# Struktur folder routes dan view

Routes/web.php

Route::get('/', function () {

    return view('welcome');

});

Sintax diatas berarti url(‘/’) memangil resource (welcome.blade.php)

Mencoba route dengan mengembalka nilai :

Route::get('/', function () {

    return view('welcome');

});

Route::get('/about', function () {

    return 'Andri Firmansyah Putra';

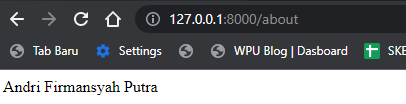
});

Route::get('/pekerjaan', function () {

    return 'Mahasiswa';

});

Coba kita lihat pada brow ser untuk memangilnya :



jika memangil sebuah view maka gunakan (buat file about.blade.php)

Route::get('/about', function () {

    return view('about');

});

Menggunakn foilder public :

* Menerapkan ccss, maka buat file css pada folder public lalu file dalam flder views tingal memangi nama filenya saja

## Mengirimkan sebuah data pada route :

Route::get('/about', function () {

    return view('about', [

        'nama' => 'Andri Firmansyah Putra',

        'alamat' => 'Mrisi Tirtinirmolo Kasihan Bantul',

        'img' => 'imgs/allTeam.jpeg'

    ]);

});

pada view :

<h3><?= $nama ?></h3>

<h4>{{ $alamat }}</h4>

<img src="<?= $img ?>" alt="<?= $img ?>">

# Blade Templatering engine

Supaya kita bias menggunakan template/ bagian html secara berulang jika bemiliki latar halaman yang sama kita dapat memanfaatkan blade templating.

* Kita akan memindahkan beberapa komponen htmlv: header , komponen html lain pada [home.blade.php](http://home.blade.php)
* Buat folder **view/layout/main.blade.php** untuk menyimpan
* Kita copy dari [home.blade.html](http://home.blade.html) ke kedalam layout/main.html lalu hapus isi utama halaman (container halaman) bagian yang berbeda dari tiap tiap halaman.

<div class="container mt-3">

        @yield('container')

    </div>

* Yield nanti akan dapat digantikan dengan tuisan lain
* Sekarang kita dapat menghapus html lain yang ada pada halaman lain (cth: about.blade.) dan hanya penrlu mengisikan isi dari layout utama yg sudah dibuat , sekarang bagaimana cara agar dapat menggunakan  **html utama**  yang menggunakan yield
* Membuat link(**extends**) dan memberikan bagian yg bias diid(**section**)

@extends('layouts.main')

@section('container')

    <div class="container mt-3">

        <h1>Halaman Home</h1>

    </div>

@endsection

## Memasukkan halaman html secara langsung

* Kita akan memisahkan bagian navigasi pada halaman layout utama(layout/main.blade.php)
* Membuat folder baru dan membuat file navigasi yang didalam sudah ada sintax” untuk membuat navigation
* Melakukan include ke file navigasi (dalam file utama) menggunakan include
* @include('partials.navbar')

## Rouute parameter

-pada web.php

Route::get('posts/{slug}', function ($slug) {

    $new\_post = [];

    foreach ($blog\_posts as $post) {

        if ($post['slug'] === $slug) {

            $new\_post = $post;

        }

    }

    // return $slug;

    return view('post', $new\_post);

});

* Pada html

# Model post Controller

Membuat model : **php artisan make: model nama\_model**

* **Jika sudah memilkikitable dapatjugauntukmembuatmigraton nya**

## Latihan Modeldengan data static

* B- membuat model (akan disavepada app/model)

**Php artisan make:model post**

* Lalu kita isi dan beberapa kita hapus (karena masih belum digunakan)

<?php

namespace App\Models;

class Post

{

    private static $blog\_post =  [

        [

            'title' => 'Judul Post Pertama',

            'slug' => 'judul-post-pertama',

            'author' => 'Andri Firmansyah Putra',

            'body' => 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Ad Nemo sunt temporibus illo odit alias deleniti, officia explicabo quis? Consequuntur tempora in totam.'

        ], [

            'title' => 'Judul Post Kedua',

            'slug' => 'judul-post-kedua',

            'author' => 'Agung Jiwandono',

            'body' => 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Ad voluptatum itaque doloribus optio, obcaecati culpa sit, sint architecto suscipit nemo, fugit incidunt totam.'

        ],

    ];

}

* Ceritanya nanti kita akan mengambil data dengan method all pada model, maka tambahkan function pada post model
* Berikut kode pada route :

Route::get('/blog', function () {

    return view('posts', [

        'title' => 'Blog Posts',

        'posts' => Post::all()

    ]);

});

**Kode pada** model :

## Membuat pencarian post dengan model

Route::get('posts/{slug}', function ($slug) {

    $new\_post = Post::find($slug);

    // return $slug;

    return view('post', $new\_post);

});

Function model yang dibuat

public static function find($slug)

    {

        $posts = self::$blog\_post;

        $new\_post = [];

        foreach ($posts as $post) {

            if ($post['slug'] === $slug) {

                $new\_post = $post;

            }

        }

        return $new\_post;

    }

## Menggunakan Array Collection :

Tinggal bungkus menggunakan collect :

$collect  = collect([1,2,3]);

public static function all()

    {

        // return self::$blog\_post; 5.5 tidak digunakan karena menggunakan collection

        return collect(self::$blog\_post);

    }

Mengubah sintax didalam post::find

$posts = static::all();

        $new\_post = $posts->firstWhere('slug', $slug);

        return $new\_post;

## Menggunakan Controller

Php artisan make:controller name

use Illuminate\Http\Request;

class PostController extends Controller

{

}

* Kita akan mengunah halaman post agar dapat menggunakan controller

use App\Models\Post;

class PostController extends Controller

{

    public function index()

    {

        return view('posts', [

            'title' => 'Blog Posts',

            'posts' => Post::all()

        ]);

    }

}

Route yang digunakan menjadi :

Route::get('/blog', [PostController::class, 'index']);

# 6 Database Migration dan Eloquent

File .env (file konfigurasi)

-didalamlaravel terdapat table table yang sudah disediakan dan dapat kita gunakan dengan melakukan :

**Php artisan migrate**

**File tersebut ada karena ada data** php yang kita sebut dengan migrasi pada setiap table yang telah dibuat, maka dari itu sekarang kita akan membuat data base dan mengkonfogurasi pada laravel pada file (**.env**)

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=wpu\_blog

DB\_USERNAME=root

DB\_PASSWORD=

**Lalu jalankan sintax migrate diatas pada terminal**

## Melakukanroleback pada migrasi

**Php artisan migrate:rollback ::** sintax ini digunakan untuk

## Melakukan hapus dan migrate sekaligs

Php artisan migrate:fresh

## Eloquent ORM

Digunakan untuk kita dapat menggunakan database. Sebuah fungsi untuk memetakan object untuk dapat berinmteaksi dengan database. Perantara (model) ke dalam database.

