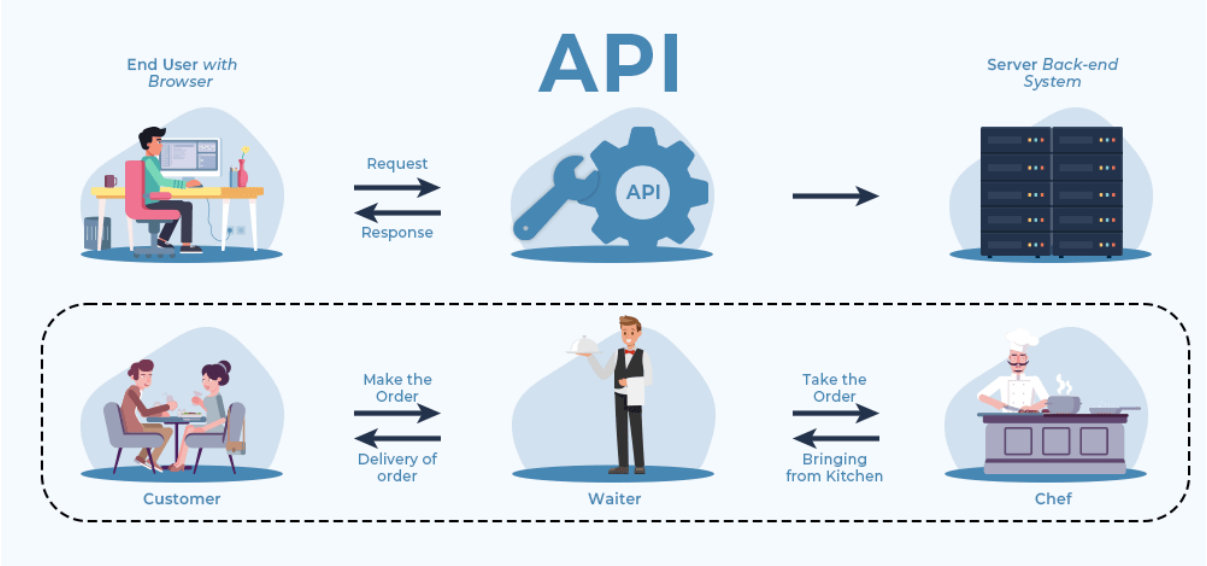
API dan Penggunaan API di Laravel

Mata Kuliah	: Pemrograman Web 2
Dosen	: Jamal Apriadi, S.Kom
Semester	: 4
Tahun	: 2025/2026

API (Application Programming Interface)

API (Application Programming Interface) adalah cara bagi dua aplikasi atau sistem komputer untuk saling berkomunikasi dan bertukar informasi tanpa harus mengetahui detail internal satu sama lain.



Bagaimana Cara Kerja API?

1. Permintaan (Request): Aplikasi klien (misalnya, sebuah website atau aplikasi mobile) mengirimkan permintaan ke API. Permintaan ini biasanya berupa data yang diminta atau tindakan yang ingin dilakukan (misalnya, melihat data product).
2. Pemrosesan: Server yang menjalankan API menerima permintaan tersebut dan memprosesnya. Ini mungkin melibatkan pengambilan data dari database, melakukan perhitungan, atau memvalidasi input.
3. Respons (Response): Server kemudian mengirimkan respons kembali ke aplikasi klien. Respons ini berisi data yang diminta atau hasil dari tindakan yang dilakukan.

Jenis-Jenis API:

1. RESTful API (Representational State Transfer): Ini adalah jenis API yang paling umum digunakan saat ini. RESTful API menggunakan prinsip-prinsip berikut:
    - Stateless: Setiap permintaan harus berisi semua informasi yang diperlukan untuk diproses oleh server. Server tidak menyimpan informasi tentang sesi pengguna sebelumnya.
    - Client-Server: Ada pemisahan yang jelas antara klien (aplikasi yang meminta data) dan server (aplikasi yang menyediakan data).
    - Cacheable: Response API dapat di-cache untuk meningkatkan kinerja.
    - Layered System: Aplikasi klien tidak perlu tahu apakah mereka terhubung langsung ke server atau melalui perantara.
    - Uniform Interface: Semua permintaan dan respons mengikuti standar tertentu (misalnya, menggunakan HTTP methods seperti GET, POST, PUT, DELETE).
2. GraphQL API: Alternatif untuk RESTful API yang lebih fleksibel. Klien dapat meminta hanya data yang mereka butuhkan, bukan seluruh dataset. Ini mengurangi jumlah data yang ditransfer dan meningkatkan kinerja.
3. SOAP (Simple Object Access Protocol): Jenis API yang lebih tua dan kurang umum digunakan dibandingkan RESTful API. SOAP menggunakan XML untuk mengirimkan pesan antar aplikasi.

Membuat API Resources

API Resources di Laravel adalah fitur yang digunakan untuk mengubah data Model Eloquent menjadi format JSON yang lebih terstruktur dan mudah dikendalikan. Ini berguna saat membangun API, sehingga kita dapat mengontrol bagaimana data dikirim ke client.

Kenapa Menggunakan API Resources?

- Memisahkan Logika Data & Presentasi - API Resources membantu kita mengontrol data apa saja yang dikirim ke client tanpa mengubah model atau controller.
- Keamanan Data - Mencegah data sensitif (seperti password) dikirim ke client secara tidak sengaja.
- Konsistensi Format Response - Memudahkan frontend untuk menerima data dengan format yang jelas.

Praktek:

1. Membuat API Resources

```
php artisan make:resource ProductCategoryResource
```

```
php artisan make:resource ProductResource
```

Jika perintah di atas berhasil dijalankan, maka kita akan mendapatkan 2 file baru yang berada di dalam folder app/Http/Resources/ProductCategoryResource.php dan app/Http/Resources/ProductResource.php.

