1 ROUTES AND VIEWS

1. Routes

a. adalah jalur/alur sistem aplikasi web kita

b. Located in ; app/

c. routes --> api.php

--> console.php

--> web.php --> web routes for your application

a. contoh isi web.php:

Routes::get('/',function)

{

return view('welcome');

}

// '/' --> adalah nama urlnya

// function --> fungsi mentah untuk mengeksekusi isinya

// web --> method untuk menuju folder resources/views/ dan mengeksekusi welcome.blade.php

// 'welcome' --> nama blade

2. View

a. Guna : Buat nyimpan template2 / file HTML nyari

b. Located in : resources/views

2 CONTROLLERS AND BLADES

1. Controller :

a. Jembatan apakah ada data yang diolah /dimainin dulu(dari routes ke view biasanya)

b. Located in : app/Http/Controller

c. Membuat controller baru : php artisan make:controller nama\_controllernya

d. contoh dasar:

1. di routes :

Route::get('/blog', 'BlogController@index');

// 'blog' --> nama urlnya

// 'BlogController' --> nama file controllernya

// '@index' --> nama method controller yang akan digunakan

2. di controller :

class BlogController extends Controller

{

public function index()

{

return view ('blog/home')

}

}

e. Contoh ngoper data : exp: blog artikel ke 1,2,3,dst

1. di routes :

Route::get('blog/{id}', 'BlogController@show');

//{id} --> parameter acuan untuk dilempar ke Controller (tanpa $)

2. di controller :

public function show($id)

{

$nilai = 'ini adalah link ke' . $id;

$user = 'hammam islami';

return view('blog/single', //file view yang dituju

['isi' => '$nilai' , 'user' => '$user' ] );

//isi dan user --> nama buat dimasukin ke view

//$nilai dan $user --> isi yang akan dimasukin

}

3. di view : single.blade.php

<h2> {{isi}} selamat datang </h2>

<h3> {{user}} </h3>

3 SISTEM TEMPLATING BLADES

1. What is?

--> its templating engine provided with laravel

--> Blades view files use the .blade.php file extension and are typically stored in the resources/vies directory

2. What it can do?

A. Extending A Layout

a. di layout/master.blade.php : //isi yang selalu sama taro sini

<html>

meta2 codingnya blabla

<title> @yield('title') </title>

// @yield --> penanda letaknya

// 'title' --> nama yieldnya untuk diakses oleh @section

<body>

.............

@yield('content')

.............

</body>

</html>

b. di home.blade.php: //isi yang beda

@extends('layout.master')

//extends master.blade.php(dipisahin . bukan / )

@section('title', 'Blog sekolah coding')

// parameter 1 : nama @yield di master.blade.php

// parameter 2 ; isinya

@section('content')

<h1> Selamat datang dan datang disini </hi>

@endsection //jika isi section panjang pake cara ini

B. Displaying Unescaped Data

-->By default, Blade {{....}} statement are automatically sent through PHP's htmlspecialchars function to prevent XSS attack

--> to disable it : {!!..........!!}

C. Conditional Statement and Loops

C.1. If..elseif..else.. :

@if (count($record) === 1)

<p> I have one record </p>

@elseif(count($record) > 1)

<p> I have multiple record </p>

@else

nothing

@endif

@switch($i)

@case(1)

blabla

@break

@default

blabla

@endswitch

C.2 Loop statement:

@for( $i=0; $i<10; $i++ )

<p> angka {{$i}} </p>

@endfor

@while(true)

I'm looping forever

@endwhile

@foreach ( $users as $user )

This is {{$user->id}}

@endforeach

@forelse ( $users as $user )

C.3. Switch statement :

<li> {{$user->name}} </li>

@empty

<p> no user </p>

@endforelse

C.4. plain PHP

@php

............

