Contents

[Folder Laravel 4](#_Toc158986438)

[Layouting 5](#_Toc158986439)

[Components 5](#_Toc158986440)

[Create Component 5](#_Toc158986441)

[Send Param to Components 5](#_Toc158986442)

[Migration 6](#_Toc158986443)

[Create Migration 6](#_Toc158986444)

[Rollback 6](#_Toc158986445)

[Define Column 6](#_Toc158986446)

[Model, Controller 7](#_Toc158986447)

[Seeder 8](#_Toc158986448)

[Factory 8](#_Toc158986449)

[Eloquent 8](#_Toc158986450)

[Soft Delete with Laravel 9](#_Toc158986451)

[Create migration 9](#_Toc158986452)

[Set Model 10](#_Toc158986453)

[Set Controller: 10](#_Toc158986454)

[Show Deleted item 10](#_Toc158986455)

[Restore deleted data 10](#_Toc158986456)

[Form Define 10](#_Toc158986457)

[Insert Form and Routes 10](#_Toc158986458)

[Edit Form and Routes 11](#_Toc158986459)

[Delete Form and Routes 11](#_Toc158986460)

[Mementukan relationship 11](#_Toc158986461)

[Many to One (belongsTo) 11](#_Toc158986462)

[One to Many (hasMany) 11](#_Toc158986463)

[Many to Many (belongsToMany) 12](#_Toc158986464)

[Nested Relation 12](#_Toc158986465)

[Lazy Loading vs Eager Loading 13](#_Toc158986466)

[Lazy Loading 13](#_Toc158986467)

[Eager Loading 13](#_Toc158986468)

[Session Flash Data 13](#_Toc158986469)

[Validation 14](#_Toc158986470)

[in controller: 14](#_Toc158986471)

[external validation: 14](#_Toc158986472)

[Pagination 15](#_Toc158986473)

[Define and Implementation 15](#_Toc158986474)

[Search and pagination 16](#_Toc158986475)

[Upload Image 17](#_Toc158986476)

[Upload Process 17](#_Toc158986477)

[show photo 18](#_Toc158986478)

[delete photo/file: 18](#_Toc158986479)

[Login & Logout 18](#_Toc158986480)

[Set middleware 18](#_Toc158986481)

[Menggunakan middleware 18](#_Toc158986482)

[Login process 19](#_Toc158986483)

[Show Data User Login 19](#_Toc158986484)

[Logout process 20](#_Toc158986485)

[Slug 20](#_Toc158986486)

[Eloquent-sluggable 20](#_Toc158986487)

[Install package 20](#_Toc158986488)

[Map 21](#_Toc158986489)

[Middleware & Hak Akses 22](#_Toc158986490)

[Breeze: Authentication 22](#_Toc158986491)

[Install Laravel Breeze 22](#_Toc158986492)

[Multi DB Connection 23](#_Toc158986493)

[Setup 23](#_Toc158986494)

[Get relation from multi DB 24](#_Toc158986495)

[Create UUID 24](#_Toc158986496)

[Setting Primary key not increment 24](#_Toc158986497)

[Setting Primary Key = String 24](#_Toc158986498)

[Create UUID while insert 24](#_Toc158986499)

[Laravel Logging 25](#_Toc158986500)

[Hide pesan error 25](#_Toc158986501)

[Tipe Logging / channel 25](#_Toc158986502)

[Setting Logging 25](#_Toc158986503)

[Jenis/channel daily 25](#_Toc158986504)

[Thrid-party viewer 25](#_Toc158986505)

[Install: 25](#_Toc158986506)

[Menambahkan log sendiri 26](#_Toc158986507)

[HTTP Client API 26](#_Toc158986508)

[Implement GET 26](#_Toc158986509)

[Implement POST 26](#_Toc158986510)

[Implement PUT 27](#_Toc158986511)

[Implement DELETE 27](#_Toc158986512)

[Implenet SignIn 27](#_Toc158986513)

[Create REST API 28](#_Toc158986514)

[GET 28](#_Toc158986515)

[POST 29](#_Toc158986516)

[PATCH (update) 29](#_Toc158986517)

[Delete 30](#_Toc158986518)

[API Resource 31](#_Toc158986519)

[Tambah Custom data 31](#_Toc158986520)

[API show detail 32](#_Toc158986521)

[Eager Loading di Backend 33](#_Toc158986522)

[Multi Relation 33](#_Toc158986523)

[API Authentication Login/Logout 35](#_Toc158986524)

[Install Sanctum 35](#_Toc158986525)

[Create Login Endpoint 35](#_Toc158986526)

[Revoke Token/ Logout 36](#_Toc158986527)

[Config expiration token 37](#_Toc158986528)

[API Upload File 38](#_Toc158986529)

# Folder Laravel

route:

ada di

/routes/web.php

view:

ada di

/resources/views

controller:

App\Http\Controllers\

show list of route:

$php artisan route:list

List instaled package:

/composer.json

--------------------------------------------------------------------------------------------------

# Layouting

menggunakan file mainlayout yg ada di folder layout

@extends('layout.mainlayout');

mengirimkan variable dan value nya ke layout/ yg ada di @extends

@section('<nama var>', '<value>');

memanggil/menggunakan var yg di kirim section

@yield('<nama var>')

membungkus isi html untuk di kirim ke layout dan lebih dari 1 baris

@section('<nama var>')

@endsection

## Components

Bagian terkecil dari tampilan, mirip dengan react. Misal kita membuat component button, maka button ini bisa di pakai di tempat lain(reuseable)

### Create Component

Php arisan make:component <nama>

Lokasi : \app\View\Components\Alert.php

Dan di \resources\views\components\alert.blade.php

Implementasi nya tinggal sisipkan di html

<x-alert />

### Send Param to Components

[Home.blade.php](http://Home.blade.php)

