

VERSION 1.1
NOVEMBER 23, 2024



PEMGORGAMAN WEBSITE

Pengenalan LARAVEL

DISUSUN OLEH:
OGYA ADYATMA PUTRA
GERALDI NATHAN TOMMY SAPUTRA

DIAUDIT OLEH:
AMINUDIN, S.KOM., M.CS

LAB. INFORMATIKA
UNIVERSITAS MUHAMMADIYAH MALANG

PEMGORGAMAN WEBSITE

PERSIAPAN MATERI

<https://getcomposer.org/>

<https://laravel.com/>

<https://laravel.com/docs/11.x>

TUJUAN

Memahami bagaimana konsep Laravel bekerja, dan melakukan implementasi laravel kedalam tugas praktikum.

TARGET MODUL

1. Praktikan mampu memahami konsep Laravel (MVC)
2. Praktikan mampu melakukan instalasi Laravel
3. Praktikan mampu membuat CRUD (Create Read Update Delete) menggunakan Laravel

PERSIAPAN SOFTWARE/APLIKASI

Hardware

- Laptop / PC
- Koneksi Internet

Software

- Text editor (Visual Studio Code, Sublime, Atom, Notepad++, dll)
- Laragon / XAMPP
- Composer

Tambahan (Extension Visual Studio Code)

- Live Server by Ritwick dey
- HTML CSS Support by Ecmel
- Prettier – Code Formatter by Prettier

MATERI



APA ITU PHP FRAMEWORK?

Framework PHP adalah sebuah platform yang digunakan sebagai kerangka kerja untuk membangun aplikasi web berbasis PHP. Framework PHP berisi kumpulan template kode yang sering digunakan, yang dapat membantu pengembang untuk membuat aplikasi web dengan lebih cepat.

APA ITU LARAVEL?

Laravel adalah sebuah framework PHP *open source* yang berisi banyak modul dasar untuk mengoptimalkan kinerja PHP dalam pengembangan aplikasi web, terutama karena PHP adalah bahasa pemrograman yang dinamis dan Laravel di sini dapat berperan untuk membuat pengembangan web menjadi lebih cepat, lebih aman, dan lebih sederhana.

Laravel sendiri bekerja di sisi back-end atau sisi server. Selain powerful, Laravel juga mudah dipahami. Dengan mengikuti pola arsitektur model-view-controller (MVC), Laravel dapat mempercepat proses pembuatan aplikasi web. Pada arsitektur MVC, pengembangan dapat dilakukan dengan lebih cepat karena pengembang dapat fokus pada satu bagian saja seperti model (bagian yang mengatur database), view (bagian yang mengatur tampilan ke pengguna), dan controller (bagian yang menghubungkan antara model dan view jika ada permintaan dari pengguna).

KELEBIHAN LARAVEL

Berikut ini adalah penjelasan mengenai kelebihan dari Laravel:

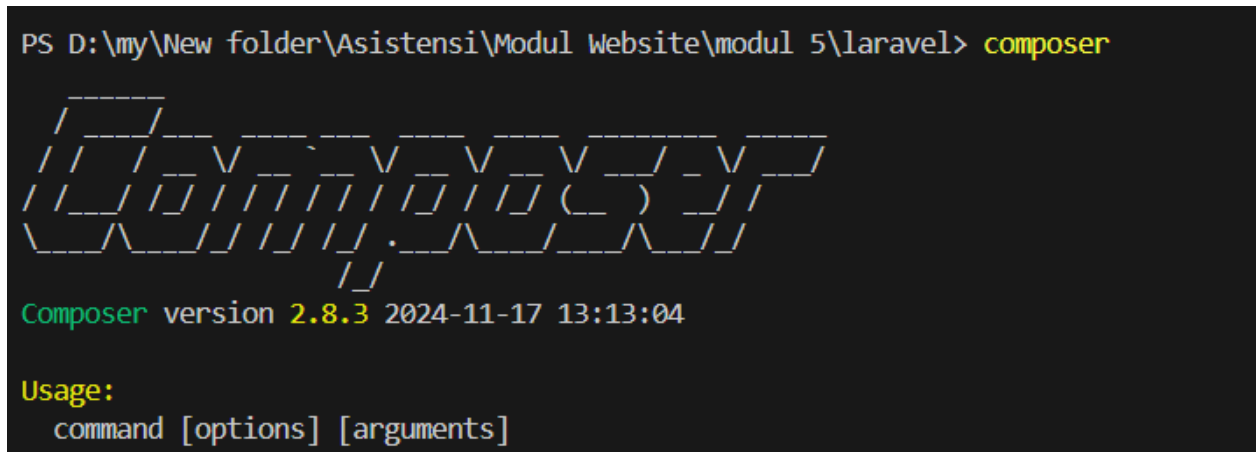
- Mempercepat waktu pengembangan aplikasi karena Laravel menggunakan komponen dari framework lain dan library bawaan dalam mengembangkan aplikasi web.
- Manajemen sumber daya yang lebih mudah karena menggunakan namespace dan interface.
- Performa aplikasi yang lebih baik. Laravel telah lulus uji kualitas dan kecepatan sehingga aplikasi yang dibangun dengan Laravel dapat memiliki performa yang lebih cepat.
- Aplikasi yang dibangun dengan Laravel lebih aman secara default. Aplikasi dapat lebih aman dari CSRF, dan injeksi SQL. Laravel juga dilengkapi dengan beberapa langkah keamanan dengan menerapkan prinsip keamanan OWASP (Open Web Application Security Project).
- Lebih sedikit kode. Dengan menggunakan framework Laravel, dapat menggunakan lebih sedikit kode asli dengan menggunakan fungsi-fungsi bawaan Laravel.
- Dukungan komunitas yang luas. Saat ini ada komunitas Laravel yang besar yang berarti bahwa masalah apa pun yang mungkin dihadapi dapat diselesaikan secara tepat waktu.