#CTT : setiap model pasti ada file migrationnya.

### File Model

#model digunakan untuk referensi table didalamdatabase

Pada protected adalah file yang boileh diisi

protected $fillable = [

        'name',

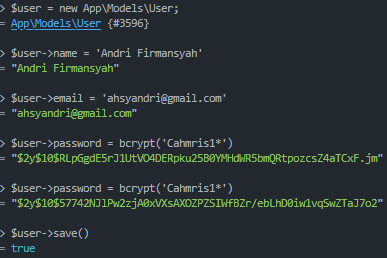
        'email',

        'password',

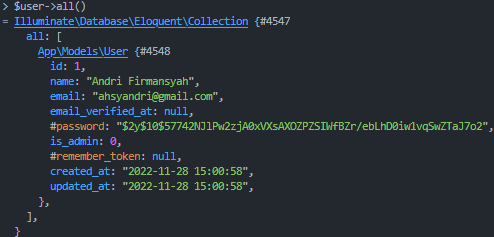
    ];

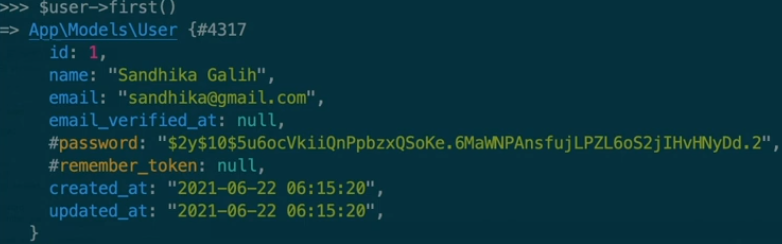
### Menggunakan artisan tinker

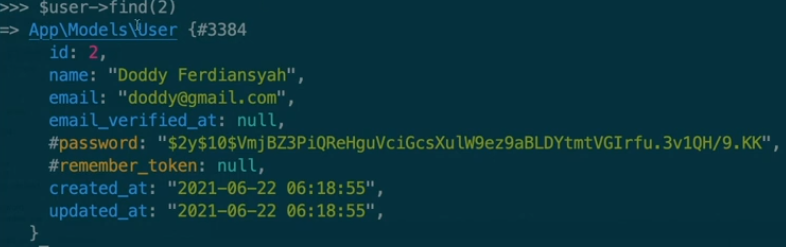
Menggunakan fitur ini untuk mengakses model dan mengisi database;

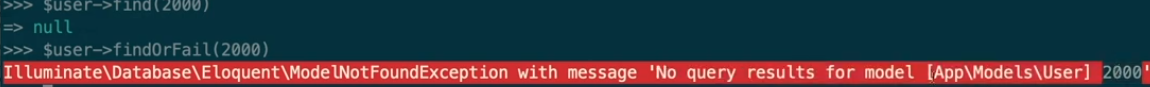


### Sintaxmilix eloquent







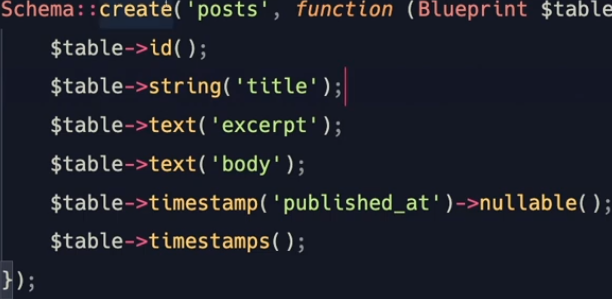


# 7 Membuat Model Post

## Membuat model post

-php artisan make:model –m Post (Membuat model dan file migrasi)

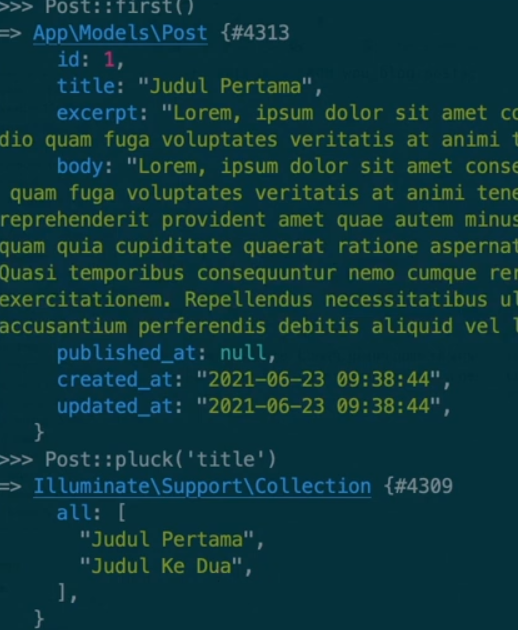
- memebnuhi file migratuuin posts



**#sintax $table->timestamps (merupakanmethod untuk menuliskan) delete at dan update add**

**-**agar dapat menambah menjadi table sekarang klakukan

**Php artisan migrate:fresh**

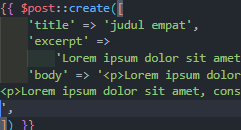
****

### Menampilkan post dengan format html

<p> {!! $post->body !!} </p>

### Cara lain menambahkan file data kedallam database dengan tinker

* Menggunakan method create



* Namun sebelum dijalankan akan muncul error masassigment

## Menghadapi error mass assignment

kita perlu untuk mendeskripsikan file diperbolehkan masuk kedalam database kedalam file model Post

protected $fillable = ['title', 'excerpt', 'body']; (ini atau yang bawah)

    protected $guard = ['id'];

## MassAssigment Update





## Orute model binding

Ini akan mengh-skip query dan akan langsungmengembalikan datanya, sebelumnya kita menggunakan id namun kita gunakan didalamfungsi yaitu tipedatanya ($nama \_model dan variable sesuai dengan route)



### Mengubah dengan model binding

Route::get('posts/{id}', [PostController::class, 'show']);

Menjadi

Route::get('posts/post}', [PostController::class, 'show']);

Pada Controller

public function show($id)

    {

        $new\_post = Post::find($id);

        return view('post', [

            'title' => 'Single post',

            'post' => $new\_post

        ]);

