API dan RESTful API

Tujuan Pembelajaran

- Memahami konsep dasar API dan manfaatnya
- Menjelaskan arsitektur RESTful API
- Mengimplementasikan API endpoints menggunakan Laravel
- Melakukan testing dan debugging API
- Menerapkan best practices dalam pengembangan API

API

API (Application Programming Interface) merupakan sekumpulan aturan dan protokol yang memungkinkan aplikasi berbeda platform untuk berkomunikasi dan saling terintegrasi.

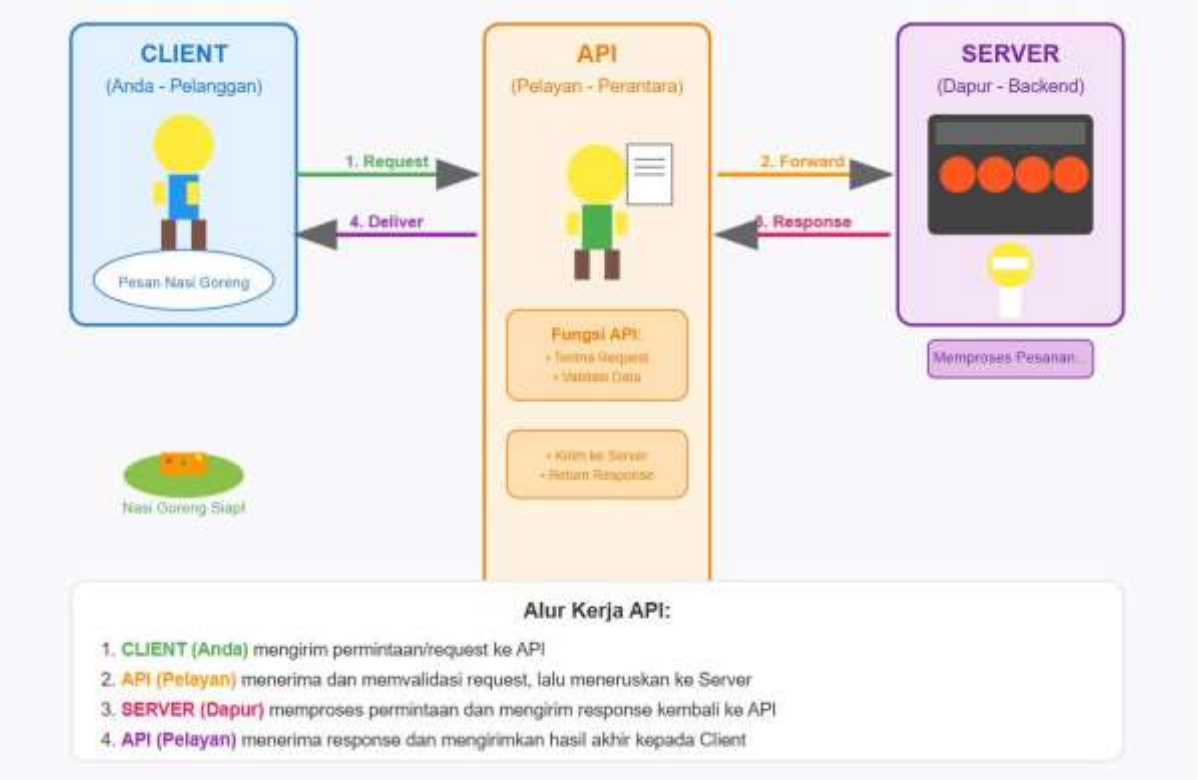
Mengapa API Penting?