Dalam Modul 5 Pemgorgaman Website kali ini, akan dibahas bagaimana melakukan instalasi Laravel 11, menjalankan Storage Link, membuat Model serta Migration yang nantinya digunakan untuk membuat tabel beserta field-field dalam database, kemudian belajar menampilkan data dari database dan membuat proses insert beserta upload gambar dengan mudah menggunakan Laravel 11.

INSTALLASI LARAVEL

Langkah 1 – Instalasi Composer

- Windows : <https://getcomposer.org/doc/00-intro.md#installation-windows>
- Linux & MacOS: <https://getcomposer.org/doc/00-intro.md#installation-linux-unix-macos>



```
PS D:\my\New folder\Asistensi\Modul Website\modul 5\laravel> composer

Composer version 2.8.3 2024-11-17 13:13:04

Usage:
  command [options] [arguments]
```

Langkah 2 – Membuat Project Baru Laravel 11

Setelah melakukan instalasi composer, maka selanjutnya adalah masuk kedalam folder / directory yang telah dibuat kemudian jalankan command dibawah ini pada terminal:

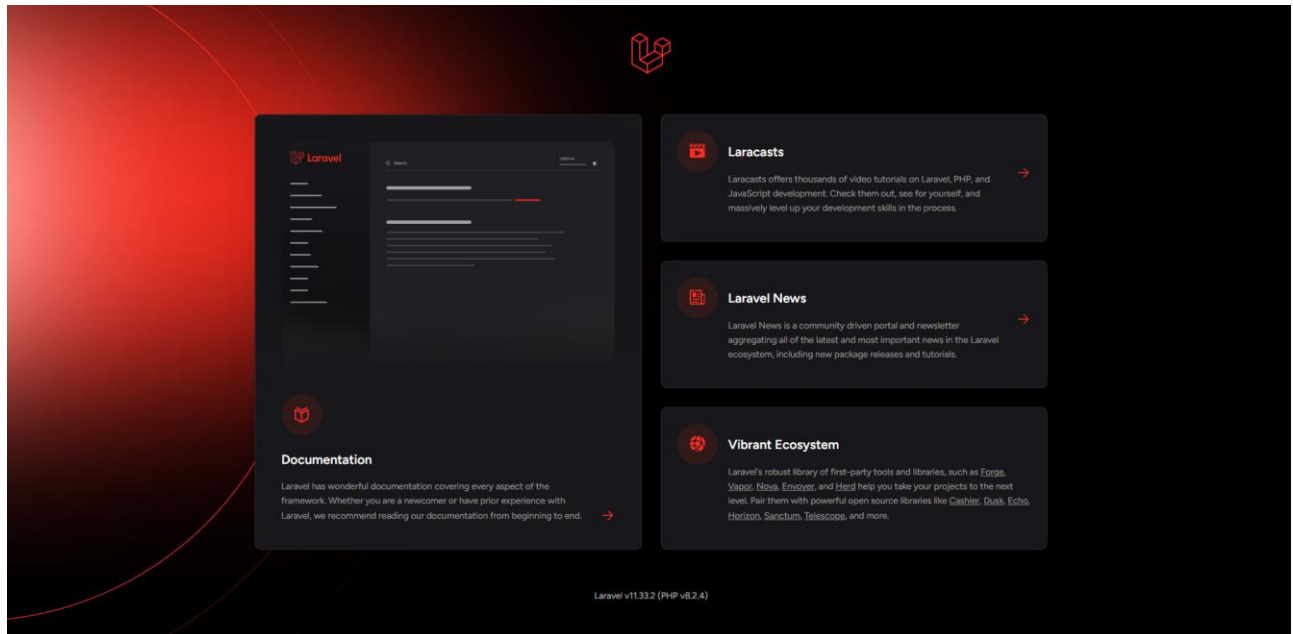
```
composer create-project --prefer-dist laravel/laravel:^11.0 laravel11-api
```