    }

Menjadi

public function show(Post $post) {

        return view('post', [

            'title' => 'Single post',

            'post' => $post

        ]);

    }

## Menambahkan slug

* Mengedit filemigrasi

Schema::create('posts', function (Blueprint $table) {

            $table->id();

            $table->string('title');

            $table->string('slug')->unique();

            $table->text('excerpt');

            $table->text('body');

            $table->timestamp('published\_at')->nullable();

            $table->timestamps();

        });

* Lakukan artisan migrate:fresh
* Sekarang datra kita sudah huilang
* Lakukan penambahan data

### Menganti pencarian tidak lagi dengan id tapi slug

* Melakukan pencarian dengan slug

<a href="/posts/{{ $post->id }}">{{ $post->title }}</a>

* Diganti
* <a href="/posts/{{ $post->slug }}">{{ $post->title }}</a>
* Mengubah route

Route::get('posts/{post:slug}', [PostController::class, 'show']);

# 8 Post Category $ Eloquent Relationship

Hubungan antar model

* Kita akan membuat kategori untuk setiap blog
* Membuat model dan migration Kategroi

Php artisan make:model –m Category

-membuatscemauntuk file migration cetegories

Schema::create('categories', function (Blueprint $table) {

            $table->id();

            $table->string('name')->unique();

            $table->string('slug')->unique();

            $table->timestamps();

        });

## Menguhubungkan keterhubungan post dengan category

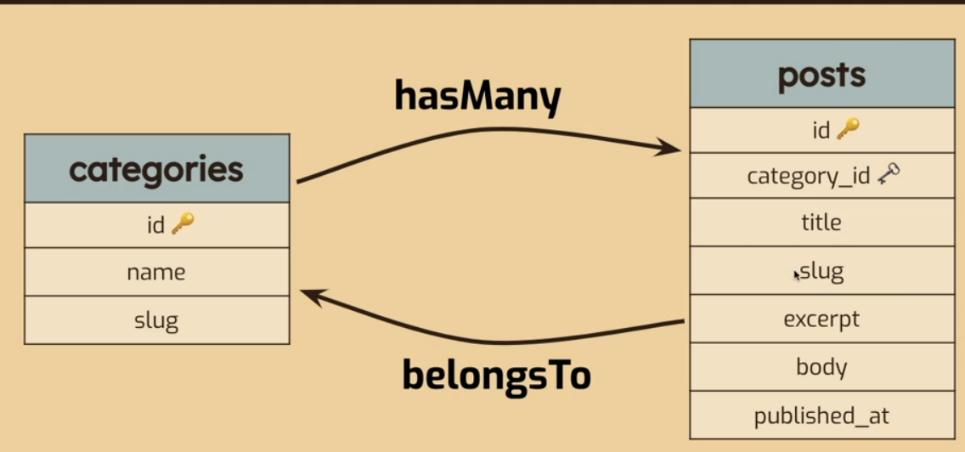
* Kita akan menggunakan foregnkey didalam post
* Tambakan field baru didalam migrasiposts

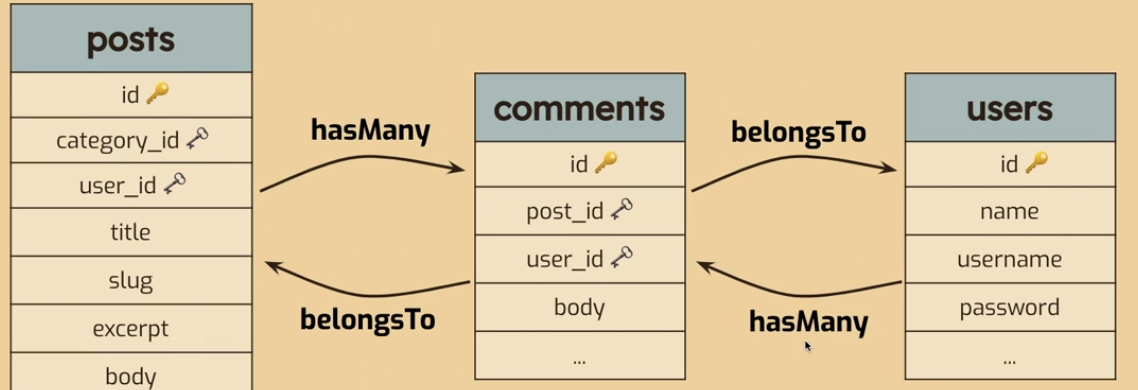
### Kardinalitas post dan category

One to many (karena pastinya satu kategori akanbanyakmemilikli banyak posts)

### Kardinaliotas invers post categor

One to one (karena satu post hanya bias memliki 1 category)





### Method penghubung keterhubungan

Menghubuungkan model post dengan model category dengan membuat method dengan nama sama dengan tablenya. Lalu menggunakan method kardinalitas sesuai dengan yang ada diatas.

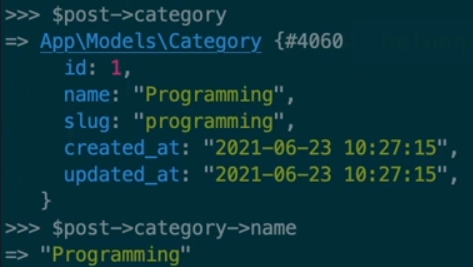
* Membuat hubungan pada model post

public function category(){

        return $this->belongsTo(category::class);

    }

* Coba menggunakan pada thinker



Edit pada tampilan(post.blade.php)

<p>category {{ $post->category->name }}</p>

## Membuat agar category yang di click akan menam[ilkan seluh post terkait dengan category

* Membuat link pada html

<p>category : <a href="/categories/{{ $post->category->slug }}">{{ $post->category->name }}</a></p>

### Membuat route dansudah ada fungsi didalamnya

Route::get('categories/{category:slug}', function (Category $category) {

    return view('category', [

        'title' => $category->name,

        'posts' => $category->posts,

        'category' => $category->name

    ]);

});

* Membuat tamilan category

### Membuat relasi category – post

Buka model category lalu tambahan buat method

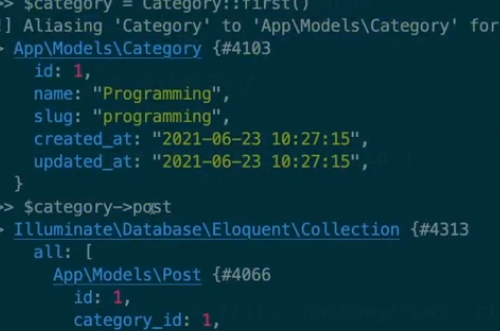
public function post()