2. Custom Response

   secara default API Resource yang telah digenerate sudah bisa langsung digunakan, tapi karena kita akan menyesuaikan format JSON yang diinginkan, maka kita perlu melakukan sedikit modifikasi di dalam file tersebut.

   a. ProductCategoryResponse.php

```php
<?php

namespace App\Http\Resources;

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

class ProductCategoryResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @return array<string, mixed>
     */
    public $status;
    public $message;
    public $resource;

    public function __construct($resource, $status = 200, $message = 'Success')
    {
        parent::__construct($resource);
        $this->status = $status;
        $this->message = $message;
    }

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

Penjelasan:

1. Namespace:

```
namespace App\Http\Resources;
```

- Ini menentukan namespace tempat kelas **ProductCategoryResource** berada. Namespace ini membantu mengatur kode dan menghindari konflik nama dengan kelas lain dalam aplikasi Laravel kamu.

2. Import Statements:

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

- **Illuminate\Http\Request**: Kelas ini digunakan untuk mengakses informasi tentang permintaan HTTP (misalnya, parameter query, header). Meskipun tidak langsung digunakan dalam kode ini, seringkali berguna saat membuat API Resource.
- **Illuminate\Http\Resources\Json\JsonResource**: Ini adalah kelas dasar yang diwarisi oleh **ProductCategoryResource**. Kelas ini menyediakan metode dan properti untuk meng-serialize data ke format JSON.

3. Class Definition:

```
class ProductCategoryResource extends JsonResource
{
    // ... kode di dalam class ...
}
```

- Ini mendefinisikan kelas **ProductCategoryResource** yang mewarisi dari **JsonResource**. Inheritance ini berarti bahwa **ProductCategoryResource** akan mendapatkan semua properti dan metode dari **JsonResource**, serta dapat menambahkan properti dan metode sendiri.

4. Properties:

```
/**
     * Transform the resource into an array.
     *
     * @return array<string, mixed>
     */
    public $status;
    public $message;
    public $resource;
```

- **$status**: Properti ini menyimpan kode status HTTP (misalnya, 200 untuk sukses, 400 untuk kesalahan).
- **$message**: Properti ini menyimpan pesan teks yang menjelaskan hasil operasi.
- **$resource**: Properti ini menyimpan data model yang akan di-serialize ke JSON.

5. Constructor:

```
public function __construct($resource, $status = 200, $message = 'Success')
{
        parent::__construct($resource);
        $this->status = $status;
        $this->message = $message;
}
```

- Ini adalah constructor dari kelas **ProductCategoryResource**. Constructor ini menerima tiga argumen:
  - **$resource**: Objek model yang akan di-serialize.
  - **$status**: Kode status HTTP (defaultnya 200).
  - **$message**: Pesan teks (defaultnya "Success").
- **parent::__construct($resource);**: Ini memanggil constructor dari kelas induk (**JsonResource**) untuk melakukan inisialisasi dasar.
- **$this->status = $status;** dan **$this->message = $message;**: Ini menetapkan nilai properti **$status** dan **$message** berdasarkan argumen yang diberikan.

6. **toArray()** Method:

```php
public function toArray(Request $request): array
{
    return [
        'success' => $this->status,
        'message' => $this->message,
        'data' => $this->resource
    ];
}
```

- Ini adalah metode yang paling penting dalam kelas **ProductCategoryResource**. Metode ini bertanggung jawab untuk meng-serialize data model ke format JSON.
- **Request $request**: Argumen ini memungkinkan kamu mengakses informasi tentang permintaan HTTP (meskipun tidak digunakan secara langsung dalam kode ini).
- **return [...]**: Kode ini membuat array asosiatif yang berisi:
    - **'success' => $this->status**: Menyertakan kode status HTTP.
    - **'message' => $this->message**: Menyertakan pesan teks.
    - **'data' => $this->resource**: Menyertakan data model yang telah di-serialize.

b. ProductResponse.php

```php
<?php

namespace App\Http\Resources;

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

class ProductResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @return array<string, mixed>
     */
    public $status;
    public $message;
    public $resource;

    public function __construct($resource, $status = 200, $message = 'Success')
    {
        parent::__construct($resource);
        $this->status = $status;
        $this->message = $message;
    }

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

MEMBUAT API CONTROLLER

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

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

Jika perintah di atas berhasil dijalankan, maka kita akan mendapatkan file baru yang berada di dalam folder **app/Http/Controllers/Api/ProductCategoryController.php** dan **app/Http/Controllers/Api/ProductController.php.**

Silahkan buka file tersebut, kemudian ubah semua kode-nya menjadi seperti berikut ini.

app/Http/Controllers/Api/ProductCategoryController.php

```php
<?php

namespace App\Http\Controllers\Api;

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

use App\Models\Categories;
use App\Http\Resources\ProductCategoryResource;

class ProductCategoryController extends Controller
{
    public function index(Request $request)
    {
        $categories = Categories::latest()->paginate(10);

        return new ProductCategoryResource($categories, 200, 'List Data Product Category');
    }
}
```

Penejelasan:
1. Namespace:

```
namespace App\Http\Controllers\Api;
```

- Ini menentukan namespace tempat controller berada. Namespace ini membantu mengatur kode dan menghindari konflik nama dengan kelas lain dalam aplikasi Laravel kamu.