<x-alert messages='ini adalah halaman home'/>

\*perhatikan tidak boleh ada spasi antara nama param dengan ‘=’

Alert.php

 public $messages;

    public function \_\_construct($messages)

    {

        $this->messages = $messages;

    }

Alert.blade.php

<div class="alert alert-primary" role="alert">

    {{$messages}}

</div>

Bila lebih dari 1 param, tinggal buat lagi nama Param nya

--------------------------------------------------------------------------------------------------

# Migration

## Create Migration

migration file location

/database/migrations/

run migrations

$ php artisan migrate

generate migration file

$ php artisan make:migration create\_students\_table

generate migration to update

$ php artisan make:migration add\_gender\_column\_to\_students\_table

\*ps : make sure in migration file name, use '<table\_name>\_table'

## Rollback

mengembalikan state migrate -1 step

$ php artisan migrate:rollback

mengembalikan state migrate -5 step

$ php artisan migrate:rollback --step=5

## Define Column

membuat kolom gender, tipe string (10), not null/harus diisi, setelah kolom name:

$table->string('gender', 10)->required()->after('name');

boleh null

notnull(false)

delete column 'gender':

$table->dropColumn('gender');

mengupdate column gender sesuai dengan yg di definisikan:

$table->string('gender', 1)->nullable()->after('name')->change();

membuat relasi menggunakan kolom class\_id, reference ke 'id' yg ada di table 'class'. Bila ada data di table 'class' yg di delete, maka di larang / restrict (karena telah di gunakan di table 'students'):

$table->foreign('class\_id')->references('id')->on('class')->onDelete('restrict');

menghapus relasi pada kolom class\_id:

$table->dropForeign(['class\_id']);

--------------------------------------------------------------------------------------------------

# Model, Controller

create model

$php artisan make:model Student

lokasi model app/Models

define nama table, jika nama Model tidak sama dengan nama table:

protected $table = 'students';

define PK, jika nama primary key nya tidak sama dengan id

protected $primaryKey = 'id';

define that in table don't have column 'created\_at' & 'updated\_at':

public $timestamps = false;

create controller:

$php artisan make:controller StudentController

lokasi controller app/Http/Controllers

set route to studentController with function name index:

Route::get('/students', [StudentController::class, 'index']);

--------------------------------------------------------------------------------------------------

# Seeder

create seeder file:

$ php artisan make:seeder <name\_seeder>

lokasi file seeder:

\database\seeders

run file seeder:

$ php artisan db:seed --class=<name\_seeder>

run file seeder yg ada di \database\seeders\DatabaseSeeder.php:

$ php artisan db:seed

## Factory

create factory file:

$ php artisan make:factory StudentFactory

call StudentFactory in StudentSeeder :

Student::factory()->count(37)->create();

# Eloquent

eloquent untuk select \* dari table students via model Student:

$studentList = Student::all(); // Lazy Loading

select id,name from class;

$classList = ClassRoom::select('id', 'name')->get();

eloquent to Insert

// insert using eloquent

        Student::create([

            'name' => 'Ei',

            'gender' => 'P',

            'nis' => '0102459',

            'class\_id' => 1

        ]);

\*before insert, set on Model mass asignment by add this code

protected $fillable = ['name', 'nis', 'gender', 'class\_id','image'];

you can use $request to get data from Form:

public function store(Request $request) {

        $student = Student::create([

                    'name' => $request->name,

                    'gender' => $request->gender,

                    'nis' => $request->nis,

                    'class\_id' => $request->class\_id

                ]);

        }

Untuk mendaftarkan variabel lain dalam $request

$request['author'] = Auth::user()->id;

Atau

$request['user\_id'] = auth()->user()->id;

Update:

Student::findOrFail(44)->update([

        'name' => $request->name,

        'class\_id' => $request->class\_id,

        etc...

    ]);

atau untuk update semua isi $request

$post = Post::findOrFail($id);

$post->update($request->all());

Atau untuk update 1 kolom saja

$comment = Comment::findOrFail($id);

$comment->update($request->only('comments\_content'));

Delete:

Student::find(44)->delete();

## Soft Delete with Laravel

### Create migration

- create migration to table, after you run this migration, Laravel will create column deleted\_at:

public function up()

    {

        Schema::table('students', function (Blueprint $table) {

            $table->softDeletes();

        });

    }

public function down()

    {

        Schema::table('students', function (Blueprint $table) {

            $table->dropSoftDeletes();

        });

    }

### Set Model

- add soft delete in Model:

use SoftDeletes;

### Set Controller:

just use regular delete function, then it already use soft delete

$delete = Student::findOrFail($id)->delete();

### Show Deleted item

to show data that already deleted using soft delete

$deleted = Student::onlyTrashed()->get();

## Restore deleted data

- routes:

Route::get('/student/{id}/restore', [StudentController::class, 'restore']);

-controller:

$restore = Student::withTrashed()->where('id', $id)->restore();

\*Student::withTrashed() show all student and all deleted data

select \* from class where id != $student->class\_id

$class = ClassRoom::where('id', '!=', $student->class\_id)->get();

--------------------------------------------------------------------------------------------------

# Form Define

## Insert Form and Routes

....blade.php

<form action='student' method='post'>

        @csrf

        ...

        <button class="btn btn-success" type="submit">Save</button>

</form>

web.php

Route::post('/student', [StudentController::class, 'store']);

Bila setelah di submit tetap di halaman yg sama:

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

        @csrf

web.php

Route::post('/cek-ongkir', [OngkirController::class, 'cekOngkir']);

## Edit Form and Routes

....blade.php

<form action='/student/{{$student->id}}' method='post'>

    @csrf

    @method('PUT')

    ...