@endphp

4 QUERY BUILDER

1. biar bisa make metode DB:

1. liat ke config/app.php , use Illuminate\Support\Facades\DB::class ke controller yang akan diisi command DB

2. setting database :

a. ada di pusat(prioritas utama) .env --> DB\_CONNECTION = nama databasenya, DB\_DATABASE, DB\_PASSWORD, etc

jika tidak di set dia menuju:

b. config/database.php

2. akses DB:

1. restart servernya dengan : php artisan serve dlu

3. command2:

1. di route :

Route::get('/blog/{id}','BlogController@show');

2. di function show() pada cotroller BlogController :

ngambil semua datanya : $var = DB::table('nama\_tables')->get();

//maybe //ini sama kaya $var = select \* from nama\_tables;

return view('blog/single',['users' =>$var] );

//oper variable $var dengan nama users yang berisi isi database ke blade

3. di bladenya:

@foreach($nama\_tables as $nama\_table)

<li> {{ $nama\_table-->name }} </li>

// $nama\_table disini adalah object, name adalah attributenya

@endforeach

4. cara debug : dd($var\_yang\_ingin\_di\_die\_dump) di controller, exp: dd($nama\_table s)

5. others :

a. $users = DB::table('users')->where('nama','hammam')->get();

->where('nama','like','%a%')->get();

b. DB::table('users')->insert( [['nama' =>'islami', 'password' => '123'], ['nama' =>'yaho', 'password' => '122']] )

// masukin2 data

$users = DB::table('users')->get();

c. DB::table('user')->where('nama','hammam')->update(['nama'=>'heuheu']);

5 ELOQUENT

1. Elequent from laravel make u interact with database easily,which is each table has own "Model" class

2. Langkah2 :

Setting awal Model :

1. bwt model: php artisan make:model nama\_model //located in app/ jika banyak bisa bikin folder Models di app/

dan pada file Model yang kita buat namespacesnya diganti jadi namespace App\Models;

2. Nama class Model yg dibuat akan otomatis mencari table dengan nama pluralnya, misal class Model Blog akan otomatis mencari table ‘blogs’

jika namanya beda di class Modelnya kasih keterangan :

protected $table = 'nama\_table\_yang\_diinginkan';

3. eloquent expect that every table has created\_at and updated\_at coloum (kolom untuk melihat data dibuat dan diupdate)jika kolom tsb nggk ada beri perintah :

public $timestamps = false;

4. di contoller : use Modelnya , contoh : use app/Blog;

Coding 1:

1. di route : Route::get('/home','BlogController@index');

2. di model : udah disinngung diatas diatas

3. di fungsi index():

$blogs = Blog::all(); //ngambil semua yang ada di table blogs

return view('blog/home' ['blogs']=> $blogs);

4. di view blog/home :

@foreach($blogs as $blog)

<li> {{$blog->title}} </li>

@endforeach

Coding 2 dengan parameter id:

1. di route : get('/blog/{id}','BlogController@show');

2. di BlogController ;

3. di controller :

1. di fungsi show():

$blog = Blog::find($id); //ngambil coloum spesifik di table blogs

return view('blog/single',['blog' => $blog] );

2. di view single:

<h2> {{blog->description}} </h2>

3. jika dari view home ke single ada linknya, pake perintah ini di view home :

<li> <a href = "blog/single/{{$blog->id}}"> {{$blogs->title}} </a> </li>

6 CRUD ELOQUENT

Insert:

1. cara biasa :

$blog = new Blog; //membuat instance table tanpa()

$blog->title = 'halo cimahi';

$blog->description = 'isi dari halo cimahi';

$blog->save(); //akhiri dengan nama\_var->save() untuk menyimpan

2. mass assignment :

Blog::create

([

'title'=>'halo', //setiap argument pisahin dengan koma

'description' => 'isi dari halo bekasi',

]);

3. white list and black list: //di Modelnya dalam kasus ini dalam class Blog

a. guna : untuk ngebolehin apa saja yang boleh diubah dalam table tersebut

b. ada 2, white list dan blacklist (pilih salah satu):

a. white list :

protected fillable = ['title','description']; //hanya 2 kolom itu yang boleh diubah

b. black list :

protected guarded = [] //nggk ada black listnya

protected guarded = ['title','description'];

Update

1. cara biasa:

Blog::where('title' , 'halo cicurug')->first(); //yang pertama ditemukan

$blog = Blog::where('title' , 'halo cicurug');

$blog->title = 'halo sukabumi';

$blog->save();

2. mass assignment:

Blog::find(9)->update

([

'title' => 'halo lampung';

'description' => 'isi dari halo lampung';

]);

Delete

1. cara biasa:

$blog = Blog::find(1);

$blog->delete();

2. mass assigment:

Blog::destroy(2); //2 itu idnya pake cara ini jika cma 1 yang ingin dihapus

Blog::destroy([8,9]); //jika banyak yang ingin dihapus

7 404 AND SOFT DELETE

1. antisipasi error jika halaman tidak ada :

1. di fungsi show() // fungsi yang ada parameter idnya:

$blog = Blog::find(1); //nyari idnya

if (!$blog)

abort(404);

2. di folder resources/error/ :

buat file 404.blade.php dan isi dengan apa yang terjadi jika error

2. Soft Delete

a. konsep mirip recycle bin, yaitu data tidak benar2 dihapus, tapi dia masuk ke kolom lain(deleted\_at)

b. langkah2 :

1. buat kolom deleted\_at dengan type timestamp dan set NULL

2. di Modelnya :

di luar class : use Illuminate\Database\Eloquent\SoftDeletes;

di dalam class : use SoftDeletes;

protected $dates = ['deleted\_at']; //taro kolom dalam variable $dates

c. cara melihat semua data baik yang udah ke hapus :

$blog = Blog::WithTrashed()->get();

d. cara restore:

$blog = BLog::WithTrashed()->find(2);

//pertama2 liat semua data yang ada trs cari id nmr 2

$blog->restore();

8 UPDATE

1. Langkah awal:

a. Bedakan dan sediakan antara :

1. tempat bwt ngedit

2. tempat coding updatenya

b. Workflow :

1. di routes: sediakan route untuk interface edit @edit

Route::get('/blog','BlogController@index');

Route::get('/blog/{id}','BlogController@show');

Route::get('/blog/{id}/edit','BlogController@edit');

//ini gunanya bwt tampilan edit

Route::post('/blog/{id}','BlogController@update');

//ini isinya coding buat update

2. di controller : function @edit menuju edit.blade.php dengan membawa parameter $id

public function edit($id) //ambil parameter $id dari http

{

$blog = Blog::find($id); //nyocokin $id dengan id pada database

if (!$blog)

abort(404);

return view('blog/edit', ['blog'] => '$blog' );

//jika ada, object dengan id ini akan di oper ke halaman edit

}

3. di edit.blade.php : isi yang pengen diisi trs menuju update

@section

<h1> Tampilan untuk update data </h1>

<form action="/blog/{{blog->id}}" method="post">

//{{blog->id}} blog berasal dari operan controller

<input type="text" name="title" value="{{blog->title}}"> <br>

//value diisi untuk memberi tahu isi sebelumnya pada id tersebut

//isi name='title' akan di simpan dalam request

<textarea name="description" rows="8" cols="80"> {{blog->description}} </textarea> <br>

//isi name='description' akan di simpan dalam request

<input type="submit" name="submit" value="edit">

{{ csrf\_field() }}

//crsf\_field() merupakan token untuk keamanan

<input type="hidden" name="\_method" value="PUT">

//palng penting : ini untuk memberi tahu routes kalo ini akan menuju metode put yaitu Route::put('/blog/{id}','BlogController@update');

bukan ke Route::get('/blog/{id}','BlogController@show');

meskipun uri mereka berdua sama

</form>

@endsection

4. di routes : sediakan routes untuk update @update

//di nmr 1

5. di function update : isi perintah update dan redirect ke terserah mw mu

public function update (Request $request, $id)

//request adalah isi dari yang sudah diisi di edit.blade.php

{

$blog = Blog::find($id);

$blog->title = $request->title;

$blog->description = $request->description;

$blog->save();

return redirect('blog'.$id);

}

9 INSERT DAN DELETE

1. Langkah2 :

a. di route tambahi:

// untuk create

Route::get('/blog/create','BlogController@create');

//untuk menuju halaman form create

Route::get('/blog','BlogController@store');

//untuk code create

// untuk delete

Route:;delete('blog/{id}','BlogController@destroy');

Route::get('/blog','BlogController@index');

Route::get('/blog/{id}','BlogController@show');

Route::get('/blog/{id}/edit','BlogController@edit');

Route::put('/blog/{id}','BlogController@update');

b. di controller:

//fungsi @create

public function create()

{

return view('blog/create');

