* Laravel untuk membangun sebuah *website*.

C. ALOKSI WAKTU

18 x 45 menit

D. DASAR TEORI

1. *LAYOUTING*

a. Pengertian *Layout*

Istilah *layout* mungkin sudah sangat familier bagi para desainer hingga arsitek. Namun bagi orang awam, istilah ini sering salah dipahami. Sederhananya, *layout* adalah tata letak, atau secara bahasa, *layout* merupakan rancangan desain tata letak dengan tujuan tertentu. Secara singkat *layout* adalah desain tata letak. Sedangkan

secara bahasa, *layout* adalah suatu susunan atau rancangan desain tata letak yang dibuat untuk tujuan tertentu. Tujuan pembuatan *layout* pada website adalah agar pengunjung dapat lebih mudah memahami informasi yang disajikan. Selain itu, tujuan pembuatan *layout* adalah agar tampilan *website* lebih menarik dilihat serta lebih *user-friendly*. Secara umum, *layout* pada *website* terdiri atas tiga bagian, yaitu *Header, Body, Sidebar,* dan *Footer*.

b. Menyusun Struktur *View* dan *Layout*

Kita akan mencoba dengan studi kasus sederhana, dimana sebuah tampilan *website* terdiri dari: *Header, Footer*, dan *Main Section*. Untuk mengakomodasi studi kasus di atas kita dapat membuat struktur *view* dan *layout* seperti berikut:

```
Resources
-- views
---- layouts
------ app.blade.php
---- products
------ guru.blade.php
------ siswa.blade.php
---- shared
------ header.blade.php
------ footer.blade.php
```

1) Layout

Layout bisa dikatakan sebagai kerangka tampilan dari sebuah website.

resources/views/layouts/app.blade.php

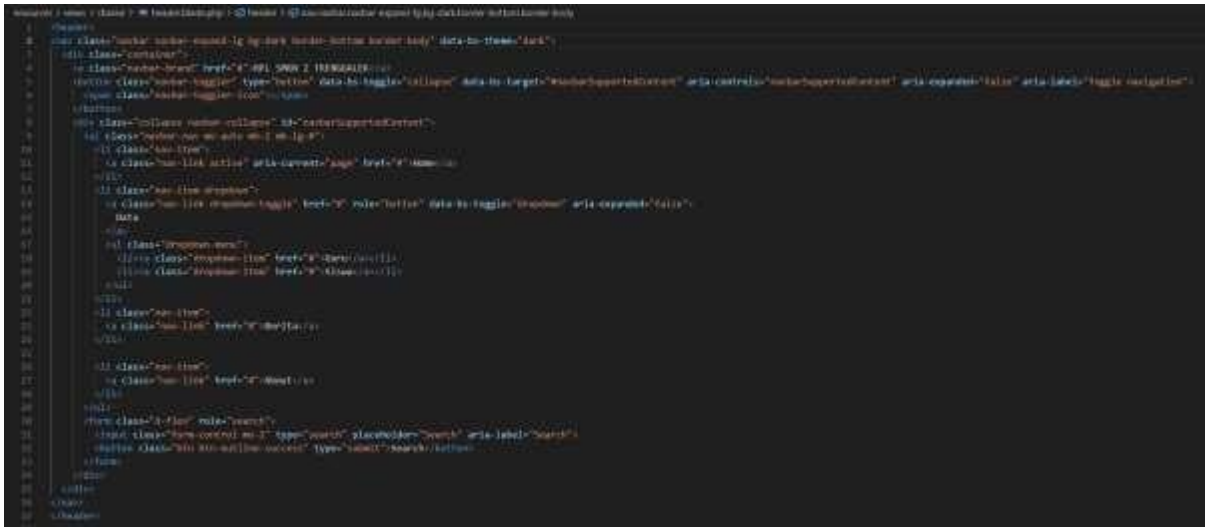
```
resources > views > layouts > app.blade.php > ...
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>MyData</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN" crossorigin="anonymous">
  </head>

  <body>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-C6RzsynM9kWDrMNeT87bh95OGNyZPhcTNXj1NW7RuBCsyN/o0jlpcV8Qyq46cDfL" crossorigin="anonymous"></script>
    <div id="app">
        <div class="main-wrapper">
            @include('shared.header')
            <div class="main-content">
            @yield('content')
            </div>
            @include('shared.footer')
            </div>
        </div>
  </body>
</html>
```

Kalau kita amati *blade* dari *layout* di atas, terdapat baris **`@include('shared.header')`** adalah untuk memanggil bagian *header*. **`@yield('content')`** akan memanggil bagian *section content* yang akan berubah sesuai yang akan ditampilkan. **`@include('shared.footer')`** akan memanggil bagian *footer*.

2) Header

Tampilan pada *header* berisi **navbar**, oleh karena itu silakan manfaatkan dokumentasi dari **bootsrap**. Adapun kode HTML untuk bagian *header* adalah seperti berikut ini:

resources/views/shared/header.blade.php



3) Footer

Kode HTML untuk bagian *footer* adalah seperti berikut ini:

resources/views/shared/footer.blade.php

```
resources > views > shared > footer.blade.php > ...
1   <div class="container">
2     <footer class="d-flex flex-wrap justify-content-between align-items-center py-3 my-4 border-top">
3       <p class="col-md-12 mb-0 text-muted">© 2023 - RPL, SMKN 2 Trenggalek </p>
4     </footer>
5   </div>
```

4) Section Content

Bagian *section content* ini kita akan mengambil contoh bagian tampil data yang ada di **resources/views/guru.blade.php**. Untuk menggunakan konsep *layout*, maka ada sedikit perubahan menjadi seperti berikut:

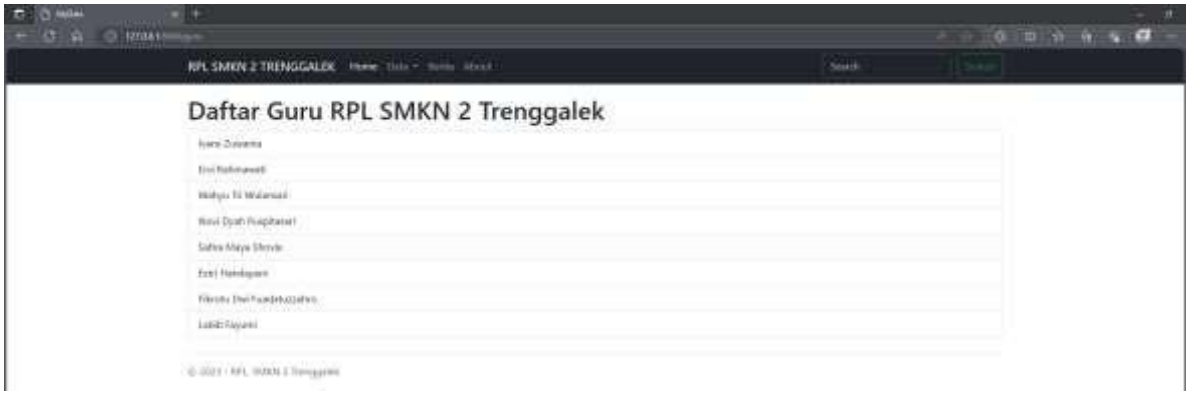
resources > views > guru.blade.php > div.container.mt-3 > ul.list-group > li.list-group-item

```
1   @extends('layouts.app')
2   @section('content')
3   <div class="container mt-3">
4   <h1>Daftar Guru RPL SMKN 2 Trenggalek</h1>
5     <ul class="list-group">
6       <li class="list-group-item">Ivans Zuwanta</li>
7       <li class="list-group-item">Ervi Rahmawati</li>
8       <li class="list-group-item">Wahyu Tri Wulansari</li>
9       <li class="list-group-item">Novi Dyah Puspitasari</li>
10      <li class="list-group-item">Safira Maya Shovie</li>
11      <li class="list-group-item">Estri Handayani</li>
12      <li class="list-group-item">Fikrotu Dwi Fuadatuzzahro</li>
13      <li class="list-group-item">Labib Fayumi</li>
14    </ul>
15    </div>
16    @endsection
```

Bagian yang perlu kita dicermati adalah `@extends('layouts.app')` dan `@section('content')`. `@extends('layouts.app')` menandakan bahwa untuk *view* tampil halaman data guru ini, kita akan menggunakan *layout* **`resources/views/layouts/app.blade.php`**. Sedangkan `@section('content')` menandakan tampilan dari *section* isi *body website* tersebut, dalam hal ini adalah menampilkan data guru dalam bentuk tabel.

5) Tampilan Hasil

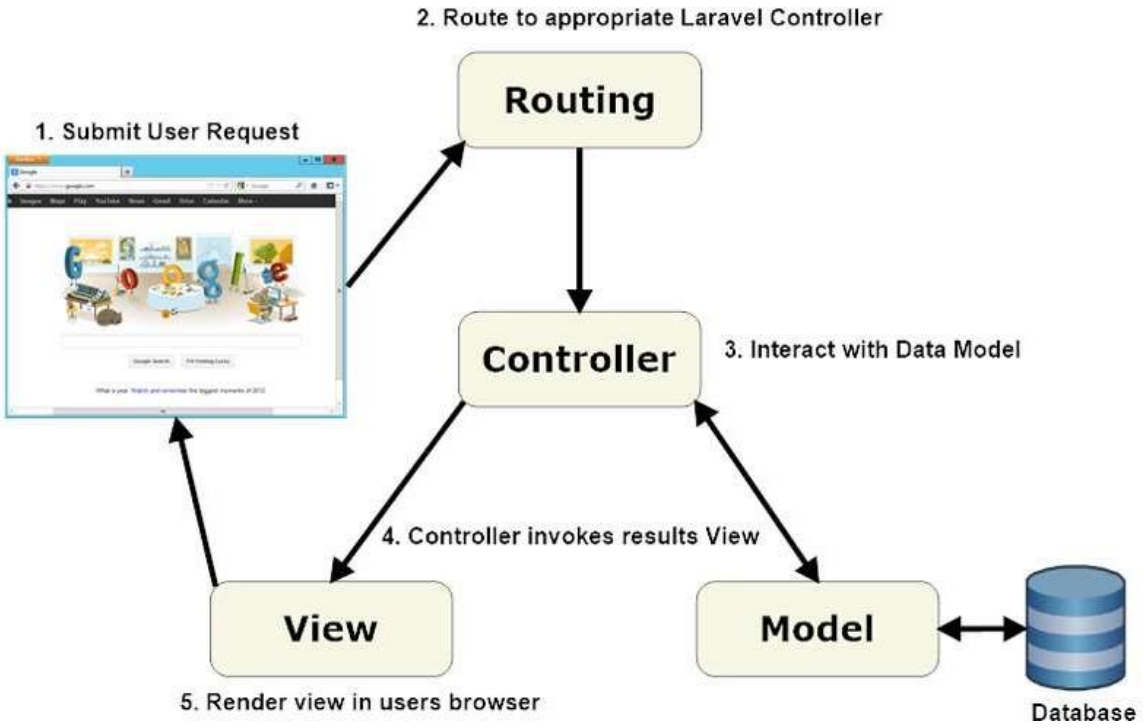
Dari susunan *view* dan *layout* di atas tampilan hasilnya adalah seperti berikut



2. *MODEL, CONTROLLER* DAN *MIGRATION* DALAM LARAVEL

Model dalam laravel mempunyai makna entitas yang berinteraksi dengan sumber data. *Model* berfungsi untuk *query* ke *database*, *insert* data baru, *update*, atau hapus *record database*. Semua itu dilakukan dengan ORM (*Object Relational Mapping*)

sehingga pada banyak kasus, kita tidak perlu menuliskan kode SQL secara langsung, akan tetapi langsung menggunakan method bawaan dari ORM seperti, **find, findOrFail, create, update, dll**. Dan ORM bawaan yang dipakai oleh Laravel adalah **Eloquent**.



