REST API Design

REST API

- **REST**: Representational State Transfer
- Arsitektur berbasis standar web (HTTP)
- Memfasilitasi komunikasi antara client dan server
- Menggunakan metode HTTP (GET, POST, PUT, DELETE)

Konsep Dasar REST API

- **Resource**: Entitas atau data yang dapat diakses melalui API.
- **Endpoints**: URL khusus untuk mengakses resource.
- **HTTP Methods**: Mendefinisikan operasi yang dilakukan pada resource.

Komponen REST API: Endpoints

- **Endpoint**: URL unik yang merepresentasikan objek atau aksi.
- Contoh: `/users`, `/products/{id}`
- Mengidentifikasi resource yang spesifik.

Komponen REST API: Metode HTTP

- `GET` : Mengambil data.
- `POST` : Membuat data baru.
- `PUT/PATCH` : Memperbarui data yang ada.
- `DELETE` : Menghapus data.

Representasi Resource, URL, dan Metode

Resource	Metode HTTP	URL
Daftar Produk	`GET`	`/api/products`
Detail Produk	`GET`	`/api/products/123`
Tambah Produk	`POST`	`/api/products`
Perbarui Produk	`PUT` atau `PATCH`	`/api/products/123`
Hapus Produk	`DELETE`	`/api/products/123`

Standar dan Prinsip REST API

Statelessness - Independensi Permintaan HTTP

- Setiap permintaan HTTP dianggap terpisah dan lengkap.
- Server tidak menyimpan informasi tentang status client setelah permintaan selesai.
- Manfaat: Memudahkan skalabilitas server karena tidak perlu menyinkronkan data atau status antar sesi.

Contoh:

- Ketika sebuah aplikasi mengirim permintaan untuk mengupdate data pengguna, permintaan tersebut harus mengandung semua informasi yang diperlukan untuk proses tersebut, seperti ID pengguna dan data baru.

Standar dan Prinsip REST API

Cacheability - Kemampuan Data untuk Di-cache

- REST API dirancang untuk mendukung cache respon di sisi client.
- Server harus menentukan secara eksplisit data mana yang bisa di-cache dan berapa lama.
- Manfaat: Mengurangi jumlah permintaan ke server, mempercepat waktu respons, dan mengurangi beban pada server.

Contoh:

- Respon untuk permintaan GET pada data produk dapat di-cache. Jika data produk tidak sering berubah, respon ini dapat disimpan di cache client untuk penggunaan selanjutnya tanpa perlu melakukan permintaan berulang.

Standar dan Prinsip REST API

Layered System - Sistem Berlapis

- Arsitektur REST memungkinkan penggunaan berbagai lapisan antara client dan server.
- Lapisan ini dapat mencakup sistem keamanan, load balancing, cache, dsb.
- Manfaat: Menambahkan fleksibilitas dan keamanan. Lapisan tambahan ini tidak memengaruhi permintaan dan respon antara client dan server.

Contoh:

- Dalam aplikasi e-commerce, lapisan keamanan dapat digunakan untuk autentikasi, sementara lapisan lain bisa bertindak sebagai cache terdistribusi untuk meningkatkan kinerja.

Best Practice

Konvensi Penamaan Resource

- Gunakan kata benda dan bentuk jamak jika representasi resource adalah kumpulan objek.
- Gunakan kata benda dan bentuk tunggal jika representasi resource adalah objek tunggal.
- Gunakan kata kerja jika representasi resource adalah aksi.
- Jelas dan konsisten.

Penanganan Kesalahan

- Gunakan kode status HTTP yang tepat.
- Sediakan pesan kesalahan yang informatif.

Best Practice

Keamanan

- Autentikasi dan otorisasi.
- Validasi input dan sanitasi data.

Versioning

- Kelola perubahan API.
- Contoh: `/api/v1/products`

Tools untuk Pengembangan REST API

Frameworks dan Libraries

- Express.js, Spring Boot, Django REST framework, Laravel, dsb.
- Mempercepat pembuatan dan pengelolaan API.

Tools Dokumentasi

- Swagger, Postman.
- Dokumentasi yang jelas dan interaktif.

Studi Kasus

API untuk aplikasi e-commerce.

Skenario: Aplikasi e-commerce yang memungkinkan pengguna untuk membeli produk yang tersedia, menambahkannya ke keranjang belanja, dan melakukan checkout.

Persyaratan

- Menampilkan data produk, keranjang belanja, dan pesanan.
- Menambahkan/Menghapus produk ke keranjang belanja.
- Checkout dan membuat pesanan baru.

Studi Kasus

Proses Desain

1. **Identifikasi Entitas**: Produk, Keranjang Belanja, Pesanan.
2. **Definisi Endpoints**:
    - `/products`
    - `/shopping-cart`
    - `/orders`
3. **Tentukan Metode HTTP**:
    - `GET` untuk membaca data.
    - `POST` untuk membuat entitas baru.
    - `PUT/PATCH` untuk memperbarui.
    - `DELETE` untuk menghapus.

Tips Implementasi

- Pilih framework yang sesuai (misalnya, Express.js untuk Node.js).
- Fokus pada desain yang modular dan reusable.
- Gunakan tools seperti Postman untuk testing.