API (Application Programming Interface) adalah media untuk komunikasi antara aplikasi yang berbeda. Dalam era digital saat ini, hampir semua aplikasi modern menggunakan API untuk:

- Integrasi dengan layanan pihak ketiga
- Memisahkan frontend dan backend
- Mendukung multiple platform (web, mobile, desktop)
- Memungkinkan microservices architecture

Konsep Dasar API

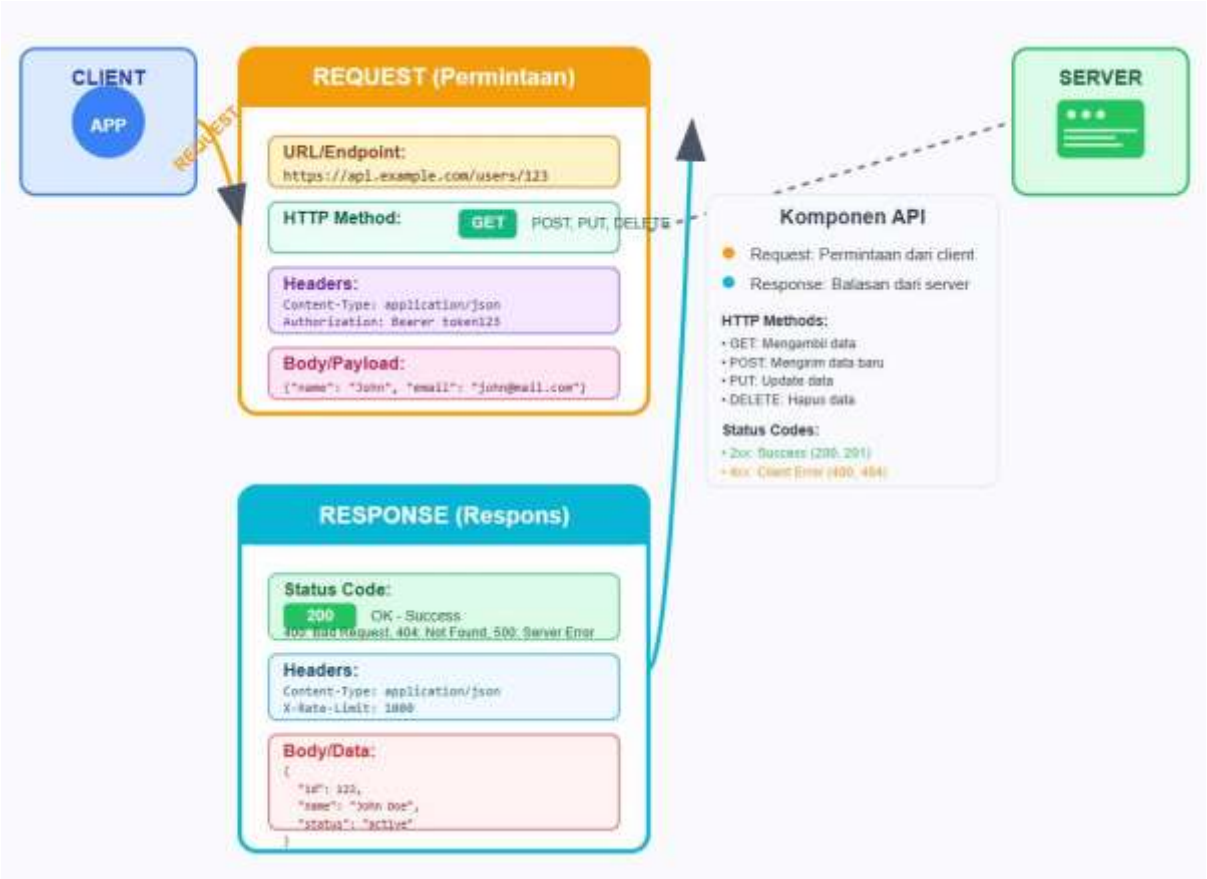
Agar lebih memahami konsep dasar API berikut analogi API pemesanan makanan pada restoran.



Komponen Utama API (Request dan Response)

1. Request (Permintaan)
    - URL/Endpoint
    - HTTP Method
    - Headers
    - Body/Payload
2. Response (Respons)
    - Status Code
    - Headers
    - Body/Data

Berikut ini merupakan ilustrasi komponen utama API



Jenis-Jenis API

- Web API : Menggunakan protokol HTTP/HTTPS
- REST API: Mengikuti arsitektur REST
- GraphQL API: Query language untuk API
- SOAP API: Protocol berbasis XML
- Library API: Interface untuk library atau framework
- Operating System API: Interface dengan sistem operasi
- Database API: Interface untuk mengakses database

RESTful API

REST (Representational State Transfer) adalah arsitektur untuk merancang web services. REST bukan protokol atau standar, melainkan seperangkat prinsip desain.