Model dalam struktur *file* aplikasi Laravel terletak pada **app/models**. Untuk membuat *model* kita bisa ketikan perintah berikut ini pada **cmd atau terminal**.

```
php artisan make:model NamaModel
```

Misalkan nama *model* yang akan di bangun yaitu **Product**

```
php artisan make:model Product
```

Kita sudah membuat **Model Product**, akan tetapi **Model Product** belum bisa diakses oleh *User* karena belum ada *Controller*. *Controller* bertindak sebagai penghubung antara *model* dan *view*. *Controller* menerima *input* dari pengguna melalui rute (*route*) dan memprosesnya dengan menggunakan *model* yang sesuai. Setelah memproses data, *controller* mengirimkannya ke *view* yang tepat untuk ditampilkan kepada pengguna (*user*). Untuk membuat *controller* melalui artisan generator, perintahnya sebagai berikut:

```
php artisan make:controller NamaController
```

Misalkan nama *controller* yang akan di bangun yaitu **ProductController**

```
php artisan make:controller ProductController
```

Ketika sudah membuat *model* dan *controller*, jangan lupa untuk melakukan *migration database*. Fitur *database migration* dalam Laravel adalah semacam *tool* yang digunakan untuk manajemen versi *database* dari sebuah *project*. Untuk membuat *migration* perintahnya sebagai berikut:

```
php artisan make:migration create_NamaTabel_table
```

Contohnya semisal kita akan membuat tabel **product**, maka perintahnya:

```
php artisan make:migration create_products_table
```

Perintah di atas akan menghasilkan *file migration* yang terletak pada *folder* **database/migrations**. *File* tersebut memuat *class* yang meng-*extend* **class Migration** dari Laravel. Di dalam *class* tersebut terdapat **2 fungsi utama** yang **`up()`** dan **`down()`**. Fungsi **`up()`** akan meng*handle* aksi yang dilakukan dalam *migration*. Sedangkan fungsi **`down()`** akan membalikkan (*reverse*) aksi dalam *migration* tersebut.

Perintah untuk migrasi bisa juga kita *include*-kan bersamaan dengan pembuatan *model*. Caranya yaitu cukup menambah **-m di belakang nama *model***. Contohnya:

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

Jika perintah di atas berhasil dijalankan, maka kita akan mendapatkan 2 *file* baru yang berada di dalam *folder*:

- **App/Models/Post.php**
- **Database/migrations/2023_09_12_000000_create_posts_table.php**

Note: Tanggal yang muncul pada *file migration* sesuai dengan tanggal pembuatan migrasi.

3. *MENERAPKAN MODEL, CONTROLLER* DAN *MIGRATION* DALAM LARAVEL

a. Konfigurasi Koneksi *Database*

Konfigurasi ini digunakan untuk mengatur koneksi aplikasi dengan laravel kita ke dalam *database*. Untuk melakukan konfigurasi kita buka *file* **.env** terlebih dahulu dan ubah bagian kode berikut ini.

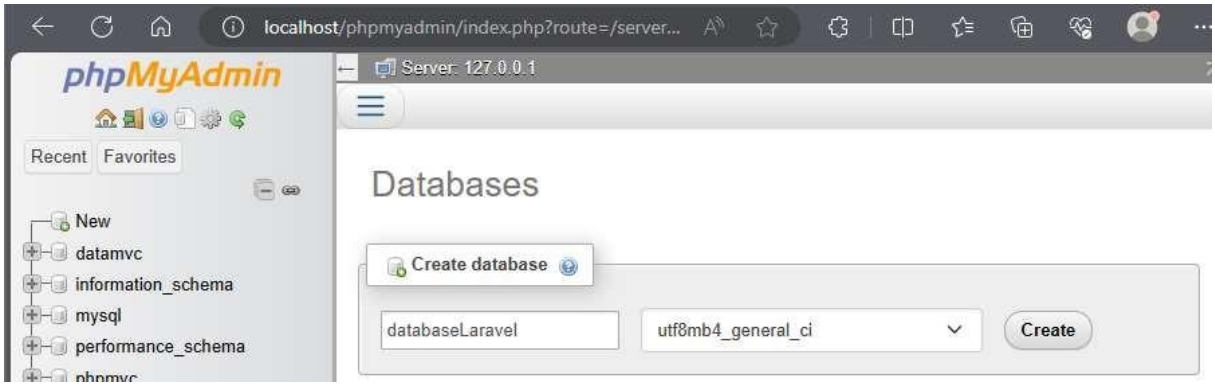
Edit bagian ini

```
14  DB_DATABASE=databaseLaravel
15  DB_USERNAME=root
16  DB_PASSWORD=
```

Note:

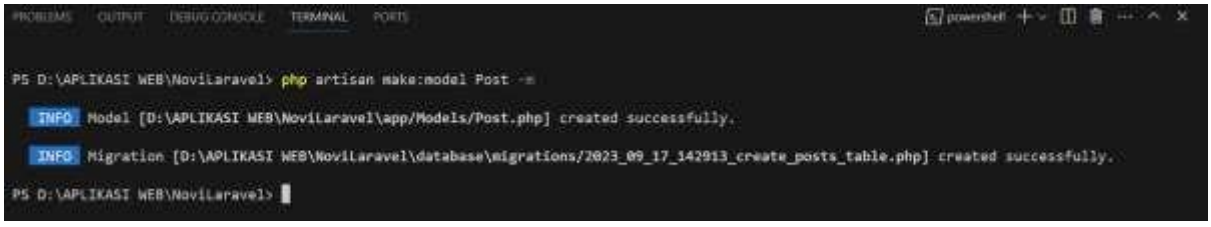
- **DB_DATABASE** isinya wajib disamakan dengan nama database yang akan atau telah dibuat.
- **DB_PASSWORD** silakan sesuaikam dengan konfigurasi MySQL-nya masing-masing

b. Membuat *Database*

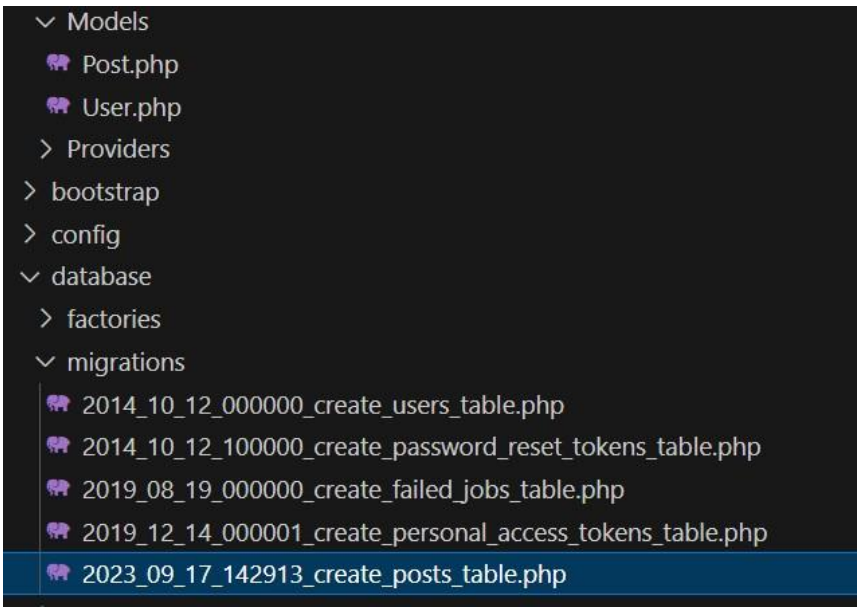


c. Membuat *Model* dan *Migration*

Kita akan membuat *model* baru dengan nama **Post** sekaligus membuat *file migration*-nya, maka perintahnya sebagai berikut:



Terjadi perubahan isi struktur *folder* Laravel, dimana pada *folder* **app/Models** akan tambah 1 *file* dengan nama **Post.php** dan pada *folder* **database/migrations** juga akan tambah 1 *file* bernama **2023_09_17_142913_create_posts_table.php**.



d. Menambahkan *Field* di Dalam *Migration*

File migrasi yang sudah dibuat kemudian berfungsi untuk menambahkan *field* atau kolom yang nanti ada dalam tabel dalam *database* kita. Kita buka *file migration*-nya (*file* saya bernama **2023_09_17_142913_create_posts_table.php**) yang terdapat dalam *folder* **database/migrations**, kemudian pada ***function up***, ubah kodenya menjadi seperti berikut ini:

```
12      public function up(): void
13      {
14          Schema::create('posts', function (Blueprint $table) {
15              $table->id();
16              $table->string('image');
17              $table->string('title');
18              $table->text('content');
19              $table->timestamps();
20          });
21      }
```

Dari perubahan kode di atas, kita menambahkan 3 *field* baru yaitu:

FIELD / COLUMN	TYPE DATA
`image`	`string`
`title`	`string`
`content`	`text`

e. Konfigurasi *Mass Assignment*

Field atau kolom yang sudah kita tambahkan di dalam *file migration* tidak akan pernah bisa menyimpan data ke dalam *database* sebelum kita melakukan *Mass Assignment* di dalam *model*. *Mass Assignment* sendiri merupakan sebuah *property* yang terdapat di dalam *model* dan digunakan untuk mengizinkan *field-field* melakukan manipulasi data ke dalam *database*, seperti *insert, update, delete*, dll.

Buka *file* **app/Models/Post.php**, kemudian ubah semua kodenya menjadi seperti berikut ini.

app > Models > Post.php > Post

```php
<?php

namespace App\Models;

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

class Post extends Model
{
    use HasFactory;
    /**
     * fillable
     *
     * @var array
     */
    protected $fillable = [
        'image',
        'title',
        'content',
    ];
}
```

Properti **`fillable`** pada **model Post** berfungsi agar dapat menyimpan data ke **tabel posts**. Isi dari *fillable* berupa *field-field* yang telah kita buat sebelumnya di dalam *Migration*.