    {

        return $this->hasMany(Post::class);

    }

Mencoba method relasicategory



## Menampilkan semua data category

* Membuat route

Route::get('/categories', function () {

    return view('categories', [

        'title' => 'Post Categories',

        'categories' => Category::all()

    ]);

});

Tampilan categories.blade.php

 @foreach ($categories as $category)

            <li>

                <h2><a href="/categories/{{ $category->slug }}">{{ $category->name }}</a></h2>

            </li>

        @endforeach

# 9 membuat seeder

Seeder digunakan untuk menambahkan data ked b secara otomatis dan ditambahkan pada (app\database\seeder)

Dan menjalankan php artisan db:seed

* Namun sebelumnya telah diisikan datanya pada file seeder

User::create([

            'name' => 'Adi Firmansyah',

            'email' => 'ahsyadri@gmail.com',

            'password' => bcrypt('12345')

        ]);

        Category::create([

            'name' => 'Personal',

            'slug' => 'personal'

        ]);

# 10 Factory dan faker

Digunakan untuk memasukkan data otomatis secara banyak, factory tergantung pada setiap kelas/model yang dipasang, sedangkan faker merupakan alat untuk mencari data secara random

* Jika kita perhatikan pada setiap model menggunakan

class Post extends Model

{

    use HasFactory;

dan contoh dari penggunaan factory yg ada pada seeder

\App\Models\User::factory(10)->create();

## Dan template contoh (App/database\factory\UserFactory.php)

 public function definition()

    {

        return [

            'name' => $this->faker->name(),

            'email' => $this->faker->unique()->safeEmail(),

            'email\_verified\_at' => now(),

            'password' => '$2y$10$92IXUNpkjO0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', // password

            'remember\_token' => Str::random(10),

        ];

    }

### Untuk membuat factory :

**Php artisan make:factory PostFactory**

### Menggunakan faker b\_indo

* Config/ap.php
* Pada config : (kita akan menggunakan supaya merujuk pada file env)

'faker\_locale' => env('FAKER\_LOCALE', 'en\_US'),

* Pada .env
* FAKER\_LOCALE=id\_ID

Karena ada file seed kita maka kita hapus : user:create dan lainnya karena kita akan menggunakn factory

## Latihan

* Membuat post factory

**Php artisan make:factory PostFactory**

return [

            'title' => $this->faker->sentence(mt\_rand(2, 8)),

            'slug' => $this->faker->slug(),

            'excerpt' => $this->faker->paragraph(),

            'body' => $this->faker->paragraph(mt\_rand(5, 10)),

            'user' => 1,

            'category\_id' => 1

        ];

* Menambahkan pada seeder

\App\Models\User::factory(5)->create();

\App\Models\Post::factory(20)->create();

## Catatan eloquent

Ketika kita membuat eloquent pada model ke model yang lain kita cukup membuat function hubungan sesuai dengan class, lalu laravel akan merujuk sesuai dengan ddeskripsi code yang kita sampaikan misalkan : kita punya kelas post yang terhubung ke kelas user maka fungsi pada model seperti berikut :

public function user()              //cek user\_id   kita ganti dengannama lain

    {

        return $this->belongsTo(user::class);

    }

Fungsi tersebut akan akan memberiktahukan eloquent bahwa yang digunakan untuk relasi adalah user\_id lalu bagaimana kita membuat fungsi jika kita mendefinisikan function dengan nama yang jauh berbeda dari field nama kita yang sebagai foregnkey ke kelas user yanitu dengan seperti berikut :

public function author()              //cek user\_id   kita ganti dengannama lain

    {

        return $this->belongsTo(user::class,'user\_id');

    }

}

# 11 N + 1 Problem

Hal ini terjadi ketika pemangilan ke dala database, dan pemanggian itu terjadisecara berulang ulang. Contohnya pada saat kita menampilkan data dengan perulangan terjadi pada saat kita meangil table lain :

@foreach ($posts as $post)

        <article class="mb-5 border-bottom">

            <h2>

                {{-- <a href="/posts/{{ $post['id'] }}">{{ $post['title'] }}</a> --}}

                <a href="/posts/{{ $post->slug }}" class="text-decoration-none">{{ $post->title }}</a>

            </h2>

            {{-- <h5>By: {{ $post['author'] }}</h5> --}}

            <p>By <a href="/author/{{ $post->author->name }}">{{ $post->author->name }}</a> in

                <a href="/categories/{{ $post->category->slug }}

Kebutuhan table lain yaitu pada post yang memangil data user, sehingga didalam perulangan terjadi query lagi (pemanggilandalam database bekali kali sesuai dengan data post) itlah yang disebut dengan **N +1 PROBLEM**

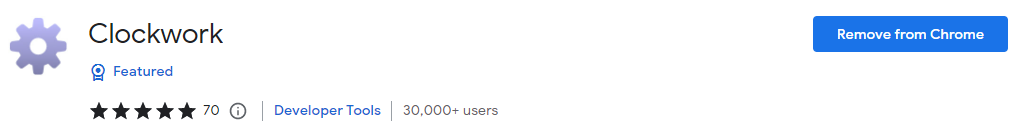
Didalam laravel hal diatas disebut degan lazy loading(ketika dibutuhkan baru dilakukan). Hal tersebut bias menjadi bagus dan tidak bagus jika kasusnya sepertii diatas.

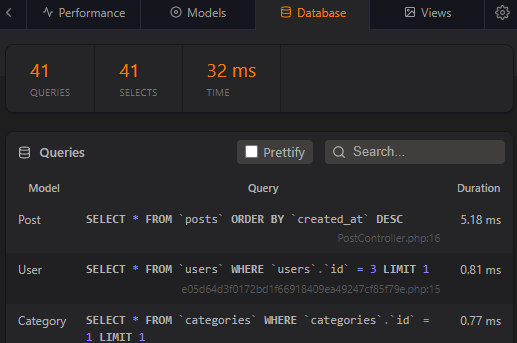
## Mengistal plugin untuk mengetahi performa dalam pemanggilan terhadap database

* Install clockwork :

**composer require itsgoingd/clockwork**

* Nstal didalam google crome





Terjadi 41 query

## Eager Loading

Pada saat kita mengakses relationship didalam eloquent, maka modelnya akan melakukan lazy eager loading. Yang berarti data relasi tidak akan dipanggil sampai kita memangil propertinya pada saat kita looping . tetapi kita bisameminta agar relasi melakuan eager loading pada parentnya. (pada kasus kita sehingga dapat mengquery langsug )