2. Import Statements:

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

use App\Models\Categories;
use App\Http\Resources\ProductCategoryResource;
```

- **App\Http\Controllers\Controller:** Kelas dasar untuk semua controller di Laravel. Ini menyediakan metode dan properti umum yang dibutuhkan oleh controller.
- **Illuminate\Http\Request**: Kelas ini digunakan untuk mengakses informasi tentang permintaan HTTP (misalnya, parameter query, header).
- **App\Models\Categories**: Model database yang merepresentasikan tabel kategori produk. Ini adalah model yang akan kita gunakan untuk mengambil data kategori dari database.
- **App\Http\Resources\ProductCategoryResource**: Resource class yang telah kita definisikan sebelumnya (lihat penjelasan kode sebelumnya) untuk meng-serialize data kategori menjadi format JSON dengan struktur tertentu.

3. Class Definition:

```
class ProductCategoryController extends Controller
{
    // ... kode di dalam class ...
}
```

- Ini mendefinisikan kelas **ProductCategoryController** yang mewarisi dari **Controller**. Inheritance ini berarti bahwa **ProductCategoryController** akan mendapatkan semua properti dan metode dari **Controller**, serta dapat menambahkan properti dan metode sendiri.

4. **index()** Method:

```
public function index()
{
    $categories = Categories::latest()->paginate(10);

    return new ProductCategoryResource($categories, 200, 'List Data Product Category');
}
```

- Ini adalah method yang akan dipanggil ketika endpoint **/api/product-categories** diakses (biasanya melalui HTTP GET).
- **$categories = Categories::latest()->paginate(10);**: Baris ini melakukan hal berikut:
    - **Categories::latest()**: Mengambil semua kategori produk terbaru berdasarkan kolom **created_at** (atau kolom tanggal lain yang relevan) dalam urutan menurun.
    - **->paginate(10)**: Membagi hasil menjadi halaman-halaman dengan ukuran 10 item per halaman. Ini memungkinkan kamu untuk menampilkan daftar kategori yang panjang tanpa membebani browser klien. Laravel akan secara otomatis menambahkan link pagination ke respons JSON.
- **return new ProductCategoryResource(true, 'List Data Product Category', $categories);**: Baris ini membuat instance dari **ProductCategoryResource** dan mengembalikannya sebagai respons API.
    - **true**: Ini adalah argumen boolean yang digunakan oleh konstruktor **ProductCategoryResource**. Dalam kasus ini, kemungkinan besar digunakan untuk menunjukkan bahwa operasi berhasil (atau setara dengan itu).
    - **'List Data Product Category'**: Ini adalah pesan teks yang akan disertakan dalam respons JSON.
    - **$categories**: Objek koleksi kategori produk yang telah di-paginate. Ini akan di-serialize menjadi bagian **data** dari respons JSON.

app/Http/Controllers/Api/ProductController.php

```php
<?php

namespace App\Http\Controllers\Api;

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

use App\Models\Product;
use App\Http\Resources\ProductResource;

class ProductController extends Controller
{
    public function index(Request $request)
    {
        $products = Product::latest()->paginate(10);

        return new ProductResource($products, 200, 'List Data Product');
    }
}
```

Membuat Route REST API untuk Menampilkan Data

secara default, laravel 12 tidak menampilkan file route untuk api, maka dari itu kita perlu memanggilnya terlebih dahulu dengan perintah sebagai berikut:

```
php artisan install:api
```

Jika perintah diatas dijalankan, maka akan secara otomatis mengunduh library yang bernama sanctum, tapi tidak akan menggunakan library tersebut untuk saat ini. Dan sekarang route untuk Rest API sudah tersedia di dalam folder routes/api.php. Silahkan buka file tersebut dan ubah kode-nya menjadi seperti berikut ini.

```php
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

//panggil dulu controllernya
use App\Http\Controllers\Api\ProductCategoryController;
use App\Http\Controllers\Api\ProductController;

Route::apiResource('/product-categories', ProductCategoryController::class)->only('index');

Route::apiResource('/products', ProductController::class)->only('index');

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

UJI COBA

buka aplikasi postman, dan ketikan url dengan method **GET**

- http://127.0.0.1:8000/api/product-categories
- http://127.0.0.1:8000/api/products

Insert data ke Database menggunakan API

langkah selanjutkan kita akan membuat sebuah fungsi untuk menyimpan data melalui api.

Langkah 1: Api/ProductCategoryController.php

Tambahkan fungsi store pada **Api/ProductCategoryController.php.**

```
public function store(Request $request)
{
    $validator = \Validator::make($request->all(), [
        'name' => 'required|string|max:255',
        'slug' => 'required|string|max:255',
        'description' => 'required'
    ]);

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

    $category = new Categories;
    $category->name = $request->name;
    $category->slug = $request->slug;
    $category->description = $request->description;

    if ($request->hasFile('image')) {
        $image = $request->file('image');
        $imageName = time() . '_' . $image->getClientOriginalName();
        $imagePath = $image->storeAs('uploads/categories', $imageName, 'public');
        $category->image = $imagePath;
    }

    $category->save();

    return new ProductCategoryResource($category, 201, 'Product Category Created Successfully');
}
```

Penjelasan:

1. **public function store(Request $request)**: Mendefinisikan method **store** pada controller **ProductCategoryController**. Method ini menerima objek **Request** sebagai parameter, yang berisi data dari permintaan HTTP (biasanya dalam format JSON).
2. **$validator = \Validator::make($request->all(), [...]);**: Membuat instance validator Laravel.
    - **\Validator::make(...)**: Membuat objek validator baru.
    - **$request->all()**: Mengambil semua data yang dikirimkan dalam permintaan HTTP sebagai array asosiatif.
    - **[...]**: Array aturan validasi (validation rules). Setiap aturan menentukan

format dan persyaratan untuk setiap field data. Dalam kasus ini:
    - **'name' => 'required|string|max:255'**: Field **name** harus diisi (**required**), berupa string (**string**), dan tidak boleh lebih dari 255 karakter (**max:255**).
    - **'slug' => 'required|string|max:255'**: Field **slug** harus diisi, berupa string, dan tidak boleh lebih dari 255 karakter.
    - **'description' => 'required'**: Field **description** harus diisi.
3. **if ($validator->fails()) { ... }**: Memeriksa apakah validasi gagal. Jika ada aturan validasi yang tidak terpenuhi, **$validator->fails()** akan mengembalikan **true**.
    - **return response()->json($validator->errors(), 422);**: Jika validasi gagal, method ini mengembalikan respons JSON dengan kode status HTTP 422 (Unprocessable Entity), yang menunjukkan bahwa data yang dikirimkan tidak valid. **$validator->errors()** mengembalikan array yang berisi pesan kesalahan untuk setiap field yang tidak valid.
4. **$category = new Categories;**: Membuat instance baru dari model **Categories**. Pastikan Anda sudah memiliki model **Categories** yang terdefinisi dengan benar di dalam aplikasi Laravel Anda.
5. **$category->name = $request->name;**: Mengisi field **name** pada model **Categories** dengan nilai yang diterima dari request (**$request->name**).
6. **$category->slug = $request->slug;**: Mengisi field **slug** pada model **Categories** dengan nilai yang diterima dari request.
7. $**category->description = $request->description;**: Mengisi field **description** pada model **Categories** dengan nilai yang diterima dari request.
8. **if ($request->hasFile('image')) { ... }**: Memeriksa apakah ada file yang dikirimkan dalam request dengan nama 'image'.
    - **$image = $request->file('image');**: Jika ada file, method ini mengambil objek **UploadedFile** dari permintaan.
    - **$imageName = time() . '_' . $image->getClientOriginalName();**: Membuat nama file baru yang unik menggunakan timestamp dan nama file asli. Ini untuk menghindari konflik nama file.
    - **$imagePath = $image->storeAs('uploads/categories', $imageName, 'public');**: Menyimpan file ke direktori **storage/app/public/uploads/categories. 'public'** adalah driver yang digunakan untuk menyimpan file di folder publik (yang dapat diakses melalui URL).
    - **$category->image = $imagePath;**: Mengisi field **image** pada model Categories dengan path lengkap ke file yang disimpan.
9. **$category->save();**: Menyimpan data dari model **Categories** ke database.
10. **return new ProductCategoryResource($category, 201, 'Product Category Created Successfully',);**: Membuat instance dari resource (misalnya, menggunakan Resource Class) dan mengembalikannya sebagai respons JSON.

- **new ProductCategoryResource(...)**: Membuat instance dari resource yang digunakan untuk merespons data kategori produk. Pastikan Anda memiliki class **ProductCategoryResource** yang terdefinisi dengan benar di dalam aplikasi Laravel Anda.
- **201**: Kode status HTTP 201 (Created) menunjukkan bahwa operasi berhasil dan sumber daya baru telah dibuat.
- **'Product Category Created Successfully'**: Pesan yang akan dikembalikan sebagai bagian dari respons JSON.

Langkah 2: ProductController.php

Tambahkan fungsi store pada **ProductController.php.**

```
public function store(Request $request)
{
    $validator = \Validator::make($request->all(), [
        'name' => 'required|string|max:255',
        'slug' => 'required|string|max:255|unique:products,slug',
        'sku' => 'required|string|unique:products,sku',
        'price' => 'required|numeric|min:0',
        'stock' => 'required|integer|min:0',
        'product_category_id' => 'nullable|exists:product_categories,id',
        'description' => 'nullable|string',
        'image' => 'nullable|image|mimes:jpeg,png,jpg,gif|max:2048',
    ]);

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

    $product = new Product;
    $product->name = $request->name;
    $product->slug = $request->slug;
    $product->description = $request->description;
    $product->sku = $request->sku;
    $product->price = $request->price;
    $product->stock = $request->stock;
    $product->product_category_id = $request->product_category_id;
    $product->is_active = $request->has('is_active') ? $request->is_active : true;

    if ($request->hasFile('image')) {
        $image = $request->file('image');
        $imageName = time() . '_' . $image->getClientOriginalName();
        $imagePath = $image->storeAs('uploads/products', $imageName, 'public');
        $product->image_url = $imagePath;
    }

    $product->save();

    return new ProductResource($product, 201, 'Product Category Created Successfully');
}
```

Penjelasan Baris per Baris:

1. **public function store(Request $request)**: Mendefinisikan method **store** pada controller **ProductController**. Method ini menerima objek **Request** sebagai parameter, yang berisi data dari permintaan HTTP (biasanya dalam format JSON).
2. **$validator = \Validator::make($request->all(), [...]);**: Membuat instance validator Laravel.

- **\Validator::make(...)**: Membuat objek validator baru.
- **$request->all()**: Mengambil semua data yang dikirimkan dalam permintaan HTTP sebagai array asosiatif.
- **[...]**: Array aturan validasi (validation rules). Aturan-aturan ini lebih spesifik dibandingkan contoh sebelumnya, dan mencakup constraint seperti unique.