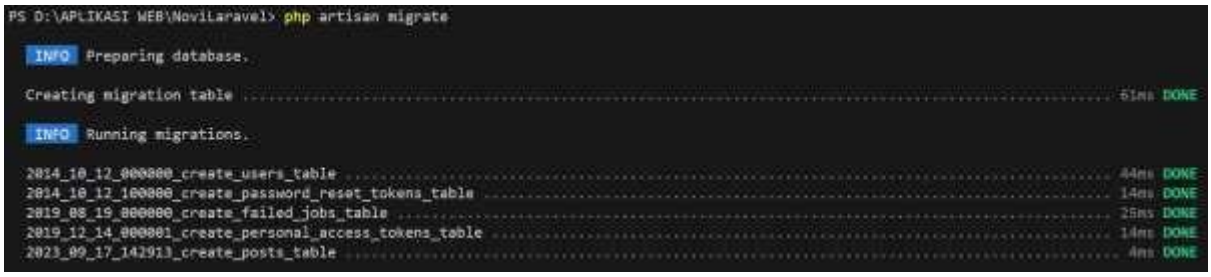
f. Menjalankan *Migration*

Perintah artisan untuk menjalankan *migration* pada Laravel adalah

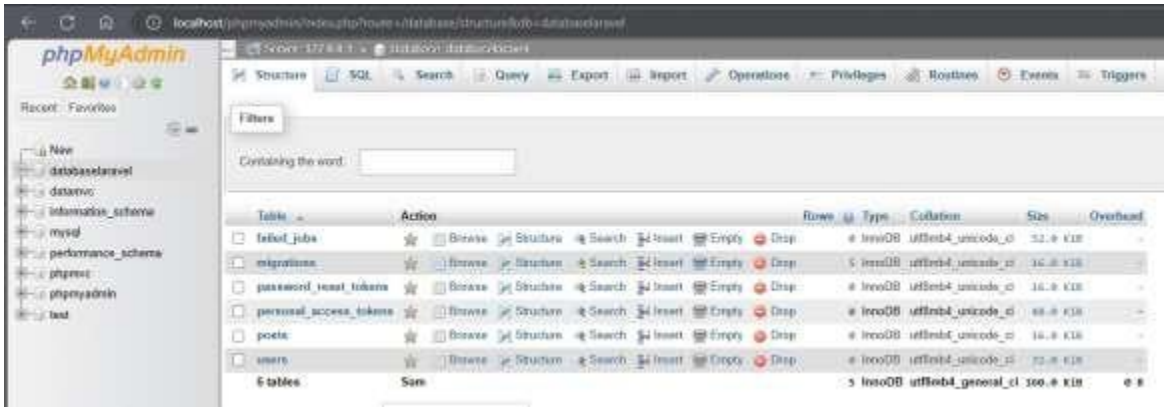
```
php artisan migrate
```

Perintah **`migrate`** berfungsi agar *field-field* yang terdapat di dalam *migration* di *generate* ke dalam *database*.

Jika perintah telah berhasil dijalankan, maka hasilnya sebegai berikut:



Dan pada *database* yang telah kita buat juga akan bertambah tabel-tabel yang telah di *migration*.



Sedangkan Untuk mundur/mengembalikan ke *migration* sebelumnya kita dapat menggunakan perintah berikut:

```
php artisan migrate:rollback
```

Atau kita juga dapat mundur/mengembalikan ke *migration* sebelumnya sebanyak langkah yang kita inginkan. Misal, kita ingin mundur ke *migration* 3 *step* dari *migration* terakhir, maka perintahnya adalah:

```
php artisan migrate:rollback --step=3
```

g. Membuat *Controller Post*

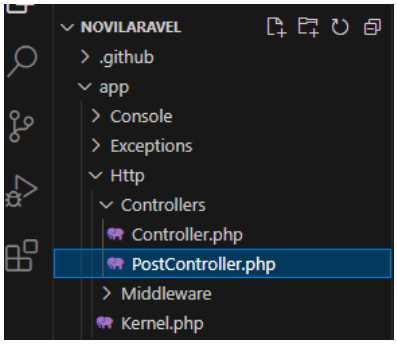
Kita ketikkan perintah berikut ini untuk membuat **controller post**.

```
PS D:\APLIKASI WEB\NoviLaravel> php artisan make:controller PostController
```

Notifikasi jika berhasil:

```
INFO  Controller [D:\APLIKASI WEB\NoviLaravel\app/Http/Controllers/PostController.php] created successfully.
```

Hasil dari perintah di atas yaitu *file controller* baru yang terdapat di dalam *folder* **app/Http/Controllers/PostController.php**



Selanjutnya buka *file* tersebut dan ubah kode programnya menjadi seperti berikut ini:

app > Http > Controllers > PostController.php > ...

```php
<?php

namespace App\Http\Controllers;

//import Model "Post"
use App\Models\Post;

//return type View
use Illuminate\View\View;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;

class PostController extends Controller
{
    /**
     * index
     *
     * @return View
     */
    public function index(): View
    {
        //get posts
        $posts = Post::latest()->paginate(5);

        //render view with posts
        return view('posts.index', compact('posts'));
    }
}
```

- **`use App\Models\Post;`** → untuk *import model* **Post** di dalam *Controller*
- **`use Illuminate\View\View;`** → untuk *import return Type* **View**
- Kemudian di dalam **`class PostController`** kita membuat *method* baru bernama **index**.

```
public function index(): View
```

- Di dalam ***method index***, kita membuat *variable* baru Bernama **$posts** dan berisi **model Post** yang mengambil data dari *database*. Kemudian kita memanggil **method latest** untu mengurutkan data yang akan ditampilkan berdasarkan yang paling terbaru dan membatasi data yang ditampilkan menggunakan **method paginate** sejumlah 5.

```
//get posts
        $posts = Post::latest()->paginate(5);
```

- Setelah data berhasil ditampung di dalam *variable* **$posts**, langkah selanjutnya adalah mengirimkan *variable* tersebut ke dalam *view* menggunakan **method compact**.

```
//render view with posts
        return view('posts.index', compact('posts'));
```

perintah di atas menandakan kita melakukan *render* ke dalam *view* **resources/posts/index.blade.php** dengan mengirimkan *variable* **$posts**.

h. Membuat *Route Posts*

Route digunakan agar kita bisa mengakses *controller* melalui URL browser. Untuk membuat *route*, kita buka *file* **routes/web.php**, kemudian ubah kodenya menjadi seperti berikut ini:

routes > web.php

```php
<?php

use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider and all of them will
| be assigned to the "web" middleware group. Make something great!
|
*/

//route resource
Route::resource('/posts', \App\Http\Controllers\PostController::class);
```

Dari perubahan kode di atas, kita akan menambahkan *route* baru dengan *path* **`/posts`** dan jenis *route* yang kita gunakan yaitu jenis **resource** yang bermakna bahwa *route-route* untuk proses CRUD akan di *generate* secara otomatis oleh Laravel, seperti *index, create, store, show, edit, update* dan *destroy* tanpa harus membuat satu persatu secara manual.