    <button class="btn btn-success" type="submit">Update</button>

</form>

web.php

Route::put('/student/{id}', [StudentController::class, 'update']);

## Delete Form and Routes

<form action="/student-destroy/{{$student->id}}"  method='post'>

    @csrf

    @method('delete')

    <button type='submit' class="btn btn-danger">Delete</button>

</form>

Route::delete('/student-destroy/{id}', [StudentController::class, 'destroy']);

--------------------------------------------------------------------------------------------------

# Mementukan relationship

## Many to One (belongsTo)

contoh table student dan class

|------------------| |----------|

|students | |classs |

|------------------| |----------|

|id|name | class\_id| |id|name |

|1 | adi | 2 | |1 | 1A |

|2 | budi| 1 | |2 | 2B |

|3 | tuti| 2 |

student -> class = many to one

di table student,

apakah 1 class hanya memiliki 1 student (class\_id unique)? tidak, berarti 'many'

apakah 1 student hanya memiliki 1 class? ia, berarti 'one'

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

## One to Many (hasMany)

class -> student = one to many

di table class,

apakah 1 student hanya memiliki 1 class? ia, berarti 'one'

apakah 1 class hanya memiliki 1 student? tidak, berarti 'many'

return $this->hasMany(Student::class, 'class\_id', 'id');

## Many to Many (belongsToMany)

|-------------| |------------------------|

|ekskul | |student\_ekskul (pivot) |

|-------------| |------------------------|

|id |name | |student\_id |eksekul\_id |

|1 | bola | |1 | 1 |

|2 | basket | |1 | 2 |

|3 | voli | |2 | 3 |

student->ekskul = many to many

apakah 1 ekskul hanya memiliki 1 student (class\_id unique)? tidak, berarti 'many'

apakah 1 student hanya memiliki 1 ekskul? tidak, berarti 'many'

In class Student:

return $this->belongsToMany(Ekskul::class, 'student\_ekskul', 'student\_id', 'ekskul\_id');

In Class Ekskul:

return $this->belongsToMany(Student::class, 'student\_ekskul', 'ekskul\_id', 'student\_id');

## Nested Relation

|------------------| +---------------------+ +------------+

|students | |classs | |teacher |

|------------------| +---------------------+ +------------+

|id|name | class\_id| |id|name |teacher\_id| |id|name |

|1 | adi | 2 | |1 | 1A | 2 | |1 | Pak Eko |

|2 | budi| 1 | |2 | 2B | 1 | |2 | Bu Mega |

|3 | tuti| 2 |

students want to see teacher info, first make relation between class and teacher:

ClassRoom Model:

public function homeRoomTeacher()

    {

        return $this->belongsTo(Teacher::class, 'teacher\_id', 'id');

    }

second, because student already joined/have relation with class, we can directly access function homeRoomTeacher in controller by adding:

StudentController:

$studentList = Student::with(['class.homeRoomTeacher', 'ekskul'])->get(); // Eager loading (recomended)

student.blade:

<td>{{$data->class->homeRoomTeacher->name}}</td>

For One to One relation:

Teacher Model:

return $this->hasOne(ClassRoom::class);

--------------------------------------------------------------------------------------------------

# Lazy Loading vs Eager Loading

## Lazy Loading

$class = ClassRoom:all();

// select \* from table class

// select \* from student where class= 1A

// select \* from student where class= 1B

// select \* from student where class= 1C

// select \* from student where class= 2A, sampai sbnyk data class nya

## Eager Loading

$class = ClassRoom:with('students')->get();

// select \* from table class

// select \* from student where class IN (1A,1B,1C,etc)

--------------------------------------------------------------------------------------------------

# Session Flash Data

constroller:

use Illuminate\Support\Facades\Session;

if($student){

            Session::flash('status','success');

            Session::flash('message','Sukses Insert Data');

        } else {

            Session::flash('status','failed');

            Session::flash('message','Gagal Insert Data');

        }

view:

@if (Session::has('status'))

        @if (Session::get('status') == 'success')

            <div class="alert alert-success" role="alert">

                {{Session::get('message')}}

            </div>

        @else

            <div class="alert alert-danger" role="alert">

                {{Session::get('message')}}

            </div>

        @endif

@endif

--------------------------------------------------------------------------------------------------

# Validation

## in controller:

$validated = $request->validate([

            'nis' => 'unique:students|max:10|required',

            'name' => 'max:50|required',

            'gender' => 'required',

            'class\_id' => 'required'

        ]);

https://laravel.com/docs/9.x/validation#available-validation-rules

## external validation:

php artisan make:request StudentCreateRequest

lokasi

app/Http/Requests/StudentCreateRequest.php

\*ubah return false menjadi return true pada function authorize

isi rules/validation nya

public function rules()

    {

        return [

            'nis' => 'unique:students|max:10|required',

            'name' => 'max:50|required',

            'gender' => 'required',

            'class\_id' => 'required'

        ];

    }

lalu pada file controller ubah:

public function store(Request $request)

menjadi

public function store(StudentCreateRequest $request)

dan tambahkan

use App\Http\Requests\StudentCreateRequest;

ubah nama field / kolom dari default menjadi custom:

public function attributes()

    {

        return [

            'class\_id' => 'class',

        ];

    }

ubah kalimat/ pesan error:

public function messages()

    {

        return [

            'nis.required' => 'NIS wajib diisi',

            'nis.max' => 'NIS maksimal :max angka',

            'name.required' => 'Nama wajib diisi',

        ];

    }

show error message in html:

@if ($errors->any())