3. **'name' => 'required|string|max:255'**: Field **name** harus diisi (**required**), berupa string (**string**), dan tidak boleh lebih dari 255 karakter (**max:255**).
4. **'slug' => 'required|string|max:255|unique:products,slug**': Field **slug** harus diisi (**required**), berupa string (**string**), dan tidak boleh lebih dari 255 karakter (**max:255**). Selain itu, nilai **slug** harus unik dalam tabel products berdasarkan kolom slug. Ini mencegah duplikasi slug.
5. **'sku' => 'required|string|unique:products,sku'**: Field **sku** (Stock Keeping Unit) harus diisi (**required**), berupa string (**string**), dan harus unik dalam tabel **products** berdasarkan kolom **sku**. Ini memastikan bahwa setiap produk memiliki SKU yang unik.
6. **'price' => 'required|numeric|min:0'**: Field price harus diisi (**required**), berupa angka (**numeric**), dan minimal 0 (**min:0**).
7. **'stock' => 'required|integer|min:0'**: Field **stock** (stok) harus diisi (**required**), berupa integer (**integer**), dan minimal 0 (**min:0**).
8. **'product_category_id' => 'nullable|exists:product_categories,id'**: Field **product_category_id** bisa kosong (**nullable**) jika tidak diberikan dalam request. Jika diberikan, harus ada di tabel **product_categories** berdasarkan kolom id. Ini memastikan bahwa produk dikaitkan dengan kategori yang valid.
9. **'description' => 'nullable|string'**: Field **description** bisa kosong (**nullable**) jika tidak diberikan. Jika diberikan, harus berupa string (**string**).
10. **'image' => 'nullable|image|mimes:jpeg,png,jpg,gif|max:2048'**: Field **image** bisa kosong (**nullable**) jika tidak diberikan. Jika diberikan, harus berupa gambar (**image**), hanya boleh menggunakan format JPEG, PNG, JPG, atau GIF (**mimes:jpeg,png,jpg,gif**), dan maksimal 2048 KB (2MB) (**max:2048**).
11. **if ($validator->fails()) { ... }**: Memeriksa apakah validasi gagal. Jika ada aturan validasi yang tidak terpenuhi, **$validator->fails()** akan mengembalikan **true**.
    - **return response()->json($validator->errors(), 422);**: Jika validasi gagal, method ini mengembalikan respons JSON dengan kode status HTTP 422 (Unprocessable Entity), yang menunjukkan bahwa data yang dikirimkan tidak valid. **$validator->errors()** mengembalikan array yang berisi pesan kesalahan untuk setiap field yang tidak valid.
12. **$product = new Product;**: Membuat instance baru dari model **Product**. Pastikan Anda sudah memiliki model Product yang terdefinisi dengan benar di dalam aplikasi Laravel Anda.
13. **$product->name = $request->name;**: Mengisi field **name** pada model Product dengan nilai yang diterima dari request (**$request->name**).

14. **$product->slug = $request->slug;**: Mengisi field **slug** pada model **Product** dengan nilai yang diterima dari request.
15. **$product->description = $request->description;**: Mengisi field **description** pada model **Product** dengan nilai yang diterima dari request.
16. **$product->sku = $request->sku;**: Mengisi field **sku** pada model **Product** dengan nilai yang diterima dari request.
17. **$product->price = $request->price;**: Mengisi field **price** pada model **Product** dengan nilai yang diterima dari request.
18. **$product->stock = $request->stock;**: Mengisi field **stock** pada model **Product** dengan nilai yang diterima dari request.
19. **$product->product_category_id = $request->product_category_id;**: Mengisi field **product_category_id** pada model **Product** dengan nilai yang diterima dari request.
20. **$product->is_active = $request->has('is_active') ? $request->is_active : true;**: Mengatur field **is_active** (apakah produk aktif atau tidak) berdasarkan apakah parameter **is_active** ada dalam request. Jika ada, nilainya digunakan. Jika tidak, defaultnya adalah **true**.
21. **if ($request->hasFile('image')) { ... }**: Memeriksa apakah ada file yang dikirimkan dalam request dengan nama 'image'.
    - **$image = $request->file('image');**: Jika ada file, method ini mengambil objek **UploadedFile** dari permintaan.
    - **$imageName = time() . '_' . $image->getClientOriginalName();**: Membuat nama file baru yang unik menggunakan timestamp dan nama file asli. Ini untuk menghindari konflik nama file.
    - **$imagePath = $image->storeAs('uploads/products', $imageName, 'public');**: Menyimpan file ke direktori **storage/app/public/uploads/products. 'public'** adalah driver yang digunakan untuk menyimpan file di folder publik (yang dapat diakses melalui URL).
    - **$product->image_url = $imagePath;**: Mengisi field **image_url** pada model **Product** dengan path lengkap ke file yang disimpan.
22. **$product->save();**: Menyimpan data dari model **Product** ke database.
23. **return new ProductResource($product, 201, 'Product Created Successfully');**: Membuat instance dari resource (misalnya, menggunakan Resource Class) dan mengembalikannya sebagai respons JSON.
    - **new ProductResource(...)**: Membuat instance dari resource yang digunakan untuk merespons data produk. Pastikan Anda memiliki class **ProductResource** yang terdefinisi dengan benar di dalam aplikasi Laravel Anda.
    - **201**: Kode status HTTP 201 (Created) menunjukkan bahwa operasi berhasil dan sumber daya baru telah dibuat.
    - **'Product Created Successfully'**: Pesan yang akan dikembalikan sebagai bagian dari respons JSON.

Langkah 3: Menambahkan route store untuk product category dan product.

```
Route::apiResource('/product-categories', ProductCategoryController::class)->only('index','store');
Route::apiResource('/products', ProductController::class)->only('index','store');
```

Catatan:

- Tambahkan store pada apiResource ->only('index',**'store'**);

Langkah 4: UJI COBA

buka aplikasi postman, dan ketikan url dengan method **POST**

- http://127.0.0.1:8000/api/product-categories
- http://127.0.0.1:8000/api/products

Show Data berdasarkan ID menggunakan API

langkah selanjutkan kita akan membuat sebuah fungsi untuk menampilkan data berdasarkan id tertentu via API.