Setelah instalasi selesai, kemudian ketik command “cd Laravel11-api” pada terminal.

Langkah 3 – Mengakses Halaman Laravel

Setelah kedua proses diatas telah dilakukan, maka selanjutnya dapat menjalankan server Laravel. Server dapat dijalankan dengan mengetikkan command berikut pada terminal:

```
php artisan serve
```



Jika berhasil, maka halaman diatas akan muncul ketika membuka url <http://127.0.0.1:8000>

Langkah 4 – Membuat Storage Link

Apa itu Storage Link?

Storage link digunakan agar akses kedalam directory Storage dapat dilakukan. Sehingga file file-file yang di upload dapat tersimpan kedalam folder Storage.

```
php artisan storage:link
```

Dengan menjalankan command diatas, maka storage link akan otomatis terbuat.

Langkah 5 – Konfigurasi Database

Untuk melakukan konfigurasi database, silakan buka file .env yang sudah terbuat otomatis pada Laravel. Kemudian, lakukan konfigurasi sesuai dengan informasi database yang dimiliki.

Dari semula	Menjadi
<pre>DB_CONNECTION=sqlite # DB_HOST=127.0.0.1 # DB_PORT=3306 # DB_DATABASE=laravel # DB_USERNAME=root # DB_PASSWORD=</pre>	<pre>DB_CONNECTION=mysql DB_HOST=127.0.0.1 DB_PORT=3306 DB_DATABASE=db_laravel11_api DB_USERNAME=root DB_PASSWORD=</pre>

Langkah 6 – Membuat Model dan Migration

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

Jika berhasil, maka akan terbuat kedua file ini:

- app\Models\Post.php
- database\migrations\2024_11_23_124414_create_posts_table.php

Langkah 7 – Menambahkan field / kolom pada database

Sebelum bisa menambahkan field pada database, buka terlebih dahulu folder migration yang telah terbuat: `\laravel11-api\database\migrations\2024_11_23_124414_create_posts_table.php`

Kemudian, tambahkan program dibawah ini.

```
public function up(): void
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->string('image');
        $table->string('title');
        $table->text('content');
        $table->timestamps();
    });
}
```

Langkah 8 – Menambahkan Mass Assignment

Mass Assignment digunakan agar field yang dibuat dapat menyimpan nilai. Sebelumnya, silakan buka terlebih dahulu folder: **app/Models/Post.php**

Kemudian, tambahkan program dibawah ini.

```
<?php

namespace App\Models;

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

class Post extends Model
{
    use HasFactory;
}
```

```

/**
 * fillable
 *
 * @var array
 */
protected $fillable = [
    'image',
    'title',
    'content',
];
}

```

Langkah 9 – Menjalankan Proses Migration

Proses migrate berarti akan menjalankan program yang telah ditulis diatas, beserta membuat field database sesuai dengan yang diinginkan.

php artisan migrate

Table	Action	Rows	Type	Collation	Size	Overhead
<input type="checkbox"/> cache	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
<input type="checkbox"/> cache_locks	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
<input type="checkbox"/> failed_jobs	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
<input type="checkbox"/> jobs	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
<input type="checkbox"/> job_batches	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
<input type="checkbox"/> migrations	★ Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
<input type="checkbox"/> password_reset_tokens	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
<input type="checkbox"/> posts	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
<input type="checkbox"/> sessions	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	48.0 KiB	-
<input type="checkbox"/> users	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
10 tables	Sum	4	InnoDB	utf8mb4_general_ci	240.0 KiB	0 B

Note: Jika terdapat error “Unknown collation: 'utf8mb4_0900_ai_ci'” pada proses migrasi, coba untuk gunakan cara ini

<https://meetanshi.com/blog/error-1273-unknown-collation-utf8mb4-0900-ai-ci-in-mysql/>

Langkah 10 – Menambahkan Accessor pada Model

Accessor digunakan untuk memberikan nilai pada saat Field diakses. Untuk mendefinisikan Accessor, dapat membuat method di dalam Model untuk menentukan Field yang akan diakses.