            <div class="alert alert-danger">

                <ul>

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

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

                    @endforeach

                </ul>

            </div>

@endif

--------------------------------------------------------------------------------------------------

# Pagination

## Define and Implementation

AppServiceProvider.php:

use Illuminate\Pagination\Paginator;

public function boot()

    {

        Paginator::useBootstrapFive();

    }

controller:

$studentList = Student::paginate(15);

view:

@foreach ($studentList as $data)

 <tr>

   <td>{{ $studentList->firstItem() + $loop->index }}</td>

 </tr>

@endforeach

<div class='my-3'>

  {{$studentList->links()}}

</div>

## Search and pagination

controller:

public function index(Request $request)

{

$keyword = $request->pencarian;

        $studentList = Student::with('class')

                        ->where('name', 'LIKE', '%'.$keyword.'%')

                        ->orWhere('gender', $keyword)

                        ->orWhere('nis', 'LIKE', '%'.$keyword.'%')

                        ->orWhereHas('class', function($query) use($keyword){

                            $query->where('name', 'LIKE', '%'.$keyword.'%');

                        })

                        ->paginate(15);

}

\*orWhereHas('<relation\_name>') function untuk mencari data yg ada dalam relasi

view:

<form action="" method="get">

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

<input type="text" class="form-control" name="pencarian" id="floatingInputGroup1" placeholder="Keyword">

<button class="input-group-text btn btn-primary">Search</button>

</div>

</form>

{{$studentList->withQueryString()->links()}}

\*untuk tetap mempertahankan keyword search saat pindah page

--------------------------------------------------------------------------------------------------

# Upload Image

## Upload Process

Lokasi folder storage public

config/filesystems.php

set .env:

FILESYSTEM\_DISK=public

karena menggunakan public maka file image akan disimpan di

storage\_path('app/public')

atau

app/storage/app/public

menghubungkan folder storage, ke publik agar bisa di akses di web

$ php artisan storage:link

controller:

$fileName = '';

if($request->file('photo')){

$extension = $request->file('photo')->getClientOriginalExtension();

    $statement = DB::select("show table status like 'students'");

    $newID = $statement[0]->Auto\_increment;

    $fileName = $request->name.'-'.$newID.'-'.now()->timestamp.'.'.$extension;

    $path = $request->file('photo')->storeAs('studentsPhoto', $fileName);

}

view form:

<form action='student' method='post' enctype="multipart/form-data">

<div class="input-group">

  <input type="file" class="form-control" id="photo" name="photo">

</div>

## show photo

<div class='my-3 d-flex justify-content-center'>

@if($student->image !='')

  <img src="{{asset('storage/studentsPhoto/'.$student->image)}}" width="200px">

  @else

  <img src="{{asset('images/anonymus.jpg')}}" width="200px">

@endif

</div>

## delete photo/file:

use Illuminate\Support\Facades\Storage;

Storage::delete('studentsPhoto/'.$student->image);

--------------------------------------------------------------------------------------------------

# Login & Logout

add role column di table users

## Set middleware

route middleware:

app/Http/Kernel.php

middleware untuk authentication:

App\Http\Middleware\Authenticate.php

## Menggunakan middleware

web.php

Route::get('/students', [StudentController::class, 'index'])->middleware('auth');

\*middleware('auth') artinya memanggil route middleware bernama 'auth' yg terdaftar di Kernel.php

dari Kernel.php, buka file middleware nya di

'auth' => \App\Http\Middleware\Authenticate::class

Authenticate.php

akan meredirect ke route bernama 'login' jika user blm login

web.php

tambahkan,

Route::get('/login', [AuthController::class, 'login'])->name('login');

\*name('login'), merupakan *named route* agar bisa di kenal di file/tempat lain, karena Authenticate.php butuh route bernama 'login', kita beri nama 'login' juga

Grouping middleware

Route::middleware(['auth:sanctum'])->group(function () {

    Route::get('logout',[AuthenticationController::class, 'logout']);

    Route::get('me',[AuthenticationController::class, 'me']);

});

## Login process

web.php:

Route::post('/login', [AuthController::class, 'authenticating'])->middleware('guest');

Create AuthController.php

public function authenticating(Request $request)

    {

        $credentials = $request->validate([

            'email' => ['required', 'email'],

            'password' => ['required'],

        ]);

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

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

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

        }

        Session::flash('status','failed');

        Session::flash('message','Login salah');

        return redirect('/login');

    }

## Show Data User Login

Show Data user yg sedang login

{{Auth::user()}}

Data User dengan relation name 'role'

{{Auth::user()->role}}

Id user cek di controller

$user = Auth::user();

dd($user->id);

middleware('guest');

\*middleware untuk yg sudah login, maka akan di redirect

\*di RouteServiceProvider.php, ada const HOME, yg bisa di ganti untuk setting mau redirect kemana

## Logout process

public function logout(Request $request)

    {

        Auth::logout();

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

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

        return redirect('/login');

    }

--------------------------------------------------------------------------------------------------

# Slug

Mengganti / replace string dengan char lain/ generates a URL friendly "slug" from the given string

use Illuminate\Support\Str;

Str::slug($request->name, '\_')

## Eloquent-sluggable

Package yg memungkinkan kita untuk auto increment slug ketika insert data, sehingga kolom slug selalu unik

<https://github.com/cviebrock/eloquent-sluggable>

### Install package

composer require cviebrock/eloquent-sluggable

Model

use Cviebrock\EloquentSluggable\Sluggable;

class Student extends Model

{

    use Sluggable;

 protected $fillable = ['name', 'slug', 'nis', 'gender', 'class\_id','image'];

public function sluggable(): array

    {

        return [

            'slug' => [

                'source' => 'name'