Langkah 1: ProductCategoryController.php

Tambahkan fungsi show pada **ProductCategoryController.php.**

```
public function show($id)
{
    $category = Categories::findOrFail($id);

    return new ProductCategoryResource($category, 200, 'Product Category Details', );
}
```

Penjelasan:

1. **public function show($id)**: Mendefinisikan method **show** pada controller **CategoryController**. Method ini menerima satu parameter **$id,** yang merupakan ID kategori produk yang ingin ditampilkan.
2. **$category = Categories::findOrFail($id);**: Ini adalah baris kunci dalam method ini.
    - **Categories::**: Mengakses model **Categories** (yang diasosiasikan dengan tabel **categories** di database).
    - **findOrFail($id)**: Metode statis dari model **Categories**. Metode ini melakukan hal berikut:
        - Mencoba menemukan record dalam tabel **categories** yang memiliki kolom **id** sama dengan **$id**.
        - Jika record ditemukan, method akan mengembalikan objek model **Category** yang sesuai.
        - Jika record *tidak* ditemukan, method akan melempar exception **ModelNotFoundException**. Exception ini kemudian ditangkap oleh Laravel dan secara otomatis dikonversi menjadi response HTTP 404 (Not Found). Ini adalah cara yang efisien untuk menangani kasus di mana ID kategori tidak valid.
3. **return new ProductCategoryResource($category, 200, 'Product Category Details',);**: Membuat instance dari resource **ProductCategoryResource** dan mengembalikannya sebagai response HTTP.
    - **new ProductCategoryResource($category, ...)**: Membuat instance baru dari resource **ProductCategoryResource**. Resource ini akan berisi data dari objek model **Category** yang ditemukan.
    - **200**: Kode status HTTP 200 (OK) menunjukkan bahwa request berhasil

diproses dan data dikembalikan dengan sukses.

- **'Product Category Details'**: Pesan yang akan dikirimkan dalam body response sebagai pesan deskriptif. Ini bisa berupa teks atau JSON.
- Tanda titik koma (;) di akhir baris penting karena menandakan bahwa semua instruksi dalam method ini harus dieksekusi secara berurutan.

Langkah 2: ProductController.php

Tambahkan fungsi show pada **ProductController.php.**

```
public function show($id)
{
    $product = Product::findOrFail($id);

    return new ProductResource($product, 200, 'Product Details');
}
```

Penjelasan:

1. **public function show($id)**: Mendefinisikan method **show** pada controller **ProductController**. Method ini menerima satu parameter **$id**, yang merupakan ID produk yang ingin ditampilkan.
2. **$product = Product::findOrFail($id);**: Ini adalah baris kunci dalam method ini.
    - **Product::**: Mengakses model **Product** (yang diasosiasikan dengan tabel **products** di database).
    - **findOrFail($id)**: Metode statis dari model **Product**. Metode ini melakukan hal berikut:
        - Mencoba menemukan record dalam tabel **products** yang memiliki kolom **id** sama dengan **$id**.
        - Jika record ditemukan, method akan mengembalikan objek model **Product** yang sesuai.
        - Jika record *tidak* ditemukan, method akan melempar exception **ModelNotFoundException**. Exception ini kemudian ditangkap oleh Laravel dan secara otomatis dikonversi menjadi response HTTP 404 (Not Found). Ini adalah cara yang efisien untuk menangani kasus di mana ID produk tidak valid.
3. **return new ProductResource($product, 200, 'Product Details');**: Membuat instance dari resource **ProductResource** dan mengembalikannya sebagai response HTTP.
    - **new ProductResource($product, ...)**: Membuat instance baru dari

resource **ProductResource**. Resource ini akan berisi data dari objek model **Product** yang ditemukan.

- **200**: Kode status HTTP 200 (OK) menunjukkan bahwa request berhasil diproses dan data dikembalikan dengan sukses.
- **'Product Details'**: Pesan yang akan dikirimkan dalam body response sebagai pesan deskriptif. Ini bisa berupa teks atau JSON.
- Tanda titik koma (;) di akhir baris penting karena menandakan bahwa semua instruksi dalam method ini harus dieksekusi secara berurutan.

Langkah 3: Menambahkan route show untuk product category dan product

```
Route::apiResource('/product-categories', ProductCategoryController::class)->only('index','store','show');
Route::apiResource('/products', ProductController::class)->only('index','store','show');
```

Catatan:

- Tambahkan store pada apiResource ->only(‘index’,’store’,**’show’**);

Update Data berdasarkan ID menggunakan API

pada langkah ini kita akan membuat sebuah fungsi untuk meng-update data berdasarkan id tertentu via API.

Langkah 1: ProductCategoryController.php

Tambahkan fungsi update pada **ProductCategoryController.php.**

```php
public function update(Request $request, $id)
{
    $validator = \Validator::make($request->all(), [
        'name' => 'required|string|max:255',
        'slug' => 'required|string|max:255',
        'description' => 'required'
    ]);

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

    $category = Categories::find($id);
    $category->name = $request->name;
    $category->slug = $request->slug;
    $category->description = $request->description;

    if ($request->hasFile('image')) {
        $image = $request->file('image');
        $imageName = time() . '_' . $image->getClientOriginalName();
        $imagePath = $image->storeAs('uploads/categories', $imageName, 'public');
        $category->image = $imagePath;
    }

    $category->save();

    return new ProductCategoryResource($category, 201, 'Product Category Updated Successfully');
}
```

Penjelasan:

1. **public function update(Request $request, $id)**: Mendefinisikan method **update** pada controller **CategoryController**. Method ini menerima dua parameter:
    - **Request $request**: Objek **Request** yang berisi data dari request HTTP (misalnya, data yang dikirimkan melalui form atau AJAX).
    - **$id**: ID kategori produk yang akan diperbarui.
2. **$validator = \Validator::make($request->all(), [...])**: Membuat instance validator Laravel.
    - **\Validator::make(...)**: Metode statis dari kelas **Validator**.
    - **$request->all()**: Mengambil semua data dari request HTTP sebagai array asosiatif.
    - **[...]**: Array asosiatif yang berisi aturan validasi untuk setiap field. Setiap

key adalah nama field, dan value adalah array aturan validasi. Dalam kasus ini:

- **'name' => 'required|string|max:255'**: Field **name** harus diisi (**required**), berupa string (**string**), dan tidak boleh lebih dari 255 karakter (**max:255**).
- **'slug' => 'required|string|max:255'**: Field **slug** harus diisi, berupa string, dan tidak boleh lebih dari 255 karakter.
- **'description' => 'required'**: Field **description** harus diisi.