Sekarang, coba untuk menambahkan Accessor untuk Field Image. Sebelumnya, buka kembali folder: **app/Models/Post.php**, kemudian ubah kode menjadi seperti ini.

```

<?php

namespace App\Models;

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

class Post extends Model
{
    use HasFactory;

    /**
     * fillable
     *
     * @var array
     */
    protected $fillable = [
        'image',
        'title',
        'content',
    ];

    /**
     * image
     *
     * @return Attribute
     */
    protected function image(): Attribute
    {
        return Attribute::make(
            get: fn ($image) => url('/storage/posts/' . $image),
        );
    }
}

```

Dari perubahan kode diatas, maka ketika mengakses image maka akan otomatis menghasilkan output seperti berikut: **website.com/storage/posts/nama_file_image.png**

Namun, jika tidak menggunakan Accessor maka hasilnya akan seperti ini: **nama_file_image.png**

Langkah 11 – Membuat API Resources

Untuk dapat membuat API Resources, dapat menjalankan command sebagai berikut:

```
php artisan make:resource PostResource
```

Setelah menjalankan command diatas, selanjutnya silakan untuk akses file:

app/Http/Resources/PostResource.php

Kemudian tambahkan kode dibawah ini:

```
<?php

namespace App\Http\Resources;

use Illuminate\Http\Request;
use Illuminate\Http\Resources\Json\JsonResource;

class PostResource extends JsonResource
{
    //define properti
    public $status;
    public $message;
    public $resource;

    /**
     * __construct
     *
     * @param mixed $status
     * @param mixed $message
     * @param mixed $resource
     * @return void
     */
    public function __construct($status, $message, $resource)
    {
        parent::__construct($resource);
        $this->status = $status;
        $this->message = $message;
    }

    /**
     * toArray

```

```

*
* @param mixed $request
* @return array
*/
public function toArray(Request $request): array
{
    return [
        'success' => $this->status,
        'message' => $this->message,
        'data'    => $this->resource
    ];
}
}

```

Dari kode diatas, dapat dilihat terdapat 3 penambahan property yaitu:

```

//define properti
public $status;
public $message;
public $resource;

```

Properti ini digunakan untuk menyimpan status, pesan, dan data yang akan dikirimkan dalam format JSON.

```

public function __construct($status, $message, $resource)
{
    parent::__construct($resource);
    $this->status = $status;
    $this->message = $message;
}

```

Kemudian pada kode diatas, ditambahkan 3 parameter yang dapat menerima ketiga properti yang telah dibuat. Jadi, tujuan dari PostResource ini adalah untuk membuat data dari Model Post ke dalam format JSON yang sesuai dengan kebutuhan.

Langkah 12 – Membuat Controller Post

Kemudian, jalankan program dibawah ini untuk membuat Controller Post

```
php artisan make:controller Api/PostController
```

Jika command diatas berhasil dijalankan, maka dapat dilihat bahwa terdapat file baru pada folder: **app/Http/Contollers/Api/PostController.php**. Silakan buka file tersebut kemudian ganti menjadi kode dibawah ini:

```
<?php

namespace App\Http\Controllers\Api;

//import model Post
use App\Models\Post;

use App\Http\Controllers\Controller;

//import resource PostResource
use App\Http\Resources\PostResource;

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

        //return collection of posts as a resource
        return new PostResource(true, 'List Data Posts', $posts);
    }
}
```

Dari kode diatas, pertama import Model **Post** terlebih dahulu. Setelah itu, import juga API Resource yang sudah dibuat sebelumnya, yaitu **PostResource**. Di dalam class **PostContoller** ditambahkan method baru dengan nama **index**. Di dalam method tersebut, dilakukan get data dari database melalui model **Post**. Setelah itu, parsing variable **\$posts** di atas ke dalam **RostResource**.

Langkah 13 – Membuat Route Rest API

Pada Laravel 11, API sudah tidak lagi tersedia secara default saat proses instalasi. Untuk tetap dapat mendapatkan fitur API, dapat menjalankan command sebagai berikut:

```
php artisan install:api
```

Jika perintah diatas dijalankan, maka secara otomatis melakukan download library yang bernama sanctum, tapi disini tidak akan menggunakan library tersebut untuk sekarang. Dan sekarang, route untuk Rest API sudah tersedia di dalam folder: **routes/api.php**. Sekarang, silakan buka file tersebut kemudian tambahkan kode dibawah ini:

```
<?php

use Illuminate\Auth\Middleware\Authenticate;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

Route::get('/user', function (Request $request) {
    return $request->user();
})->middleware(Authenticate::using('sanctum'));

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

Setelah ditambahkan kode tersebut, maka selanjutnya adalah memastikan route sudah terpasang dengan command dibawah ini