            ]

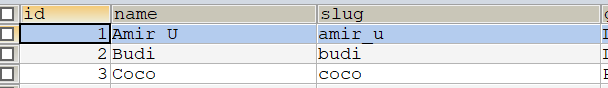
        ];

    }

Perhatikan untuk source, kita menggunakan name, karena bahan kolom yg mau di slug adalah ‘name’

Sql Table

Tambah kolom slug



Controller

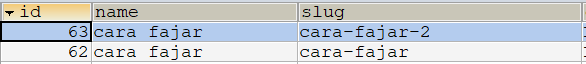
Tidak perlu tambah apa2, hapus pembuatan slug manual jika ada

$student = Student::create([

            'name' => $request->name,

            //'slug' => Str::slug($request->name, '\_'),

Lalu ketika kita insert data baru otomatis kolom slug akan diinsert dengan increment di belakang nya, jika data slug yg sama sudah ada.



# Map

Melakukan proses dalam collection

public function massUpdate()

    {

        $students = Student::all();

        collect($students)->map(function ($item, $key) {

            $item->slug = Str::slug($item->name, '\_');

            $item->save();

        });

    }

Pada contoh diatas, kita meng-get semua data student, lalu di tamping di dalam collection dan map

Setiap kolom ‘slug’ kita replace dengan char ’\_’, lalu langsung di update dengan save();

# Middleware & Hak Akses

Middleware, untuk memberika hak akses

$ php artisan make:middleware <nama\_Middleware>

lokasi:

app/Http/Middleware

daftarkan middleware yg di buat di Kernel.php dengan format:

'<name>' => \<namespace middleware>\<nama\_class>::class

Contoh:

'must-admin' => \App\Http\Middleware\MustAdmin::class,

implement di route web.php:

Route::get('/student-delete/{id}', [StudentController::class, 'delete'])->middleware(['auth','must-admin']);

# Breeze: Authentication

Min node version 16

## Install Laravel Breeze

composer require laravel/breeze –dev

php artisan breeze:install

php artisan migrate

npm install

npm run dev

lalu buka url <http://localhost:8000/login> untuk halaman login, dan <http://localhost:8000/register> untuk register

FIle-file yg dibuat oleh breeze:

* \resources\views\auth
* \resources\views\components
* \resources\views\layouts
* \routes\auth.php

# Multi DB Connection

## Setup

.env

Buat duplicate DB\_DATABASE, DB\_USERNAME,DB\_PASSWORD dengan menambahkan ‘2’ dibelakang

DB\_DATABASE=laravel\_belajar\_20231212

DB\_USERNAME=root

DB\_PASSWORD=

DB\_DATABASE2=mycode

DB\_USERNAME2=root

DB\_PASSWORD2=

\config\database.php

Cari array connection, buat duplicate dari mysql dan tambahkan angka2, lalu gunakan env DB\_DATABASE2, DB\_USERNAME2, DB\_PASSWORD2

'mysql2' => [

            'driver' => 'mysql',

            'url' => env('DATABASE\_URL'),

            'host' => env('DB\_HOST', '127.0.0.1'),

            'port' => env('DB\_PORT', '3306'),

            'database' => env('DB\_DATABASE2', 'forge'),

            'username' => env('DB\_USERNAME2', 'forge'),

            'password' => env('DB\_PASSWORD2', ''),

            'unix\_socket' => env('DB\_SOCKET', ''),

            'charset' => 'utf8mb4',

            'collation' => 'utf8mb4\_unicode\_ci',

            'prefix' => '',

            'prefix\_indexes' => true,

            'strict' => true,

            'engine' => null,

            'options' => extension\_loaded('pdo\_mysql') ? array\_filter([

                PDO::MYSQL\_ATTR\_SSL\_CA => env('MYSQL\_ATTR\_SSL\_CA'),

            ]) : [],

        ],

Model

Define connection ‘mysql2’, atau bisa juga define $table dengan format <nama\_db>.<nama\_table>

class Domicile extends Model

{

    use HasFactory;

    protected $connection = 'mysql2';

    // atau

    // protected $table = 'mycode.domiciles';

}

## Get relation from multi DB

Cara nya sama dengan relation biasa, karena dari sisi model sudah di connect/ setup ke db lain

# Create UUID

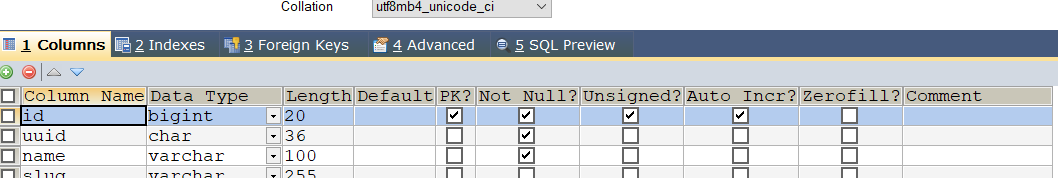
File migration:

Schema::table('students', function (Blueprint $table) {

            $table->uuid('uuid')->after('id');

        });

Hasil:



## Setting Primary key not increment

Model

public function getIncrementing()

    {

        return false;

    }

## Setting Primary Key = String

Model

public function getKeyType()

    {

        return 'string';

    }

## Create UUID while insert

use Illuminate\Support\Str;

$student = Student::create([

'uuid' => Str::uuid(),

        ]);

# Laravel Logging

## Hide pesan error

.env

APP\_DEBUG=false

## Tipe Logging / channel

<https://laravel.com/docs/9.x/logging#available-channel-drivers>

## Setting Logging

\config\logging.php

Untuk lihat tipe logging ada di .env

LOG\_CHANNEL=stack

By default menggunakan stack.