}

//funsi @store

public function store(Request $request)

{

$blog = new Blog;

$blog->title = $request->title;

$blog->description = $request->description;

$blog->save();

return redirect('blog');

}

//fungsi destroy

public function destroy($id)

{

$blog = Blog::find($id);

$blog->delete();

return redirect('blog');

}

c. di blade :

1. bwt create.blade.php dan isi dengan:

@section

<h1> Create blog </h1>

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

<input type="text" name="title"> <br>

<textarea name="description" rows="8" cols="80"></textarea> <br>

<input type="submit" name="submit" value="edit">

{{ csrf\_field() }}

</form>

@endsection

d. ini bukan metode get jadi kita nggak bisa blog/1/delete untuk itu di single.blade.php , diberi kode dibawah ini pada a hrefnya agar tiap klik redirect ke idnya:

<form action="/blog/{{blog->id}}" method="post">

<input type="submit" name="submit" value="DELETE">

{{ csrf\_field() }}

</form>

2. PENTING : perhatikan routenya jangan sampai tabrakan, perhatikan juga actionnya jangan sampai salah menujunya

10 BASIC VALIDATION

1. Guna: valadasi apakah data yang diisi sesuai kriteria apa nggak

2. Yang diuji adalah requestnya

3. Langkah 1 ngeluarin semua error :contoh :

a. di function store untuk form create:

public function store(Request $request)

{

$this->validate($request,

[

'title' => 'required|min:5'

'description' => 'required|min:5|max:10'

] );

// pisahin tiap validasi dengan |

// 'title dan description' adlah name dari form

blabla.....................

}

b. di create.blade.php:

Letakkan ini pada tempat dimana error mau ditampilkan:

@if (count($errors) > 0)

@foreach($errors->all() as $error)

<li> {{$error}}</li>

@endforeach

@endif

4. Langkah 2 : error tertentu :

a. di create.blade.php di bagian error mw ditampilkan

// di title

@if

@if(@errors->has('title')) //title adalah name session dari form

<p> ada error pada title </p> atau

<p> {{ $errors->first('title') }} </p>

@endif

//di description

@if

@if(@errors->has('description')) //title adalah name session dari form

<p> ada error pada title </p> atau

<p> {{ $errors->first('description') }} </p>

@endif

11 MENGAMBIL NILAI LAMA

1. Guna : misal udah panjang2 ngetik trs ada yang salah, pasti kan yang barusan dihapus hilang,

antiisipasinya pake ini biar nggk hilang

2. Langkah :

1. di bagian property value diberi old('isi\_name'), contoh :

<input type="text" name="title" value=" {{ old('title') }} ">

12 MIGRATION

1. Guna : Sebagai control version system untuk database dan table2 yang kita buat:

a. bisa memberi check point untuk kembali jika ada kesalahan

--> untuk ini buat diawal project

b. bisa share database dengan orang lain

2. Cara membuat :

a. php artisan make:migration nama\_table

--> akan membuat file di app/migration/nama\_table

--> di file akan otomatis membuat 2 metode up() dan down()

--> up() untuk membuat tablenya

--> down() untuk ngedrop tablenya

--> dari 2 metode ini kita bisa ngereset atau rollback data di db

b. di up() diisi :

Schema::create('nama\_tablenya', function (Blueprint $table)

{

$table->increments('id');

$table->string('title', 100); //100 adalah max lenghtnya

$table->string('description');

$table->timestamps();

});

c. di down() diisi:

public function down()

{

Schema::drop('nama\_tablenya');

}

3. cara menjalankan:

php artisan migrate //ini akan otomatis membuat table ke database

php atrtisan migrate:rollback --step=5 //untuk roleback ke step 5 yang kita lakukan

php atrtisan migrate:reset //jika pengen ngilangin table databasenya

13 CSS dan JAVASCRIPT pada LARAVEL

1. Semua file css dan js harus ditaro di folder public/

2. cara memanggil dari file htmlnya :

-->css :

<link rel="stylesheet" href="/css/nama\_file\_cssnya.css">

//folder public/ nggk usah dicantumin

-->js:

<script src="/js/nama\_file\_jsnya.js" ></script>

14 TAMBAHAN

1. Php artisan tinker
2. Dbal laravel