Prinsip REST (Representational State Transfer)

1. **Client-Server Architecture**
   Client dan server terpisah dan dapat berkembang secara independen.
2. **Stateless**
   Setiap request harus berisi semua informasi yang diperlukan server untuk memproses request tersebut.
3. **Cacheable**
   Response harus dapat di-cache untuk meningkatkan performa.
4. **Uniform Interface**
   Interface yang konsisten antara client dan server.
5. **Layered System**
   Arsitektur berlapis yang memungkinkan scalability.
6. **Code on Demand (Opsional)**
   Server dapat mengirim kode executable ke client

Struktur URL REST

```
GET    /api/users          # Mendapatkan semua users
GET    /api/users/1        # Mendapatkan user dengan ID 1 POST    /api/users
# Membuat user baru
PUT    /api/users/1        # Update user dengan ID 1 DELETE /api/users/1
# Hapus user dengan ID 1
```

HTTP Methods dalam REST

GET - Mengambil Data

```
GET /api/users GET
/api/users/1
```

Keterangan :

- Digunakan untuk retrieve data
- Idempotent (hasil sama meski dipanggil berulang)
- Tidak mengubah state server

POST - Membuat Data Baru

```
POST /api/users
Content-Type: application/json

{
   "name": "John Doe",
   "email": "john@example.com" }
```

Keterangan :

- Digunakan untuk membuat resource baru
- Tidak idempotent

PUT - Update Seluruh Resource

```
PUT /api/users/1 Content-Type:
application/json

{
   "name": "John Smith",
   "email": "johnsmith@example.com" }
```

Keterangan :

- Update seluruh resource
- Idempotent

PATCH - Update Sebagian Resource

```
PATCH /api/users/1 Content-Type:
application/json

{
   "name": "John Smith"
}
```

Keterangan :

- Update sebagian field
- Idempotent

DELETE - Menghapus Resource

```
DELETE /api/users/1
```

Keterangan :

- Menghapus resource
- Idempotent

HTTP Status Codes

2xx Success

- **200 OK**: Request berhasil
- **201 Created**: Resource berhasil dibuat
- **204 No Content**: Request berhasil, tidak ada content

4xx Client Error

- **400 Bad Request**: Request tidak valid
- **401 Unauthorized**: Autentikasi diperlukan

- **403 Forbidden**: Akses ditolak
- **404 Not Found**: Resource tidak ditemukan
- **422 Unprocessable Entity**: Validation error

5xx Server Error

- **500 Internal Server Error**: Error di server
- **502 Bad Gateway**: Gateway error
- **503 Service Unavailable**: Service tidak tersedia

Membuat API dengan Laravel

Setup Project

```
composer create-project laravel/laravel api-project
```

```
cd api-project\
```

```
php artisan serve
```

Database dan Migration

```
#  Membuat migration
php artisan make:migration create_products_table

#  migration
Schema::create('products', function (Blueprint $table) {      $table->id();
    $table->string('name');
    $table->text('description');
    $table->decimal('price', 10, 2);
    $table->integer('stock');
    $table->timestamps();
});

#  Jalankan migration php
artisan migrate
```

Model

php artisan make:model Product

```
// app/Models/Product.php
class Product extends Model
{
    protected $fillable = [
        'name', 'description', 'price', 'stock'      ];

    protected $casts = [
        'price' => 'decimal:2'      ];
}
```

API Routes

```
// routes/api.php
Route::apiResource('products', ProductController::class);

// manual:
Route::get('products', [ProductController::class, 'index']); Route::post('products', [ProductController::class,
'store']); Route::get('products/{product}', [ProductController::class, 'show']);
Route::put('products/{product}', [ProductController::class, 'update']); Route::delete('products/{product}',
[ProductController::class, 'destroy']);
```