Stack adalah bila terjadi error, maka akan dikumpulkan di dalam satu file. Lokasi ada di

storage/logs/laravel.log

## Jenis/channel daily

Pada Daily, laravel akan membuat log berdasarkan hari

LOG\_CHANNEL=daily

Lokasi log akan dibuat di storage\logs\laravel-2024-02-15.log

## Thrid-party viewer

<https://github.com/opcodesio/log-viewer>

### Install:

composer require opcodesio/log-viewer

Untuk lihat: <http://localhost:8000/log-viewer>

### Menambahkan log sendiri

Dari web.php

Route::get('/create-log', function () {

    Log::info('ini adalah log');

Log::alert('ini adalah alert');

    Log::debug('ini debug');

    Log::warning('Warning');

});

# HTTP Client API

Install package

$ composer require guzzlehttp/guzzle

## Implement GET

use Illuminate\Support\Facades\Http;

public function index(){

        // $response = Http::get('https://api.rajaongkir.com/starter/province');

        $response = Http::withHeaders([

            'key' => 'd2066b08becc7e1e8a27e1764cf85de0'

        ])->get('https://api.rajaongkir.com/starter/province');

        // $result = $response->json()['rajaongkir']['results'];

        $result = $response['rajaongkir']['results'];

        dd($result);

    }

## Implement POST

public function cekOngkir(Request $request)

    {

$response\_post =  Http::withHeaders([

            'key' => 'd2066b08becc7e1e8a27e1764cf85de0'

        ])->post('https://api.rajaongkir.com/starter/cost', [

            'origin' => $request->origin,

            'destination' => $request->destination,

            'weight' => $request->weight,

            'courier' => $request->courier,

        ]);

$ongkir = $response\_post['rajaongkir'];

## Implement PUT

public function testUpdate()

    {

        $response = Http::withToken('eyJhbGciOiJIUzI1NiIsI….’)

                    ->put('http://localhost:8080/auth/edituser/14',[

                        "name" => "bobi2",

                        "alamat" => "jkt",

                        "umur" => "18",

                        "nohp" => "777"

                    ]);

        dd($response['metadata']);

    }

withToken() bias digunakan untuk jenis token Bearer

## Implement DELETE

public function testDelete()

    {

        $response = Http::withToken('eyJhbGciOiJI….')

                    ->delete('http://localhost:8080/auth/deleteuser/13');

        dd($response->json());

    }

## Implenet SignIn

public function testSignIn()

    {

        $response = Http::post('http://localhost:8080/auth/signin',[

            'username' => 'admin',

            'password' => '12345'

        ]);

        dd($response->json());

    }

# Create REST API

Lokasi routes:

\routes\api.php

## GET

Route:

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

Controller :

public function index()

    {

        $data = Post::all();

        return response()->json($data);

    }

Akses URL:

<http://localhost:8081/belajar-laravel/public/api/getPosts/>

Menggunakan key data:

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

sehingga hasilnya:

{

    "data": [

        {

            "id": 1,

            "title": "Welcome",

            "news\_content": "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent

## POST

Route:

Route::post('/posts', [PostController::class, 'store']);

Controller:

public function store(Request $request)

    {

        $validated = $request->validate([

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

            'news\_content' => 'required'

        ]);

        $request['author'] = Auth::user()->id;

        $post = Post::create($request->all());

        return new PostDetailResource($post->loadMissing('writer:id,name'));

    }

Untuk

return new PostDetailResource($post->loadMissing('writer:id,name'));

bisa di lihat di bagian API Resource

## PATCH (update)

Route:

Route::patch('/posts/{id}', [PostController::class, 'update'])->middleware('pemilik-postingan');

Middleware:

public function handle(Request $request, Closure $next)

    {

        $currentUser = Auth::user();

        $post = Post::findOrFail($request->id);

        if($post->author != $currentUser->id)

        {

            return response()->json(['message' => 'Anda tidak punya akses untuk Edit Post ini'],403);

        }

        return $next($request);

    }

Controller:

public function update(Request $request, $id)

    {

        $validated = $request->validate([

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

            'news\_content' => 'required'

        ]);

        $post = Post::findOrFail($id);

        $post->update($request->all());

        return new PostDetailResource($post->loadMissing('writer:id,name'));

    }

## Delete

Route:

Route::delete('/posts/{id}', [PostController::class, 'destroy'])->middleware('pemilik-postingan');

Controller:

public function destroy($id)

    {

        $post = Post::findOrFail($id);

        $post->delete();

        return new PostDetailResource($post->loadMissing('writer:id,name'));

    }

# API Resource

Agar bisa custom return api kita

Create Resource:

php artisan make:resource PostResource

Lokasi:

\app\Http\Resources\PostResource.php

Implement:

use App\Http\Resources\PostResource;

public function index()

    {

        $data = Post::all();

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

        return PostResource::collection($data);

    }

Maka hasilnya akan sama (berupa array) jika kita menggunakan

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

## Tambah Custom data

Pada file PostResource.php

public function toArray($request)

    {

        // return parent::toArray($request);

        return [

            'id' => $this->id,

            'title' => $this->title,

        ];

    }

Hasilnya:

{

    "data": [

        {

            "id": 1,

            "title": "Welcome"

        },

        {

            "id": 2,

            "title": "Pengumuman"

        }

    ]

}

## API show detail

Route:

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

Model:

public function writer(): BelongsTo

    {

        return $this->belongsTo(User::class, 'author', 'id');

    }

Controller:

public function show($id)

    {

       $post = Post::with('writer:id,name')->findOrFail($id);

       return new PostDetailResource($post);