3. **if ($validator->fails()) { return response()->json($validator->errors(), 422); }**: Memeriksa apakah validator gagal (yaitu, ada field yang tidak memenuhi aturan validasi).
    - **$validator->fails()**: Metode dari instance validator yang mengembalikan **true** jika ada kesalahan validasi, dan **false** jika semua field valid.
    - **response()->json($validator->errors(), 422)**: Membuat response JSON dengan kode status HTTP 422 (Unprocessable Entity). Response ini berisi array kesalahan validasi dari **$validator->errors()**.
4. **$category = Categories::find($id);**: Mencari record kategori di tabel **categories** berdasarkan ID yang diberikan (**$id**). Metode **find()** mengembalikan objek model **Category** jika ditemukan, atau melempar exception **ModelNotFoundException** jika tidak ditemukan.
5. **$category->name = $request->name;**: Mengatur nilai field **name** dari object kategori ke nilai yang dikirimkan melalui request (**$request->name**).
6. **$category->slug = $request->slug;**: Mengatur nilai field **slug** dari object kategori ke nilai yang dikirimkan melalui request (**$request->slug**).
7. **$category->description = $request->description;**: Mengatur nilai field **description** dari object kategori ke nilai yang dikirimkan melalui request (**$request->description**).
8. **if ($request->hasFile('image')) { ... }**: Memeriksa apakah ada file gambar yang diunggah melalui request.
    - **$request->hasFile('image')**: Metode dari objek **Request** yang mengembalikan true jika ada file dengan nama 'image' yang diunggah, dan **false** jika tidak.
9. **$image = $request->file('image');**: Jika ada file gambar yang diunggah, variabel **$image** akan berisi objek **UploadedFile**.
10. **$imageName = time() . '_' . $image->getClientOriginalName();**: Membuat nama file baru dengan menggabungkan timestamp saat ini dan nama file asli. Ini untuk menghindari konflik nama file jika ada beberapa file dengan nama yang sama.
11. **$imagePath = $image->storeAs('uploads/categories', $imageName, 'public');**: Menyimpan file gambar ke dalam direktori **uploads/categories** di storage Laravel.
    - **$image->storeAs(...)**: Metode dari objek **UploadedFile** yang

menyimpan file ke lokasi tertentu.
    - **'uploads/categories'**: Direktori tempat file akan disimpan.
    - **$imageName**: Nama file baru yang akan digunakan.
    - **'public'**: Menentukan bahwa file harus disimpan dalam direktori public storage (yang dapat diakses melalui URL).
12. **$category->image = $imagePath;**: Mengatur nilai field **image** dari object kategori ke path file gambar yang baru disimpan (**$imagePath**).
13. **$category->save();**: Menyimpan perubahan pada object kategori ke database.
14. **return new ProductCategoryResource($category, 201, 'Product Category Updated Successfully');**: Membuat instance dari resource **ProductCategoryResource** dan mengembalikannya sebagai response HTTP.
    - **new ProductCategoryResource(...)**: Membuat instance baru dari resource.
        - **$category**: Objek model **Category** yang telah diperbarui.
        - **201**: Kode status HTTP 201 (Created) menunjukkan bahwa request berhasil dan sumber daya baru telah dibuat (dalam kasus ini, kategori produk telah diperbarui).
        - **'Product Category Updated Successfully'**: Pesan deskriptif yang akan dikirimkan dalam body response.

Langkah 2: ProductController.php

Tambahkan fungsi update pada **ProductController.php.**

```
public function update(Request $request, $id)
{
    $product = Product::findOrFail($id);

    $validator = \Validator::make($request->all(), [
        'name' => 'required|string|max:255',
        'slug' => 'required|string|max:255|unique:products,slug,' . $product->id,
        'description' => 'nullable|string',
        'sku' => 'required|string|unique:products,sku,' . $product->id,
        'price' => 'required|numeric|min:0',
        'stock' => 'required|integer|min:0',
        'product_category_id' => 'nullable|exists:product_categories,id',
        'image' => 'nullable|image|mimes:jpeg,png,jpg,gif|max:2048',
        'is_active' => 'boolean',
    ]);

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

    $product->name = $request->name;
    $product->slug = $request->slug;
    $product->description = $request->description;
    $product->sku = $request->sku;
    $product->product_category_id = $request->product_category_id;
    $product->is_active = $request->has('is_active') ? $request->is_active : true;

    if ($request->hasFile('image')) {
        $image = $request->file('image');
        $imageName = time() . '_' . $image->getClientOriginalName();
        $imagePath = $image->storeAs('uploads/products', $imageName, 'public');
        $product->image_url = $imagePath;
    }

    $product->save();

    return new ProductResource($product, 201, 'Product Updated Successfully');
}
```

Langkah 3: Menambahkan route update untuk product category dan product

```
Route::apiResource('/product-categories', ProductCategoryController::class)->only('index','store','show','update');
Route::apiResource('/products', ProductController::class)->only('index','store','show','update');
```

Catatan:

- Tambahkan store pada apiResource ->only(‘index’,’store’,’show’,**’update’**);

Delete Data berdasarkan ID menggunakan API

pada langkah ini kita akan membuat sebuah fungsi untuk men-delete data berdasarkan id tertentu via API.