API Controller

```
php artisan make:controller ProductController --api
```

```
class ProductController extends Controller {
    public function index()
    {
        $products = Product::all();
        return response()->json([
            'status' => 'success',
            'data' => $products
        ]);
    }

    public function store(Request $request)
    {
        $validated = $request->validate([
            'name' => 'required|string|max:255',            'description' =>
'required|string',            'price' => 'required|numeric|min:0',
'stock' => 'required|integer|min:0'         ]);

                $product = Product::create($validated);

        return response()->json([
            'status' => 'success',
            'message' => 'Product created successfully',            'data' => $product
        ], 201);
    }

    public function show(Product $product)     {
        return response()->json([
            'status' => 'success',
            'data' => $product
        ]);
    }

         public function update(Request $request, Product $product)
    {
        $validated = $request->validate([
            'name' => 'sometimes|string|max:255',            'description' =>
'sometimes|string',            'price' => 'sometimes|numeric|min:0',
'stock' => 'sometimes|integer|min:0'         ]);
```

```
            $product->update($validated);

        return response()->json([
            'status' => 'success',
            'message' => 'Product updated successfully',            'data' => $product
        ]);
    }

    public function destroy(Product $product)     {
        $product->delete();

        return response()->json([
            'status' => 'success',
            'message' => 'Product deleted successfully'         ]);
    }
}
```

API Resources (Data Transformation)

Fitur yang memungkinkan untuk mentransformasi model data atau collection menjadi format JSON yang konsisten dan mudah dikustomisasi untuk API response. API Resource berfungsi sebagai layer transformasi antara model Eloquent dan JSON response yang dikirim ke client sehingga dapat digunakan untuk Mengontrol format output JSON, Menyembunyikan field sensitive, Menambahkan field computed dan Membuat response yang konsisten.

Membuat Resource

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

```
// app/Http/Resources/ProductResource.php
class ProductResource extends JsonResource
{
    public function toArray($request)
    {
        return [
            'id' => $this->id,
            'name' => $this->name,
            'description' => $this->description,
            'price' => $this->price,
            'stock' => $this->stock,
            'created_at' => $this->created_at->format('Y-m-d H:i:s'),            'updated_at' => $this-
>updated_at->format('Y-m-d H:i:s')         ];
    }
}
```

Menggunakan Resource di Controller

Silahkan ubah method index dan show menggunakan API Resource

```
public function index()
{
    $products = Product::all();
    return ProductResource::collection($products); }

public function show(Product $product)
```

```
{
    return new ProductResource($product); }
```

Validasi dan Error Handling

Form Request Validation

```
php artisan make:request StoreProductRequest
```

```
// app/Http/Requests/StoreProductRequest.php class StoreProductRequest extends FormRequest {
    public function authorize()
    {
        return true;
    }

    public function rules()
    {
        return [
            'name' => 'required|string|max:255',            'description' => 'required|string',            'price' =>
'required|numeric|min:0',            'stock' => 'required|integer|min:0'        ];
    }

    public function messages()
    {
        return [
            'name.required' => 'Nama produk wajib diisi',            'price.min' => 'Harga tidak boleh negatif'
        ];
    }

}
```

Implementasikan pada method **store**

Global Exception Handler

Tambahkan kode program berikut pada ***app/Exceptions/Handler.php*** untuk menanggani Exception.

```
public function render($request, Throwable $exception) {
    if ($request->wantsJson()) {
        if ($exception instanceof ValidationException) {            return response()-
>json([
                'status' => 'error',
                'message' => 'Validation failed',
                'errors' => $exception->errors()
            ], 422);
        }

        if ($exception instanceof ModelNotFoundException) {            return
response()->json([
                'status' => 'error',
                'message' => 'Resource not found'
            ], 404);
        }
    }
        return parent::render($request, $exception); }
```