Caranya dengan menggunakan width :

### Menerapkan eager load pada poat all

 public function index()

    {

        return view('posts', [

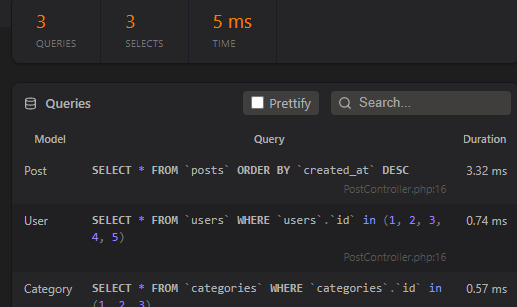
            'title' => 'All Posts',

            'posts' => Post::with(['author','category'])->latest()->get()

        ]);

        // return Post::latest()->get();

    }



### Eager Load pada Model binding

Disini kita akan menggunakanlazy eager loading . digunakan pada saat setelah parent sudah didapatkan. Dengan menggunakan load contoh :



### Eager load pada kasus post kita

Route::get('author/{user:name}', function (User $user) {

    return view('posts', [

        'title' => 'Posts by : '.$user->name,

        'posts' => $user->posts,

    ]);

});

# 13 remake

## Membuat eager load didalam model

Code yang akan kita ubah :

'posts' => Post::with(['author','category'])->latest()->get()

Kita kita hanya perlu menambahkan variable dibawah didalam model

protected $with = ['category','author'];

sehingga sekarang code untuk dapat data post sekarang sekarang :

# 14 search and pagination

* Membuat form untuk pencarian

## Percobaan menangkap variable request

* Mencoba dengan nilaiyang ditentukan didalam request

<http://127.0.0.1:8000/blog?search=iasdjiasjed>

* Code didalam controller

dd(request('search'));

* Update postsController apa bila ada pencarian

public function index()

    {

        $posts = Post::latest()->get();

        // dd(request('search'));

        if(request('search')){

            $posts->where('title','like','%'.request('search').'%');

        }

        return view('posts', [

            'title' => 'All Posts',

            'active' => 'Blog Posts',

            'posts' => $posts

        ]);

    }

* Menabahkan variable pencarian

 public function index()

    {

        $posts = Post::latest();

        if(request('search')){

            $posts->where('title','like','%'. request('search') .'%')

            ->orWhere('body','like','%'.request('search').'%');

        }

        return view('posts', [

            'title' => 'All Posts',

            'active' => 'Blog Posts',

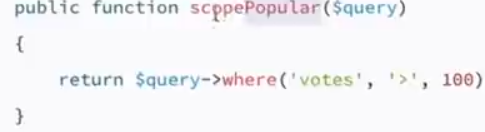
            'posts' => $posts->get()

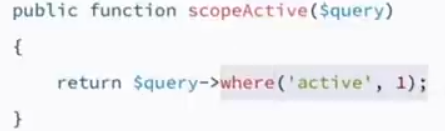
        ]);

    }

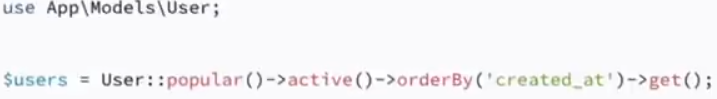
### Local Scope

Sekarang kita akan memindahkan apa yang telah kita kerjakan kedalam model, kita akan meggunakan local scope yang berguna untuk menggunaan kembali query query kedalam aplikasi : contoh :





Penggunaan



### Menerapkan local scope

* Membuat funcsi scope kedalam model post

public function scopeFilter($query){

        if(request('search')){

            return $query->where('title','like','%'. request('search') .'%')

            ->orWhere('body','like','%'.request('search').'%');

        }

    }

* Mengedit function pada post controller

 public function index()

    {

        $posts = Post::latest();

        return view('posts', [

            'title' => 'All Posts',

            'active' => 'Blog Posts',

            'posts' => $posts->filter()->get()

        ]);

    }

### Merapikan localscope pada post

Kita akan mengubahnya menjadi : request pada local scope akan dikerjakan oleh controller dan data quesy dikerjaan didalam model

public function index()

    {

        $posts = Post::latest();

        return view('posts', [

            'title' => 'All Posts',

            'active' => 'Blog Posts',

            'posts' => $posts->filter(request(['search']))->get()

        ]);

    }

 public function scopeFilter($query, array $filters){

        if(isset($filters['search']) ? $filters['search'] : false){

            return $query->where('title','like','%'. $filters['search'] .'%')

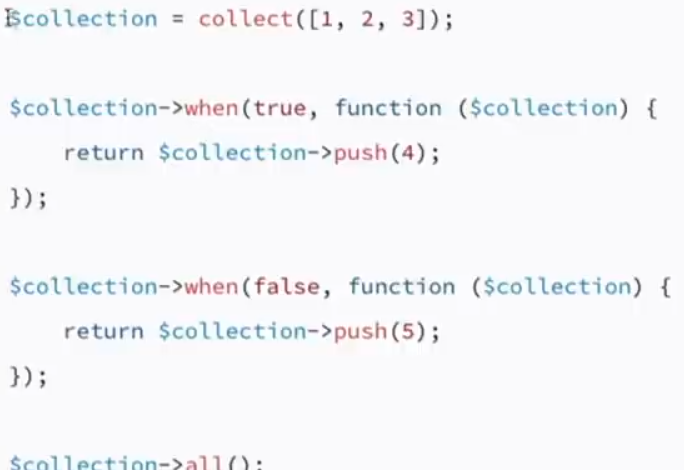
            ->orWhere('body','like','%'.$filters['search'].'%');

        }

    }

### Merapikan localscope pada post 2

Jika kita lihat code kita diats kita akan engantinya dengan method kita dengan menggunakan when:

Ehen argument akan dijalankan ketika first argument erisi true contoh :

Ketika sudah berisi true kita akan bias melakukan sesuatu : code ini akan terasa enak ketika kita sudah memiliki banyak.