Langkah 1: ProductCategoryController.php

Tambahkan fungsi update pada **ProductCategoryController.php.**

```
public function destroy($id)
{
    $category = Categories::findOrFail($id);
    $category->delete();

    return response()->json(['success' => true, 'message' => 'Product Category Deleted Successfully']);
}
```

Langkah 2: ProductController.php

Tambahkan fungsi update pada **ProductController.php.**

```
public function destroy($id)
{
    $product = Product::findOrFail($id);
    $product->delete();

    return response()->json(['success' => true, 'message' => 'Product Deleted Successfully']);
}
```

Langkah 3: Menambahkan route delete untuk product category dan product

```
Route::apiResource('/product-categories', ProductCategoryController::class)->only('index','store','show','update','destroy');
Route::apiResource('/products', ProductController::class)->only('index','store','show','update','destroy');
```

Catatan:

- Tambahkan store pada apiResource ->only(‘index’,’store’,’show’,’update’,’destroy’);

Mengakses API dengan Guzzle di Laravel

Guzzle adalah library PHP yang sangat populer untuk melakukan permintaan HTTP. Di Laravel, Guzzle sudah terintegrasi secara default, sehingga memudahkan kita untuk berinteraksi dengan API eksternal. Tutorial ini akan memandu Anda melalui langkah-langkah dasar dan beberapa praktik terbaik.

Langkah 1: Membuat Request ke API

Ada beberapa cara untuk membuat request ke API di Laravel. Berikut adalah dua metode utama:

- Menggunakan **get()** method: Ini adalah cara yang paling sederhana untuk melakukan permintaan GET.
- Menggunakan **request()** method: Metode ini lebih fleksibel dan memungkinkan Anda untuk mengkonfigurasi berbagai opsi seperti headers, body, dan method (GET, POST, PUT, DELETE).

Contoh 1: Menggunakan get() Method

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\Http; // Import facade Http untuk menggunakan Guzzle

class ApiController extends Controller
{
    public function getApiData()
    {
        try {
            $response = Http::get('https://jsonplaceholder.typicode.com/posts/1');

            // Memeriksa apakah request berhasil
            if ($response->successful()) {
                $data = $response->json(); // Mengubah response menjadi format JSON
                return view('api_view', ['data' => $data]); // Kirim data ke view
            } else {
                // Menangani error jika request gagal
                return 'Terjadi kesalahan saat mengambil data dari API.';
            }

        } catch (\Exception $e) {
            // Menangani exception yang mungkin terjadi
            return 'Terjadi kesalahan: ' . $e->getMessage();
        }
    }
}
```

Penjelasan:

- **use Illuminate\Support\Facades\Http;**: Mengimpor facade **Http** untuk menggunakan Guzzle. Facade ini menyediakan cara mudah untuk mengakses kelas-kelas Guzzle tanpa perlu menginstansiasinya.
- **Http::get('https://jsonplaceholder.typicode.com/posts/1');**: Membuat request GET ke URL yang ditentukan.
- **$response->successful()**: Memeriksa apakah request berhasil (status code 200-299).
- **$response->json()**: Mengubah response dari Guzzle menjadi array PHP yang sesuai dengan format JSON.
- **return view('api_view', ['data' => $data]);**: Mengirim data ke view bernama **api_view**.

Contoh 2: Menggunakan request() Method (Lebih Fleksibel)

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\Http;

class ApiController extends Controller
{
    public function postApiData()
    {
        try {
            $response = Http::post('https://jsonplaceholder.typicode.com/posts', [
                'title' => 'Judul Postingan Baru',
                'body' => 'Isi postingan baru',
                'userId' => 1,
            ]);

            if ($response->successful()) {
                $data = $response->json();
                return view('api_view', ['data' => $data]);
            } else {
                return 'Terjadi kesalahan saat mengirim data ke API.';
            }

        } catch (\Exception $e) {
            return 'Terjadi kesalahan: ' . $e->getMessage();
        }
    }
}
```

Penjelasan:

- **Http::post('https://jsonplaceholder.typicode.com/posts', ...);**: Membuat request POST ke URL yang ditentukan dengan data yang dikirimkan dalam array asosiatif.
- Data yang dikirimkan menggunakan **['title' => '...', 'body' => '...']**. Ini adalah format umum untuk mengirim data JSON melalui Guzzle.

Langkah 2: Menangani Response dan Error

Penting untuk menangani response dari API dengan benar, termasuk memeriksa status code dan mengurai data JSON. Selain itu, Anda juga harus menangani error yang mungkin terjadi selama proses request.

- Status Code: Gunakan **$response->successful()** untuk memeriksa apakah request berhasil (status code 200-299).
- Mengurai Data JSON: Gunakan **$response->json()** untuk mengonversi response menjadi array PHP.
- Penanganan Error: Gunakan blok **try...catch** untuk menangkap exception yang mungkin terjadi selama proses request. Ini akan membantu Anda mencegah aplikasi Anda crash dan memberikan pesan error yang lebih informatif kepada pengguna.

Langkah 3: Menggunakan Headers (Opsional)

Terkadang, API memerlukan header tertentu untuk diatur. Anda dapat melakukannya dengan menggunakan opsi **headers** pada metode **get()** atau **request()**.

```
$response = Http::get('https://example.com/api', [
    'headers' => [
        'Authorization' => 'Bearer YOUR_API_TOKEN',
        'Content-Type' => 'application/json',
    ],
]);
```