Mengakses API Products dengan Postman

Untuk memastikan API yang telah dibuat berjalan dengan baik maka perlu dilakukan percobaan mengakses API tersebut, untuk testing menggunakan POSTMAN, silahkan download dan install POSTMAN pada computer dan lakukan testing API yang telah dibuat, sebelum melakukan testing silahkan tonton video tutorial cara penggunaan POSTMAN pada link berikut :

https://www.youtube.com/watch?v=7cJy1pFubAc atau https://www.youtube.com/watch?v=VcYuIsKlOfg

1. **GET - Mengambil Semua Products**

Method: GET
URL: http://localhost:8000/products

Headers:

Accept: application/json Content-

Type: application/json **Response**

Example (200 OK):

```
{
   "data": [
      {
         "id": 1,
         "name": "Laptop Gaming",
         "description": "Laptop gaming dengan spek tinggi",          "price":
"15000000.00",
         "stock": 10,
         "created_at": "2024-01-15T10:30:00.000000Z",
         "updated_at": "2024-01-15T10:30:00.000000Z"
      },
      {
         "id": 2,
         "name": "Mouse Wireless",
         "description": "Mouse wireless ergonomis",          "price":
"250000.00",
         "stock": 50,
         "created_at": "2024-01-15T11:00:00.000000Z",          "updated_at":
"2024-01-15T11:00:00.000000Z"       }
   ]
}
```

2. **POST - Membuat Product Baru**

Method: POST
URL: http://localhost:8000/products

Headers:

Accept: application/json Content-

Type: application/json **Body**

(JSON):

```
{
   "name": "Smartphone Android",
```

```
    "description": "Smartphone dengan kamera 108MP dan RAM 8GB",     "price":
4500000.00,
    "stock": 25
}
```


Response Example (201 Created):


```
{
    "message": "Product created successfully",
    "data": {
        "id": 3,
        "name": "Smartphone Android",
        "description": "Smartphone dengan kamera 108MP dan RAM 8GB",         "price":
"4500000.00",
        "stock": 25,
        "created_at": "2024-01-15T12:00:00.000000Z",
        "updated_at": "2024-01-15T12:00:00.000000Z"
    }
}
```


Validation Error Example (422 Unprocessable Entity):


```
{
    "message": "The given data was invalid.",
    "errors": {
        "name": ["The name field is required."],         "price": ["The price
field is required."]     }
}
```

3. **GET - Mengambil Product Berdasarkan ID**

Method: GET
URL: http://localhost:8000/products/{id}
Contoh: http://localhost:8000/products/1

Headers:

Accept: application/json Content-

Type: application/json **Response**

Example (200 OK):


```
{
   "data": {
      "id": 1,
      "name": "Laptop Gaming",
      "description": "Laptop gaming dengan spek tinggi",        "price":
"15000000.00",
      "stock": 10,
      "created_at": "2024-01-15T10:30:00.000000Z",
      "updated_at": "2024-01-15T10:30:00.000000Z"
   }
}
```


Not Found Example (404 Not Found):


```
{
   "message": "Product not found" }
```


---

4. **PUT - Update Product**

Method: PUT
URL: http://localhost:8000/products/{id}
Contoh: http://localhost:8000/products/1

Headers:

Accept: application/json Content-

Type: application/json **Body**

(JSON):


```
{
   "name": "Laptop Gaming Updated",
   "description": "Laptop gaming dengan spek tinggi dan SSD 1TB",     "price": 16500000.00,
   "stock": 8
}
```


Response Example (200 OK):


```
{
   "message": "Product updated successfully",
   "data": {
      "id": 1,
      "name": "Laptop Gaming Updated",
```

"description": "Laptop gaming dengan spek tinggi dan SSD 1TB",
"price": "16500000.00",
"stock": 8,
"created_at": "2024-01-15T10:30:00.000000Z", "updated_at": "2024-01-15T13:15:00.000000Z" }
}

5. **DELETE - Hapus Product**

Method: DELETE
URL: http://localhost:8000/products/{id}
Contoh: http://localhost:8000/products/1

Headers:

Accept: application/json Content-

Type: application/json **Response**

Example (200 OK):

{
"message": "Product deleted successfully" }

Not Found Example (404 Not Found):

{
"message": "Product not found" }