### Menggunakan null coalesing

Pada code :

isset($filters['search']) ? $filters['search'] : false

kita akan emngganti code diatas (fitur pada php 7): yaitu sintantik sugar yang daapt digunakan saat menggunaan ternary dan juga digunakan untuk mengece isset : jadi code diatas menjadi

$filters['search']) ?? false

Kode dalam scopeFilters menjadi seperti ini :

Didalam function dalam when : kita menempatkan $query, dan Boolean Dari search

 $query->when($filters['search']) ?? false, function($query,$search) {

            return $query->where('title','like','%'. $search .'%')

            ->orWhere('body','like','%'.$search.'%');

        });

Sekarang code scopefilters milik model post :

$query->when(($filters['search']) ?? false, function($query,$search) {

            return $query->where('title','like','%'. $search .'%')

            ->orWhere('body','like','%'.$search.'%');

        });

## Permasalahan pencarian

Sekarang kita sudah dapat menggunakan fitur untuk melakuka pencarian, namun sekarang kita ada beberapa masalah yang perlukita cari untuk membenarkannya yaitu pada fitur pencarian kita view yang digunakan adalah posts yaitu view yang digunakan untuk tampilan post, category dan user saat kita menjalankan fitur pencarian kita akan mencari ke semua postingan namu hal yang kita ingginkan hal yang lebih spesifik, misalkan saat kita membuat apostigan dari suatu kategori lalu menggunakan pencarian makan akan dicarikan postingan dalam category tertentu.

## Pemecahan

Maka dari itu kita akan menanbahkan parameter lain dalam request untuk melakjukan pencarian :

**Category=** [**http://127.0.0.1:8000/posts?search=Aperiam&category=personal**](http://127.0.0.1:8000/posts?search=Aperiam&category=personal)

Maka kita akan mengubahh codingan post controller dan scope kita yang telah kita buat sebelumnya

Kita akan mengembaliklan data yang di join table ke table category, karena kita akan mencari postingan yang dicari tetapi dia juga merupakan bagian dari category (menggunakan **whereHas**) :

Where has mempunyai sebuah callback lagi \

$query->when(isset($filters['category']) ? $filters['search'] : false, function($query,$category) {

            return $query->whereHas('category',function($query){

                 return $query->where('slug',$category);

            });

        });

Kode diatas masih salah karena $category sebagai callback didak dapat digunakan. Atau kita masukkan kedalam callback **whereHas**

**Cara agar dapat memakai USE**

$query->when(isset($filters['category']) ? $filters['search'] : false, function($query,$category) {

            return $query->whereHas('category',function($query) use ($category){

                 return $query->where('slug',$category);

            });

        });

Sekarang kita dapat melakukan filter dengan 2 kondisi

## Mengubah url apliakasi agar dapat menambah filter pencarian

Menambahkan hidden :

<form action="/posts">

            @if (request('category'))

                <input type="hidden" name="category" value="{{ request('category') }}">

            @endif

            @if (request('author'))

                <input type="hidden" name="author" value="{{ request('author') }}">

            @endif

            <div class="input-group mb-3">

                <input type="text" class="form-control" placeholder="Search..." name="search" value="{{ request('search') }}">

                <button class="btn btn-danger">Search</button>

            </div>

        </form>

## Pagination

'posts' => $posts->filter(request(['search','category','author']))->paginate()

{{ $posts->links()}}

App\Providers\AppServiceProvider

Paginator::useBootstrap();

## Permasalahan pagination

Saat ini pagination yang kita gunakan masih belum dapat melakukan pagination dalam keadaan tertentu. Misalkan saat kita membuka halaman dengan get ke category lalu ke page selanjutnya akan menyebaab kan semua postingan tampil bukan karena category terkait namun akan menampikan semua postingan di halaman selanjutnya. Cara memperbaikinya kita akan mengaitkan. Mengatasinya kita dengan membaw3a quesy string sebelmnya menggunakan method ini.

'posts' => $posts->filter(request(['search','category','author']))->paginate()->withQueryString()

# 15 View login & registration

Menggunakan icon

Memasang cdn icon bootstrap

Membuat route login

Membuat controller ->load pada web.php

Membuat view (kita gunakan yang dari bootstrap example kiita download lalu kita pakai pada halaman login)

Agar terlihat sempurna kita akan memasukkan file css dari bootstrap kedalam aplikasi kita

(copy css bawaan formnya saja)->buat folder didalam public/css

Lakukan import pada file css pada vvview utama kita(layouts/main.blade.php)

# 16 User registration

* Membuat method pada form :

<form action="/register" method="POST">

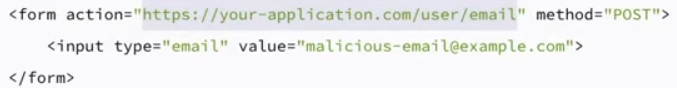
* Membuat route saat untuk menangani aksi dan ethod diatas

Route::post('register',[RegisterController::class,'store']);

* Membuat function store pada controller register
* Mencoba menampikan data yang dikirimklan dari form

## Muncul error page expired

Ini dikarenakan laravel mengamankan POST melalui form, pada laravel terdapat CSRF dimana itu adalah semacam serangan yang memalsukan request di wesite, menggunakan URL yang sesuai yang padahal uru bukan dari web kita tapi dari web lain. Misalkan mengirimkan data jahat tapi urlnya ke web kita :



Sehingga kita menjaga

### Prevennting CSRF Request

Dengan lini kita akan menjaga data masuk hanya dari web kita menggunakan ***csrf token***. Cara kerja csrf token saat laravel akan mengenerate token dan mencocokan token didalam request.

### Menggunakan csrf token



Kita cukup menggunakan method **@csrf** yang secara tidak alangsung laravekl akan memberikan sebuah input vbertipe hidden nilai dari csrf

* Setelah menambahkan method **@csrf** kita kedalam
* Mencoba menampilkan request yang dikirimkan

public function store(){

        return request()->all();

    }

* Cara kedua

public function store(Request $request){

        return $request->all();

    }

## Membuat validasi form

Contoh :



* Kita praktikan terhadap studi kasus kita
* Kita akn mencoba jika

public function store(Request $request){

        // return $request->all();

        $request->validate([

            'name' => 'required|max:255',

            'username' => ['required','min:3','unique:users'],

            'email' => 'required|email|unique:users',

            'password' => 'required|min:5|max:255'

        ]);

        dd('registrasi berhasil!!');

    }

### Validasi alert

Pada kodingan diatas akan muncukl “**registrasi berhasil**” ketika data byang dikirimkan s8udah sesuai dengan ketentuan validasi. Namun pada kpdingan kita sebelumnya kita bel;um dpat mengetahui jika misalkan pada request data selanjutnya bisasaja terjadi kesalahan namun kita tidak tau kesalahan apa yang tel;ah terjadi. Maka dari itu kita kita akan mencoba menampikan pemberitahuan kesalahan dari request yang dikirimkan .