Untuk memastikan *route* berhasil atau tidak, ketikkan perintah berikut ini:

```
php artisan route:list
```

i. Membuat *View* dan Menampilkan Data

Kita buat *view* yang akan digunakan untuk menampilkan data. Pada *folder* **resources/views**, kita buat *folder* baru bernama **posts**, kemudian pada *folder* **posts** kita tambahkan *file* baru bernama **index.blade.php** dan masukkan kode berikut ini ke dalamnya.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>MyData</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.c
ss" rel="stylesheet" integrity="sha384-
T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN"
crossorigin="anonymous">

</head>
<body style="background: lightgray">

    <div class="container mt-5">
        <div class="row">
            <div class="col-md-12">
                <div>
                    <h3 class="text-center my-4">HALAMAN ADMIN</h3>
                    <hr>
                </div>
```

```
                <div class="card border-0 shadow-sm rounded">
                    <div class="card-body">
                        <a href="{{ route('posts.create') }}" class="btn
btn-md btn-success mb-3">TAMBAH POST</a>
                        <table class="table table-bordered">
                            <thead>
                              <tr>
                                <th scope="col">GAMBAR</th>
                                <th scope="col">JUDUL</th>
                                <th scope="col">CONTENT</th>
                                <th scope="col">AKSI</th>
                              </tr>
                            </thead>
                            <tbody>
                              @forelse ($posts as $post)
                                <tr>
                                    <td class="text-center">
                                        <img src="{{
asset('/storage/posts/'.$post->image) }}" class="rounded" style="width:
150px">
                                    </td>
                                    <td>{{ $post->title }}</td>
                                    <td>{!! $post->content !!}</td>
                                    <td class="text-center">
                                        <form onsubmit="return
confirm('Apakah Anda Yakin ?');" action="{{ route('posts.destroy', $post-
>id) }}" method="POST">
                                            <a href="{{ route('posts.show',
$post->id) }}" class="btn btn-sm btn-dark">SHOW</a>
                                            <a href="{{ route('posts.edit',
$post->id) }}" class="btn btn-sm btn-primary">EDIT</a>
                                            @csrf
                                            @method('DELETE')
                                            <button type="submit"
class="btn btn-sm btn-danger">HAPUS</button>
                                        </form>
                                    </td>
                                </tr>
                              @empty
                                  <div class="alert alert-danger">
                                      Data Post belum Tersedia.
                                  </div>
                              @endforelse
                            </tbody>
                        </table>
                        {{ $posts->links() }}
                    </div>
                </div>
```

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

</body>
</html>
```

Pada kode di atas, untuk menampilkan data maka kita bisa menggunakan sintaks **`@forelse`**.

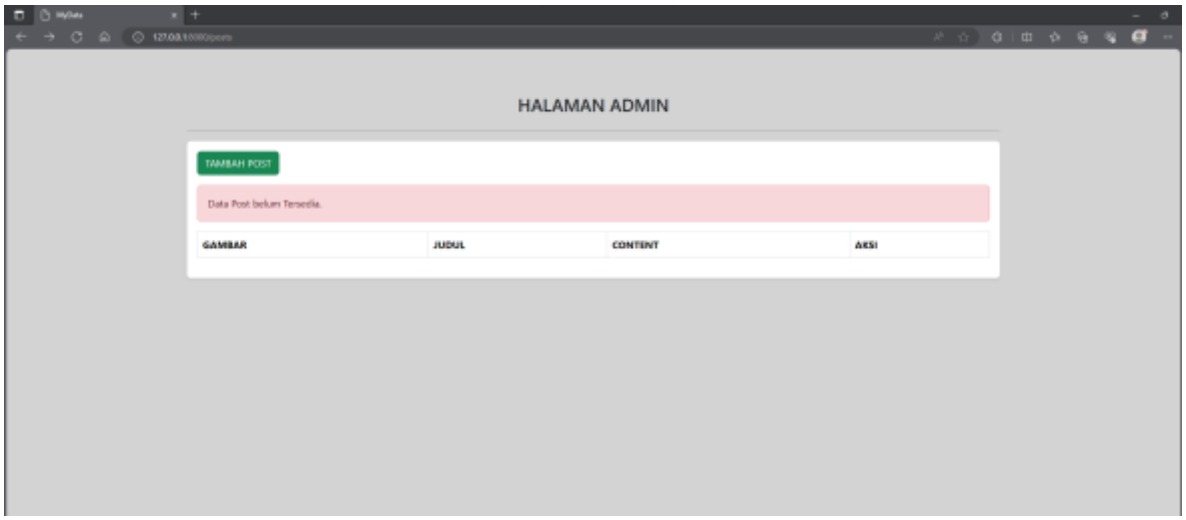
```
@forelse ($posts as $post)

//tampilkan data

@empty

//Data Post belum Tersedia.

@endforelse
```

Kemudian untuk menampilkan *pagination*, kita bisa memanggil method **`links`**.

```
{{ $posts->links() }}
```

j. Uji Coba Menampilkan Data

Untuk menguji hasilnya, silakan ketikkan http://localhost:8000/posts pada *address bar browser*.