```
php artisan route:list
```

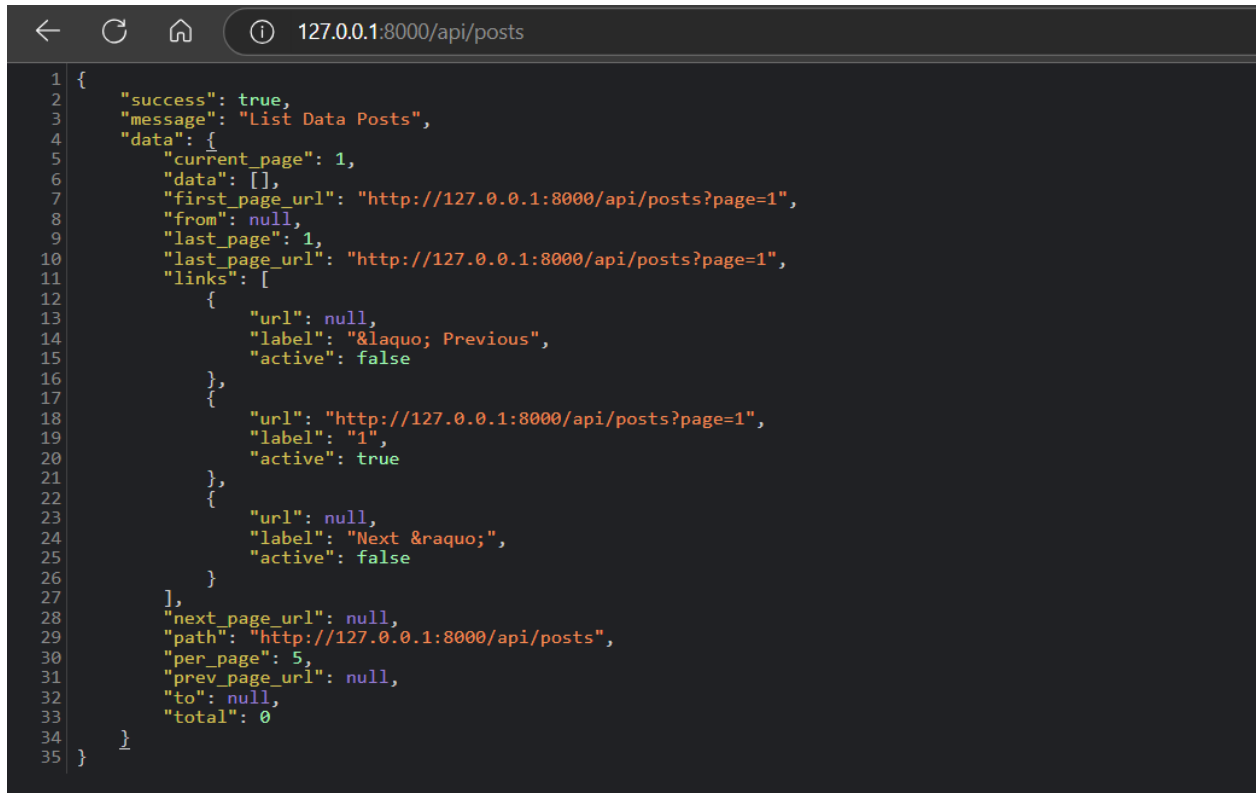
```
PS D:\my\New folder\Asistensi\Modul Website\modul 5\laravel\laravel11-api> php artisan route:list

GET|HEAD      / ..... ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController
POST          _ignition/execute-solution ..... ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController
GET|HEAD      _ignition/health-check ..... ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController
POST          _ignition/health-check ..... ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController
GET|HEAD      _ignition/update-config ..... ignition.updateConfig > Spatie\LaravelIgnition > UpdateConfigController
POST          api/posts ..... posts.index > Api\PostController@index
GET|HEAD      api/posts ..... posts.store > Api\PostController@store
PUT|PATCH    api/posts/{post} ..... posts.show > Api\PostController@show
DELETE        api/posts/{post} ..... posts.update > Api\PostController@update
GET|HEAD      api/user ..... posts.destroy > Api\PostController@destroy
GET|HEAD      sanctum/csrf-cookie ..... sanctum.csrf-cookie > Laravel\Sanctum > CsrfCookieController@show
GET|HEAD      up .....