Kita akan menampilkan error melalui css pada input contoh **@error(‘variabel’)@enderror**:

<input type="text" class="form-control rounded-top @error('name') is-invalid @enderror" id="name" placeholder="name" name="name">



Menampilkan pesan error dalam Dario validasi yang telah kita buat

@error('email')

                    <div class="invalid-feedback mb-2">

                        {{ $message }}

                      </div>

                    @enderror

## Menyimpan kedalam database

public function store(Request $request){

        // return $request->all();

        $validatedData = $request->validate([

            'name' => 'required|max:255',

            'username' => ['required','min:3','unique:users'],

            'email' => 'required|email|unique:users',

            'password' => 'required|min:5|max:255'

        ]);

        // dd('registrasi berhasil!!');

        // $validatedData['pasword'] = bcrypt($validatedData['password']);

        $validatedData['pasword'] = Hash::make($validatedData['password']);

        User::create($validatedData);

        return redirect('/login');

    }

### Memberikan flash message saat ke halaman login

Ini akan memberikan tanda jika kit ate;ah berhasil melakukan registrasi, lalu kit akan kembali ke halaman login setelah itu barulah kita memberikan pesan bahwa registrasi berhadil pada halamn login, caranya dengan memanfaatkan sintax dibawah sebelum melakukan redirect ke halaman login

$request->session()->flash('success','Registrasion SuccessFull! Please Login');

### Menangkap session flash message

Pada index.blade.php milik login lakukan :

 @if (session()->has('success'))

                <div class="alert alert-success alert-dismissible fade show" role="alert">

                    <strong>{{ session('success') }}</strong>

                    <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>

                  </div>

                @endif

# 17 user login middleware

Didalam laravel terdapat fitur fitur yang dapat menangani autentikasi, seperti laravel breze tetapi saat ini kita akan menggunakan fitur laravel dari sebuah kelas yang bernama facades

* Kehalaman logon/index.bklade
* Memperbaiki form(menambahkan method action dan @csrf)

<form action="/login" method="post">

@csrf

* Menambahkan route

Route::post('login',[LoginController::class,'authenticate']);

* Membuat function pada controoler tertuju(LoginController)

public function authenticate(Request $request){

        $validatedData = $request->validate([

            'email'=>'required|email:dns',//dns= depannya hars ada {.com} nya

            'password'=>'required'

        ]);

        if(Auth::attempt($validatedData)){

            $request->session()->regenerate();

            return redirect()->intended('/dashboard');

        }

// return back()->withError('loginError','Login Error');

        return back()->with('loginError','Login Error');

    }

#catatan :

* Auth::attempt : digunakan untuk mencocokan
* $request->session->regenerate() : digunakan utk menghindari teknik hacking fixsession
* Redirect()->intended(‘/dashboard’) = intended digunakan agar saat menuju kesuat7 halaman dengan redirect dapat melalui middleware terlebih dahulu
* Jika attempt tidak berhasil maka akan kembali dengan membawa pesan error
* withError = akan mengembalikan nilainya dalam variable error
* with = digunakan untuk mengembalikan pesan melalui flash message
* jangan lupa memasang flash message (loginError)
* membuat dashboardController
* membuat route mengarah kedashboard
* membuat controller untuk dashboard dan viewnya

### Permasalahan

Saat ini setelah melakukan aksi login saat in masih dapat kembali ke halaman login lagi dan kita masih bel;um ada cara agar dapat logout. Untuk dapat melakukan itu kita mestinya fahan dengan yang dinamakan sebuah middleware authentication.

## Middleware

Diguankan untuk cek autentifikasi. Dengan memfilter request dai http.misalkan user akan diautentidfikasi apakah seorang user sudah melakukan autentikasi apa belum jika belum maka aplikasi akan melakukan redirect(pemindahal) kehalaman login dan jika sudah melakukan sebuah autentikasi maka aplikasi akan mengijinkan masuk kedalam aplikasi.

* Kita dapat menggunakan middleware pada route login (berjalan ditengah tengah(antara http dan aksi(cxxcontroller::index))

Global middleware

Middleware yang otomatis berjalan setiap request http dijalankan pada App.middleware.kernrl ada auth,guest dll

### Menbcoba middleware

Kita akan menerapkan pada halaman login

Route::get('login',[LoginController::class,'index'])->middleware('guest');

* Jika kita sudah melakukan autentikasi jika kita mengunjungi halaman login akan diarahkan ke halaman **(/home)**
* Kita akan mengubah pengatura default tersebut (App\Http\profiders\Route Service provider kita ubah

public const HOME = '/home';

* Menjadi

public const HOME = '/';

## blade middleware

@auth

                    <li class="nav-item dropdown">

                        <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button"

                            data-bs-toggle="dropdown" aria-expanded="false">

                            {{auth()->user()->name}}

                        </a>

                        <ul class="dropdown-menu" aria-labelledby="navbarDropdown">

                            <li><a class="dropdown-item" href="/dashboard"><i class="bi bi-layout-sidebar-inset"></i>My Dashboard</a></li>

                            <li>

                                <hr class="dropdown-divider">

                            </li>

                            <li><a class="dropdown-item" href="#"><i class="bi bi-box-arrow-right">Logout</i></a></li>

                        </ul>

                    </li>

                @else

                    <li class="nav-item">

                        <a class="nav-link {{ $active === 'login' ? 'active' : '' }}" href="/login">

                            <i class="bi bi-box-arrow-in-right"></i>Login

                        </a>

                    </li>

                @endauth

## Logout

<li>

   <form action="/logout" method="post">

   @csrf

   <button class="dropdown-item"><i class="bi bi-box-arrow-right">Logout</i></button>

   </form>

</li>

public function logout(Request $request){

        Auth::logout();

        $request->session()->invalidate();

        $request->session()->regenerateToken();

        return redirect('/');

    }

Invalid dated : supaya tidak bias dipakai

RegenerateTOken : bikin baru supaya tidak dibajak

## Perbaiki route dengan middleware

Route::get('login',[LoginController::class,'index'])->middleware('guest');

Route::post('login',[LoginController::class,'authenticate']);

Route::post('logout',[LoginController::class,'logout']);

Route::get('register',[RegisterController::class,'index'])->middleware('guest');

Route::post('register',[RegisterController::class,'store']);

Route::get('dashboard',[DashboardController::class,'index'])->middleware('auth');

## Peermasalahan route middleware

Saat ini kita telah menambahkan sebuah route middleware, namun jika kita belum melakukan autentikasi maka laravel akan mengarahkan pada routename login, yang menjadi masalah saat ini kita tidak ada route dengan nama login, maka kita akan memenamai rote http(/login sebagai routename login)

Route::get('login',[LoginController::class,'index'])->name('login')->middleware('guest');

Ini ada di kebijakan **App\htp\middleware\autenticate.php**  yaitu kelas yang mengatasi jika tidak terdapat autentikasi

class Authenticate extends Middleware

{

    /\*\*

     \* Get the path the user should be redirected to when they are not authenticated.