Pada tampilan di atas terdapat pesan "**Data Post Belum Tersedia**" karena dalam **table posts** memang belum terdapat data sama sekali.

4. *INSERT DATA KEDATABASE*

a. Menambahkan Method Create dan Store

Untuk insert data pada database hal yang pertama dilakukan adalah menambahkan 2 method didalam controller, yaitu :

- *Function create* – method yang digunakan untuk menampilkan halaman form tambah data
- Function store – method yang digunakan untuk memproses data kedalam database dan juga melakukan upload gambar.

Buka file App\Http\Controllers\PostController, kemudian ubah kodenya seperti berikut ini.

```php
<?php

namespace App\Http\Controllers;

//import Model "Post
use App\Models\Post;

//return type View
use Illuminate\View\View;

//return type redirectResponse
use Illuminate\Http\RedirectResponse;

use Illuminate\Http\Request;

class PostController extends Controller
{
    public function index(): View
    {
        //get posts
        $posts = Post::latest()->paginate(5);

        //render view with posts
        return view('posts.index', compact('posts'));
    }
   public function create(): View
   {
       return view('posts.create');
   }



   public function store(Request $request): RedirectResponse
```

```
    {
        //validate form
        $this->validate($request, [
            'image'     => 'required|image|mimes:jpeg,jpg,png|max:2048',
            'title'     => 'required|min:5',
            'content'   => 'required|min:10'
        ]);

        //upload image
        $image = $request->file('image');
        $image->storeAs('public/posts', $image->hashName());

        //create post
        Post::create([
            'image'     => $image->hashName(),
            'title'     => $request->title,
            'content'   => $request->content
        ]);

        //redirect to index
           return redirect()->route('posts.index')->with(['success'    =>  'Data
Berhasil Disimpan!']);
    }
}
```

Dari perubahan kode diatas, pertama import return type *RedirectResponse*

```
//return type redirectResponse
use Illuminate\Http\RedirectResponse;
```

Kemudian menambahkan method baru yang Bernama *create*.

```
public function create(): View
    {
        //...
    }
```

Didalam method tersebut kita dapat melakukan return dengan mengarahkan pada sebuah view yang nanti akan kita buat yaitu *resource/views/posts/create.blade.php*.

Setelah itu membuat method yang Bernama *store*. Method ini digunakan untuk proses data dan memasukkannya kedalam database dan sekaligus membuat fungsi untuk melakukan upload gambar.

```
public function store(Request $request): RedirectResponse
    {
       //..
    }
```

Didalam method store yang dibuat pertama adalah validasi, fungsinya untuk memastikan data yang dikirimkan sudah sesuai dengan yang diharapkan.

```
//validate form
$this->validate($request, [
    'image'     => 'required|image|mimes:jpeg,jpg,png|max:2048',
    'title'     => 'required|min:5',
    'content'   => 'required|min:10'
]);
```

Dari validasi diatas, penjelasannya seperti dibawah.

KEY	VALIDATION	KETERANGAN
`image`	required	*field* wajib diisi.
	image	*field* harus berupa gambar
	mimes:jpeg,jpg,png	*filed* harus memiliki extensi `jpeg`, `jpg` dan `png`.
	max:2048	*field* maksimal berukuran 2048 Mb / 2Mb.
`title`	required	*field* wajib diisi.
	min:5	*field* minimal memiliki 5 karakter/huruf.
`content`	required	*field* wajib diisi.
	min:10	*field* minimal memiliki 10 karakter/huruf.

Jika data yang dikirimkan sudah sesuai dengan validasi diatas, Langkah selanjutnya yaitu melakukan upload gambar.

```
//upload image
$image = $request->file('image');
$image->storeAs('public/posts', $image->hashName());
```

Pada kode diatas, terdapat variable image dan berisi request dengan jenis file yang Bernama image. Request tersebuat merupakan file yang dikirim dari form. Setelah upload gambar menggunakan method *storeAs* bawaan Laravel maka file gambar yang diupload akan tersimpan dalam folder *storage/app/public/posts* dan nama dari file gambar tersebut akan dirandom menggunakan method *hashName().*

Jika gambar sudah berhasil diupload, Langkah selanjutnya yaitu melakukan insert data kedalam database, yaitu kedalam table posts.

```php
//create post
Post::create([
    'image'     => $image->hashName(),
    'title'     => $request->title,
    'content'   => $request->content
]);
```

Jika proses insert data berhasil dilakukan, maka akan diredirect atau diarahkan pada route yang Bernama posts.index dengan memberikan session flash yang memiliki key success dan isinya adalah Data berhasil disimpan!.

b. Membuat view form Create

Setelah selesai membuat controller, selanjutnya membuat view untuk menampilkan halaman tambah data post.