    }

new disini untuk data single, sedangkan ::collection untuk return array/lebih dari satu

penulisan relation with('writer:id,name') artinya, select id dan name dari relasi writer, untuk id(PK) wajib diisi

Post:

Jika ketika post ingin di tampilkan return nama author nya

return new PostDetailResource($post->loadMissing('writer:id,name'));

Resource:

public function toArray($request)

    {

        return [

            'id' => $this->id,

            'title' => $this->title,

            'news\_content' => $this->news\_content,

            'created\_at' => date\_format($this->created\_at, "Y-m-d H:i:s"),

            'author' => $this->author,

            'writer' => $this->writer

        ];

    }

## Eager Loading di Backend

Controller:

public function show2($id)

    {

        $post = Post::findOrFail($id);

        return new PostDetailResource($post);

    }

Resource:

public function toArray($request)

    {

        return [

            'id' => $this->id,

            'title' => $this->title,

            'news\_content' => $this->news\_content,

            'created\_at' => date\_format($this->created\_at, "Y-m-d H:i:s"),

            'author' => $this->author,

            'writer' => $this->whenLoaded('writer')

        ];

    }

Perhatikan, $this->whenLoaded('writer') artinya hanya di munculkan jika kita menggunakan relation / with writter

# Multi Relation

Pada api get Post kita ingin menampilkan isi komentar, beserta nama user yg menulis komen

Model Post.php:

public function commentsUser()

    {

        return $this->hasMany(Comment::class, 'post\_id', 'id');

    }

Model Comment.php:

public function komentator()

    {

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

    }

Controller:

$data = Post::with(['writer:id,name', 'commentsUser:id,post\_id,comments\_content,user\_id'])->get();

        return PostDetailResource::collection($data);

\*wajib menggunakan user\_id sebagai bahan memnculkan nama user yg komen

Resource:

public function toArray($request)

    {

        return [

            'id' => $this->id,

            'title' => $this->title,

            'news\_content' => $this->news\_content,

            'created\_at' => date\_format($this->created\_at, "Y-m-d H:i:s"),

            'author' => $this->author,

            'writer' => $this->whenLoaded('writer'),

            'comments' => $this->whenLoaded('commentsUser', function(){

                return collect($this->commentsUser)->each(function ($c){

                    $c->komentator;

                    return $c;

                });

            }),

            'total\_comment' => $this->whenLoaded('commentsUser', function(){

                return $this->commentsUser->count();

            })

        ];

    }

Penjelasan

* Pada object 'comments' kita ingin memunculkan juga nama user yg menulis
* return collect($this->commentsUser)->each artinya setiap isi dari relasi commentsUser akan di proses
* $c->komentator; memangil model yg ada di Comment.php (sudah punya relasi dengan User)

Cara 2

Buat relasi di eloquent

$data = Post::with(['writer:id,name', 'commentsUser:id,post\_id,comments\_content,user\_id', 'commentsUser.komentator'])->get();

# API Authentication Login/Logout

## Install Sanctum

If Laravel > version 9, already installed

Route to implement authentication check:

Route::get('/getPosts', [PostController::class, 'index'])->middleware(['auth:sanctum']);

In \app\Models\User.php check if already have this line

use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable

{

    use HasApiTokens, HasFactory, Notifiable;

## Create Login Endpoint

Route to login/generate token(create manual)

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

AuthenticationController.php

use Illuminate\Support\Facades\Hash;

use Illuminate\Validation\ValidationException;

class AuthenticationController extends Controller

{

    public function login(Request $request)

    {

        $request->validate([

            'email' => 'required|email',

            'password' => 'required',

        ]);

        $user = User::where('email', $request->email)->first();

        if (! $user || ! Hash::check($request->password, $user->password)) {

            throw ValidationException::withMessages([

                'email' => ['The provided credentials are incorrect.'],

            ]);

        }

        return $user->createToken('user login')->plainTextToken;

    }

}

Body json:

{

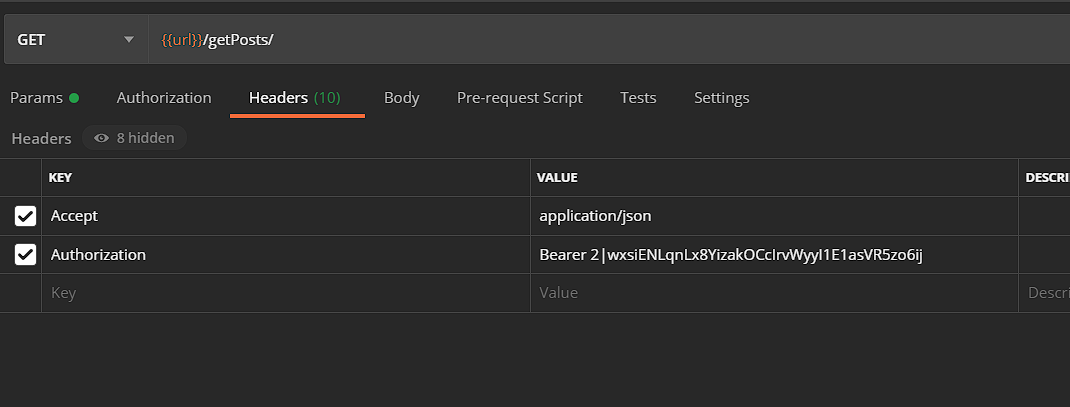
    "email": "admin@mail.com",

    "password" : "12345"

}

In Database, open table personal\_access\_tokens

Tinggal taruh return token nya ke dalam Header endpoint lain



Get User data login:

return response()->json(Auth::user());

\*selama menggunakan token dalam mengaksesnya

## Revoke Token/ Logout

Route:

Route::get('logout',[AuthenticationController::class, 'logout'])->middleware(['auth:sanctum']);

Controller:

public function logout(Request $request)

    {

        // Revoke all tokens...