Showing [12] routes
```

Langkah 13 – Uji Coba Rest API

Sekarang, silakan buka kembali POSTMAN pada modul 4. Kemudian, masukkan URL berikut: <http://localhost:8000/api/posts> dan untuk method-nya silakan gunakan **GET**. Jika sudah, silakan klik **SEND** dan jika berhasil maka akan mendapatkan hasil seperti dibawah ini.



```

1 {
2   "success": true,
3   "message": "List Data Posts",
4   "data": {
5     "current_page": 1,
6     "data": [],
7     "first_page_url": "http://127.0.0.1:8000/api/posts?page=1",
8     "from": null,
9     "last_page": 1,
10    "last_page_url": "http://127.0.0.1:8000/api/posts?page=1",
11    "links": [
12      {
13        "url": null,
14        "label": "&laquo; Previous",
15        "active": false
16      },
17      {
18        "url": "http://127.0.0.1:8000/api/posts?page=1",
19        "label": "1",
20        "active": true
21      },
22      {
23        "url": null,
24        "label": "Next &raquo;",
25        "active": false
26      }
27    ],
28    "next_page_url": null,
29    "path": "http://127.0.0.1:8000/api/posts",
30    "per_page": 5,
31    "prev_page_url": null,
32    "to": null,
33    "total": 0
34  }
35 }

```

Dapat dilihat bahwa belum terdapat data yang masuk karena memang belum ditambahkan kedalam database untuk data yang ingin ditampilkan.

Langkah 14 – Menambahkan Method Store

Disini, method store ditambahkan untuk dapat melakukan insert data pada database. Silakan buka file: **app/Http/Controllers/Api/PostController.php** kemudian ubah kode menjadi seperti dibawah ini:

```

<?php

namespace App\Http\Controllers\Api;

//import model Post
use App\Models\Post;

use App\Http\Controllers\Controller;

//import resource PostResource
use App\Http\Resources\PostResource;

//import Http request
use Illuminate\Http\Request;

//import facade Validator

```

```

use Illuminate\Support\Facades\Validator;

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

        //return collection of posts as a resource
        return new PostResource(true, 'List Data Posts', $posts);
    }

    /**
     * store
     *
     * @param mixed $request
     * @return void
     */
    public function store(Request $request)
    {
        //define validation rules
        $validator = Validator::make($request->all(), [
            'image'      => 'required|image|mimes:jpeg,png,jpg,gif,svg|max:2048',
            'title'       => 'required',
            'content'     => 'required',
        ]);

        //check if validation fails
        if ($validator->fails()) {
            return response()->json($validator->errors(), 422);
        }

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

        //create post
        $post = Post::create([
            'image'      => $image->hashName(),

```

```

        'title'    => $request->title,
        'content'  => $request->content,
    ]);

    //return response
    return new PostResource(true, 'Data Post Berhasil Ditambahkan!', $post);
}
}

```

Dari perubahan kode diatas, pertama import http request agar controller dapat menerima request yang dikirim oleh pengguna. Karena ada proses insert data ke dalam database, maka perlu memberikan sebuah validasi terhadap data yang akan di insert. Oleh sebab itu, import Facades Validator diperlukan.

Dalam class **PostController**, method baru dibuat menggunakan nama **store**. Dalam method tersebut, dibuat kode untuk melakukan validasi request yang masuk.

```

{
    //define validation rules
    $validator = Validator::make($request->all(), [
        'image'    => 'required|image|mimes:jpeg,png,jpg,gif,svg|max:2048',
        'title'    => 'required',
        'content'  => 'required',
    ]);
}

```

Jika request tidak sesuai dengan validasi diatas, maka akan melakukan return ke dalam format JSON dengan status kode 422 dan memberikan informasi error. Tapi, jika semua request sesuai dengan yang diharapkan, maka hal pertama yang dilakukan adalah upload gambar menggunakan **storeAs**.

```

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

```

Setelah upload file gambar berhasil, Langkah berikutnya adalah menjalankan proses insert data ke dalam database menggunakan Model.

```

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

```

Langkah 15 – Uji Coba Rest API Insert

Silakan buka kembali postman, kemudian masukkan url: <http://localhost:8000/api/posts> dan method nya adalah **POST**. Setelah itu, masuk kedalam tab body, dan pilih form-data, kemudian masukkan key dan value sebagai berikut:

Key	Type	Value
image	file	(pilih gambar dari computer)
title	text	Belajar Laravel Modul 5
content	text	Belajar Laravel Modul 5

Jika berhasil, maka akan mendapatkan hasil seperti ini:

```
{
  "success": true,
  "message": "Data Post Berhasil Ditambahkan!",
  "data": {
    "image":
"http://localhost:8000/storage/posts/NuJYSsdyeNRPoiOPkN4VXhvtDSnYSvsyJTodiYNt.png",
    "title": "Belajar Laravel Modul 5",
    "content": "Belajar Laravel Modul 5",
    "updated_at": "2024-02-10T05:59:53.000000Z",
    "created_at": "2024-02-10T05:59:53.000000Z",
    "id": 1
  }
}
```

Langkah 16 – Menambahkan Method Show

Silakan buka file Controller pada path: **app/Http/Controlles/Api/PostController.php** kemudian ubah kode menjadi seperti dibawah ini:

```
<?php

namespace App\Http\Controllers\Api;

//import model Post
use App\Models\Post;

use App\Http\Controllers\Controller;

//import resource PostResource
use App\Http\Resources\PostResource;
```



```

//import Http request
use Illuminate\Http\Request;

//import facade Validator
use Illuminate\Support\Facades\Validator;

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

        //return collection of posts as a resource
        return new PostResource(true, 'List Data Posts', $posts);
    }

    /**
     * store
     *
     * @param mixed $request
     * @return void
     */
    public function store(Request $request)
    {
        //define validation rules
        $validator = Validator::make($request->all(), [
            'image' => 'required|image|mimes:jpeg,png,jpg,gif,svg|max:2048',
            'title' => 'required',
            'content' => 'required',
        ]);

        //check if validation fails
        if ($validator->fails()) {
            return response()->json($validator->errors(), 422);
        }

        //upload image
        $image = $request->file('image');
    }
}

```

```

$image->storeAs('public/posts', $image->hashName());

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

//return response
return new PostResource(true, 'Data Post Berhasil Ditambahkan!', $post);
}

/**
 * show
 *
 * @param mixed $id
 * @return void
 */
public function show($id)
{
    //find post by ID
    $post = Post::find($id);

    //return single post as a resource
    return new PostResource(true, 'Detail Data Post!', $post);
}
}

```

Dari perubahan kode diatas, maka dapat dilihat bahwa terdapat penambahan method **show** dengan parameter **\$id**. Di dalam method tersebut, dapat melakukan get data ke dalam database menggunakan Model **POST** dengan method **find** dan diberikan parameter **\$id**.

```

/**
public function show($id)
{
    //find post by ID
    $post = Post::find($id);

```

Jika data ditemukan, maka akan melakukan return menggunakan **PostResource** dengan mengirim data post yang sudah di get.

Langkah 17 – Uji Coba Rest API

Silakan kembali buka postman kemudian isikan url ini: <http://localhost:8000/api/posts/1> dan untuk method nya adalah **GET**. Jika step tersebut sudah dilakukan, maka seharusnya akan muncul seperti dibawah ini:

```
{
  "success": true,
  "message": "Detail Data Post!",
  "data": {
    "id": 1,
    "image":
"http://localhost:8000/storage/posts/NuJYSsdyenRPoiOPkN4VXhvtDSnYSvsyJTodiYNT.png",
    "title": "Belajar Laravel Modul 5",
    "content": "Belajar Laravel Modul 5",
    "created_at": "2024-02-10T05:59:53.000000Z",
    "updated_at": "2024-02-10T05:59:53.000000Z"
  }
}
```

Langkah 18 – Menambahkan Method Update

Sekarang, buka file controller **PostController** yang ada di dalam folder: **app/Http/Controllers/Api/PostController.php**, kemudian semua kode rubah menjadi dibawah ini:

<https://pastecode.io/s/775bajtt>

Dari perubahan kode diatas, didalam class **PostController** ditambahkan method baru dengan nama **update** yang memiliki 2 parameter **\$request** dan **\$id**. Dimana parameter **\$request** akan berisi data yang dikirimkan pengguna seperti title, gambar atau content. Sedangkan, untuk parameter **\$id** akan berisi id post yang dibutuhkan.

Didalam method tersebut, dibuat validasi request yang masuk.

```
//define validation rules
$validator = Validator::make($request->all(), [
    'title' => 'required',
    'content' => 'required',
]);
```

Dari validasi diatas, berikut ini merupakan penjelasannya.

KEY	VALIDATION	DESCRIPTION
title	required	(wajib diisi)
content	required	(wajib diisi)

Kenapa **image** tidak wajib diberikan required? karena gambar akan bersifat opsional dan bisa dikosongkan.

Setelah itu, untuk melakukan get data post dari database perlu ditambahkan Model **Post** dengan method **find** dan berdasarkan **\$id**.

```
//find post by ID
$post = Post::find($id);
```

Kemudian, dibuat kondisi untuk melakukan pengecekan adanya request file yang bernama **image** atau tidak.