Buat file baru dengan nama *create.blade.php* dalam folder *resource/view/posts*. Kemudian masukkan kode program berikut.

```php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Tambah Data Post</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css
">
</head>
<body style="background: lightgray">

    <div class="container mt-5 mb-5">
        <div class="row">
            <div class="col-md-12">
                <div class="card border-0 shadow-sm rounded">
                    <div class="card-body">
                        <form action="{{ route('posts.store') }}"
method="POST" enctype="multipart/form-data">

                            @csrf

                            <div class="form-group">
                                <label class="font-weight-bold">GAMBAR</label>
                                <input type="file" class="form-control
@error('image') is-invalid @enderror" name="image">

                                <!-- error message untuk title -->
```

```
                                @error('image')
                                    <div class="alert alert-danger mt-2">
                                        {{ $message }}
                                    </div>
                                @enderror
                            </div>

                            <div class="form-group">
                                <label class="font-weight-bold">JUDUL</label>
                                <input type="text" class="form-control
@error('title') is-invalid @enderror" name="title" value="{{ old('title') }}"
placeholder="Masukkan Judul Post">

                                <!-- error message untuk title -->
                                @error('title')
                                    <div class="alert alert-danger mt-2">
                                        {{ $message }}
                                    </div>
                                @enderror
                            </div>

                            <div class="form-group">
                                <label class="font-weight-bold">KONTEN</label>
                                <textarea class="form-control
@error('content') is-invalid @enderror" name="content" rows="5"
placeholder="Masukkan Konten Post">{{ old('content') }}</textarea>

                                <!-- error message untuk content -->
                                @error('content')
                                    <div class="alert alert-danger mt-2">
                                        {{ $message }}
                                    </div>
                                @enderror
                            </div>

                            <button type="submit" class="btn btn-md btn-
primary">SIMPAN</button>
                            <button type="reset" class="btn btn-md btn-
warning">RESET</button>

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

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.min.js"></scri
pt>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"><
/script>
<script src="https://cdn.ckeditor.com/4.13.1/standard/ckeditor.js"></script>
<script>
    CKEDITOR.replace( 'content' );
</script>
</body>
</html>
```

Dari penambahan kode diatas, dibuat form yang mana untuk actionnya diarahkan dalam route yang bernama posts.store. Route tersebut akan memanggil method dicontroller yang Bernama store.

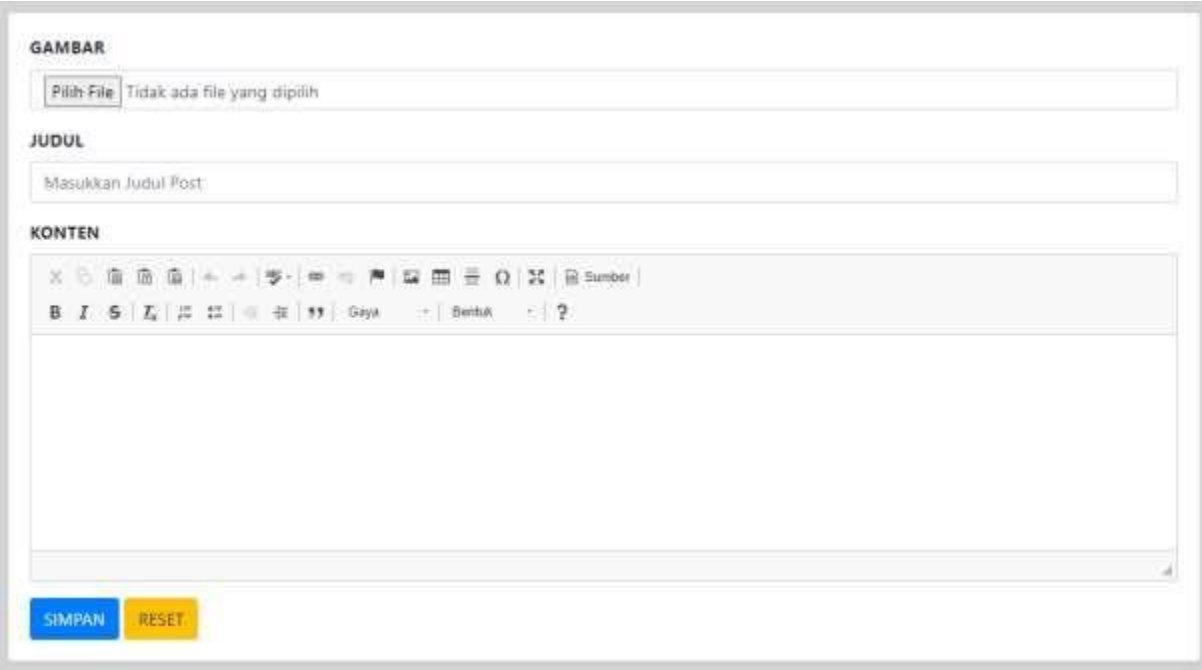
```
<form action="{{ route('posts.store') }}" method="POST"
enctype="multipart/form-data">

..

</form>
```

Form attribute enctype="multipart/form-data digunakan untuk upload file didalam formnya. Kemudian didalam Javascript perlu diperhatikan kita melakukan inialisasi Rich Text Editor menggunakan **CKEdtor.** Dan akan diterpakan pada textarea yang memiliki name content.

```
<script>
    CKEDITOR.replace( 'content' );
</script>
```

c. Uji Coba Insert Data Post

Silahkan jalankan server Laravel kemudian akses URL http://127.0.0.1:8000/posts/create, jika berhasil maka akan tampil halaman berikut



Kemudian masukkan data – data yang diinginkan, kemudian klik button simpan, jika berhasil maka akan diarahkan pada halaman posts index dengan alert sukses tambah data.

E. TUGAS

1. Bacalah dengan seksama materi ini. Jika ada yang kurang paham, silakan bertanya kepada guru.
2. Buatlah struktur web kalian (di sesuaikan dengan tema). Bisa mencontoh di link ini.
3. Buatlah view untuk web blog kalian menggunakan layouting seperti halaman guru.
4. Editlah *admin page* yang sudah kalian buat dengan menerapkan *blade templating layouting* Laravel. Harap di sesuaikan dengan tema *web* masing-masing.

5. Buatlah 1 halaman untuk tampilan *front end web* yang akan berkaitan langsung dengan *end user*. Dimana *front end web* ini akan menampilkan setiap data yang kita inputkan melalui *admin page*. *View Front end web* nya harap di sesuaikan dengan tema kalian masing-masing.

☺☺☺☺☺ SELAMAT BELAJAR ☺☺☺☺☺