     \*

     \* @param  \Illuminate\Http\Request  $request

     \* @return string|null

     \*/

    protected function redirectTo($request)

    {

        if (! $request->expectsJson()) {

            return route('login');

        }

    }

}

# 18 Dashboard UI

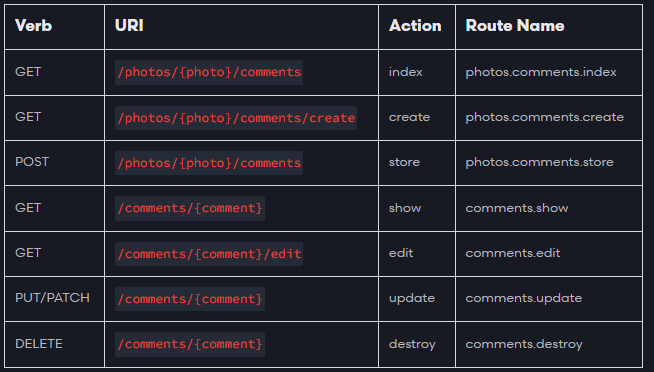
Kita akan menggunakan fasilitas dari bootstrap dengan template dashboard yang tersedia. :

* Pindahkan folder css dan javascript dashboard dari bootstrap pada folder public
* - mwenfgedit tampilan pada

## Mulai Menggunakan resource xontroller

Yakni resource yang digunakan untuk mengelolka crud,

**Php artisan make:conctoller namacontroller –resource**

****

**Semua diatas dapat ditangani hanya dengan :**

**use App\Http\Controllers\CommentController;**

**Route::resource('photos.comments', CommentController::class)->shallow();**

* Kita akan Menambahkan controller baru untuk dapat pegggelolaan halaman dashboard
* Kita akan melakukan pembuatan controller dengan memuat resource dan menghubungkan dengan model post

**php artisan make:controller DashboardPostController --resource --model=post**

* Kita sudah dibuatkan beberapa fungsi didalam conrooler tsbt. Dan sekarang kita perlu mennambahkan routenya

Route::resource('/dashboard/posts', DashboardPostController::class);

Dalam resource secara default untk menampilkan suatu data(method show) dalam menggunakan fitur resource masih menggunakan pencarian menggunkan id sekarang bagaimana kita mengatasi jika pencarian tidak menggunakan data id.

* Yaitu dengan membuiat slug sebagai nilai pencarian dengan memanfaatkan function : get route leyname
* Pasang pada model :

public function getRouteKeyName()

    {

        return 'slug';

    }

* Sekarang kita bias menggunakan slug

<a href="/dashboard/posts/{{ $post->slug }}" class="badge bg-info text-decoration-none">lihat</a>

   public function show(Post $post)

    {

        return view('dashboard.posts.show',[

            'post' => $post

        ]);

    }

# 19 create form

<form method="post" action="/dashoard/posts">

Ini nanri secara otomatis dihandle oleh dashboardPostsControoler method store

* Menginstal eloquest sluggable

**composer require cviebrock/eloquent-sluggable**

### memasangsluggable

* Karena kita akan menggunakan pada pembuatan data post maka kita memasangnya pada
* Model post lau use dan jangan lupa import
* Membuat functuion **sluggable**

use Cviebrock\EloquentSluggable\Sluggable;

class Post extends Model

{

    use Sluggable;

    /\*\*

     \* Return the sluggable configuration array for this model.

     \*

     \* @return array

     \*/

    public function sluggable(): array

    {

        return [

            'slug' => [

                'source' => 'title'

            ]

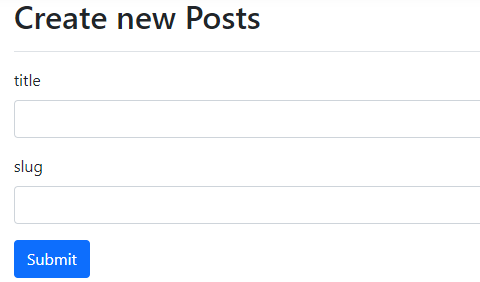
        ];

    }

}

### Menggunakan sluggable

Pada kasus kali ini kita akan menggunkannya pada saat kita mengetikkan title post pada form (menggunakan js)



  const title = document.querySelector('#title');

  const slug = document.querySelector('#slug');

  title.addEventListener('change',function(){

    fetch('dashboard/posts/createSlug?title=' + title.value)

    .then(response => response.json())  //kita ambil isinya |responsenya kita jalankan dimethod json| json masih (promise)

    .then(data => slug.value = data.slug)//then lagi |misal kembalian adl data | lalu data akan menganti slug value (inputan slug)

  })

Membuat route

Route::get('/dashboard/posts/checkSlug',[DashboardPostController::class,'checkSlug'])->middleware('auth');

Membuat function pada controller

public function checkSlug(request $request){

    $slug = SlugService::createSlug(Post::class, 'slug', $request->title);

    }

Use

: use \Cviebrock\EloquentSluggable\Services\SlugService;

public function checkSlug(request $request)

    {

        $slug = SlugService::createSlug(Post::class, 'slug', $request->title);

        return response()->json(['slug' => $slug]);

    }

## Menggunakan editor untuk membuat body

Kunjungi : <https://github.com/basecamp/trix>

* Kita akan menggunakn trix ini dengan beberrapa tahap
* Unduh trix editor dari link diatas
* Mengimpoert css dan js trix yg sudah dipindahkan kedalam laravel
* Mengunakan tag dari trix
* Membuat sebuah input dengan type hidden
* Pasang pada main.blade,.php pada head

{{-- trix Editor --}}

<link rel="stylesheet" type="text/css" href="https://unpkg.com/trix@2.0.0/dist/trix.css">

<script type="text/javascript" src="https://unpkg.com/trix@2.0.0/dist/trix.umd.min.js"></script>

<style>

  trix-toolbar [data-trix-button-group="file-tools"]{

    display: none;

  }

</style>

Pasang pada create.blade.php

<div class="mb-3">

      <label for="body" class="form-label">Body</label>

      <input type="hidden" id="body" name="body">

      <trix-editor input="body"></trix-editor>

    </div>