```
//check if image is not empty
if ($request->hasFile('image')) {

    // code

} else {

    // code

}
```

Jika terdapat sebuah requests **image** maka artinya akan masuk ke dalam method upload gambar. Kemudian, gambar baru akan tersimpan dan gambar lama akan dihapus.

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

//delete old image
Storage::delete('public/posts/' . basename($post->image));

//update post with new image
$post->update([
    'image' => $image->hashName(),
    'title' => $request->title,
    'content' => $request->content,
]);
```

Kemudian, jika program berhasil dilakukan maka akan menghasilkan response JSON berhasil.

```
//return response
return new PostResource(true, 'Data Post Berhasil Diubah!', $post);
```

Langkah 19 – Uji Coba Rest API

Silakan buka kembali postman, kemudian masukkan URL sebagai berikut: <http://localhost:8000/api/posts/1>. Kemudian, pilih tab **body** dan pilih **form—data**, kemdian masukkan key dan value berikut:

KEY	TYPE	VALUE
image	file	(Opsional)
title	text	Belajar Laravel Modul 5 - Edit
content	text	Belajar Laravel Modul 5 – Edit
_method	text	PUT

Diatas, ditambahkan **method PUT** untuk melakukan update data. Jika berhasil, maka seharusnya akan menampilkan data dibawah ini:

```
{
  "success": true,
  "message": "Data Post Berhasil Diubah!",
  "data": {
    "id": 1,
    "image": "http://localhost:8000/storage/posts/NuJYSsdyeNRPoiOPkN4VXhvtDSnYSvsyjTodiYNT.png",
    "title": "Belajar Laravel Modul 5 - Edit",
    "content": "Belajar Laravel Modul 5 - Edit",
    "created_at": "2024-02-10T05:59:53.000000Z",
```

```

    "updated_at": "2024-02-12T06:01:14.000000Z"
  }
}

```

Langkah 20 – Delete Data

Silakan buka kembali file Controller pada folder: **app/Http/Controllers/Api/PostController.php**, kemudian rubah kode menjadi dibawah ini:

<https://pastecode.io/s/ixcstiuj>

Dari kode diatas, ditambahkan method baru bernama **destroy** dan dialam parameternya diberikan value **\$id**.

```

public function destroy($id)
{
    //find post by ID
    $post = Post::find($id);

    //delete image
    Storage::delete('public/posts/'.basename($post->image));

    //delete post
    $post->delete();

    //return response
    return new PostResource(true, 'Data Post Berhasil Dihapus!', null);
}

```

Dimana didalam kode tersebut, dilakukan get data melalui **\$id**, dan jika data berhasil didapatkan maka langkah selanjutnya adalah menghapus file gambar dari server menggunakan Facades **Storage**. Kemudian delete data menggunakan method **delete()**. Terakhir, jika berhasil maka akan mengreturn dalam format JSON.

Langkah 21 – Uji Coba Rest API

Sekarang buka kembali aplikasi postman, kemudian masukkan url berikut: <http://localhost:8000/api/posts/1> dengan method nya adalah **DELETE**. Jika sudah, klik **SEND** dan jika berhasil maka akan mendapatkan response JSON sebagai berikut:

```

{
  "success": true,
  "message": "Data Post Berhasil Dihapus!",
  "data": null
}

```

AKTIFITAS LAB

CODELAB 1

Ikuti materi pada **INSTALLASI LARAVEL** kemudian pastikan Laravel 11 dapat berjalan dengan normal sesuai dengan contoh pada local pc masing masing. Kemudian lakukan upload ke github sebagai bukti sudah menyelesaikan Codelab 1 Modul 5.

TUGAS PRAKTIKUM

TUGAS 1

Lakukan proses instalasi Laravel **secara live** pada asisten dari **Langkah 1 – 9** (sampai menjalankan migrate). Dengan waktu maksimal setiap praktikan adalah 10 menit.

Note: gunakan nama field database sesuai dengan tema, dan tanpa copy paste.

TUGAS 2

Buatlah Rest API menggunakan Laravel sesuai dengan tema website yang telah ditentukan kemudian pastikan dapat melakukan CRUD menggunakan postman, dan terdapat minimal 2 tabel CRUD yang berbeda dengan route yang juga berbeda.

Note: dilarang merubah tema dari yang telah ditetapkan, jika terdapat perubahan tema maka dapat dipastikan nilai maksimal tugas 2 adalah C+.

KRITERIA & DETAIL PENILAIAN

Kriteria Penilaian	Persentase Penilaian
Tugas 1	20%
Tugas 2	35%
Pemahaman	45%