        // $user->tokens()->delete();

        // Revoke the token that was used to authenticate the current request...

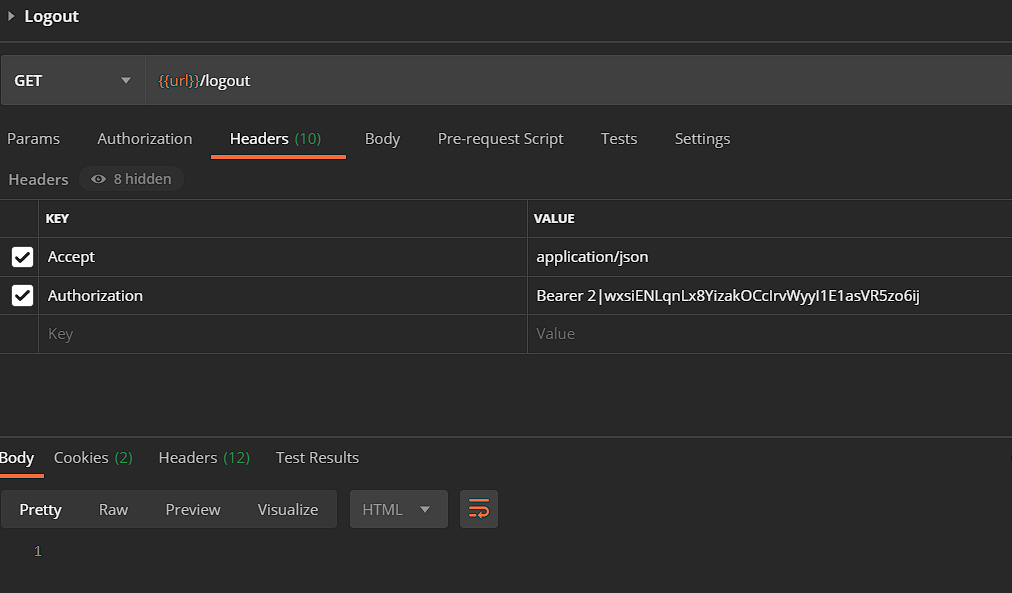
        $request->user()->currentAccessToken()->delete();

        // Revoke a specific token...

        // $user->tokens()->where('id', $tokenId)->delete();

    }

Postman:



Di database, table personal\_access\_tokens akan hilang dari record

## Config expiration token

Ada di \config\sanctum.php

/\*

    |--------------------------------------------------------------------------

    | Expiration Minutes

    |--------------------------------------------------------------------------

    |

    | This value controls the number of minutes until an issued token will be

    | considered expired. If this value is null, personal access tokens do

    | not expire. This won't tweak the lifetime of first-party sessions.

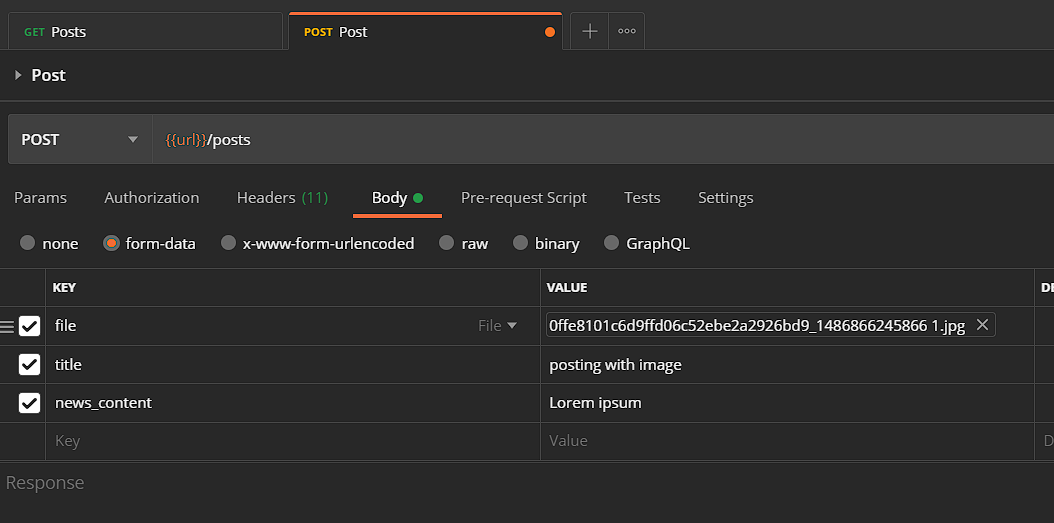
    |

    \*/

    'expiration' => null,

# API Upload File

Postman, gunakan body form-data, dan key untuk file gunakan type ‘File’ (hover pada kolom key):



Controller:

use Illuminate\Support\Facades\Storage;

if($request->file)

        {

            //upload file

            $fileName = $this->generateRandomString();

            $extension = $request->file->extension();

            $path = Storage::putFileAs('image\_post\_folder', $request->file, $fileName.'.'.$extension);

            $request['image'] = $fileName.'.'.$extension;

        }

generateRandomString(); memanggil function untuk generate random string

Storage::putFileAs('image\_post\_folder', $request->file, $fileName.'.'.$extension);

Untuk upload image ke dalam folder /app/storage/app/public/image\_post\_folder

Param 1 = nama folder tempat file di upload

Param 2 = file yg mau di upload

Param 3 = nama file

$request['image'] = $fileName.'.'.$extension;

Memasukan nama file ke dalam array $request, agar bisa di insert dengan Eloquent :

 Post::create($request->all());