



RESTful API

Gold - Chapter 5 - Topic 2

**Selamat datang di Chapter 5 Topik 2 online
course Fullstack Web dari Binar Academy!**





Halo, jumpa lagi 🙌

Di topik sebelumnya kita bahas tentang Express.js yang kita pakai untuk membuat HTTP Server.

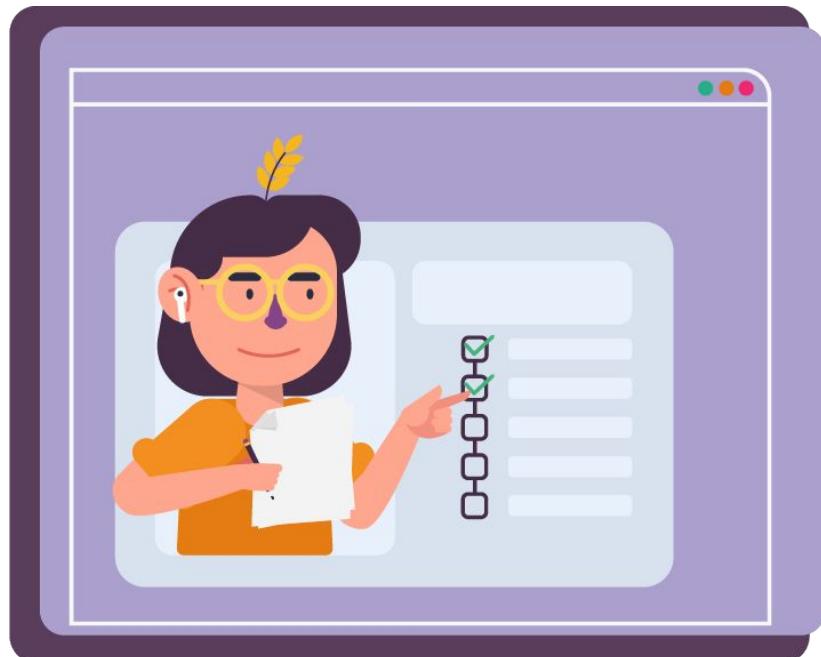
Ketika kita bahas tentang HTTP Server dan beberapa endpoint yang ada disana. Munculah pertanyaan gimana cara membuat *HTTP Server dengan baik dan benar*.

Untuk menjawab pertanyaan tersebut, di topik ini kita akan bahas tentang kaidah pembuatan HTTP Server yang biasa disebut sebagai **RESTful API**.



Detailnya, kita bakal bahas hal-hal berikut:

- Mengenal desain API
- Pengenalan konsep REST API dan RESTful API
- Resource identifiers
- Resource methods
- Penerapan REST API pada Express.Js



Eits, sebelum kita nyemplung ke REST API, ada baiknya kita ingat-ingat lagi dulu pemahaman terkait **Client-Server Architecture**

Ada yang masih ingat? hehehe



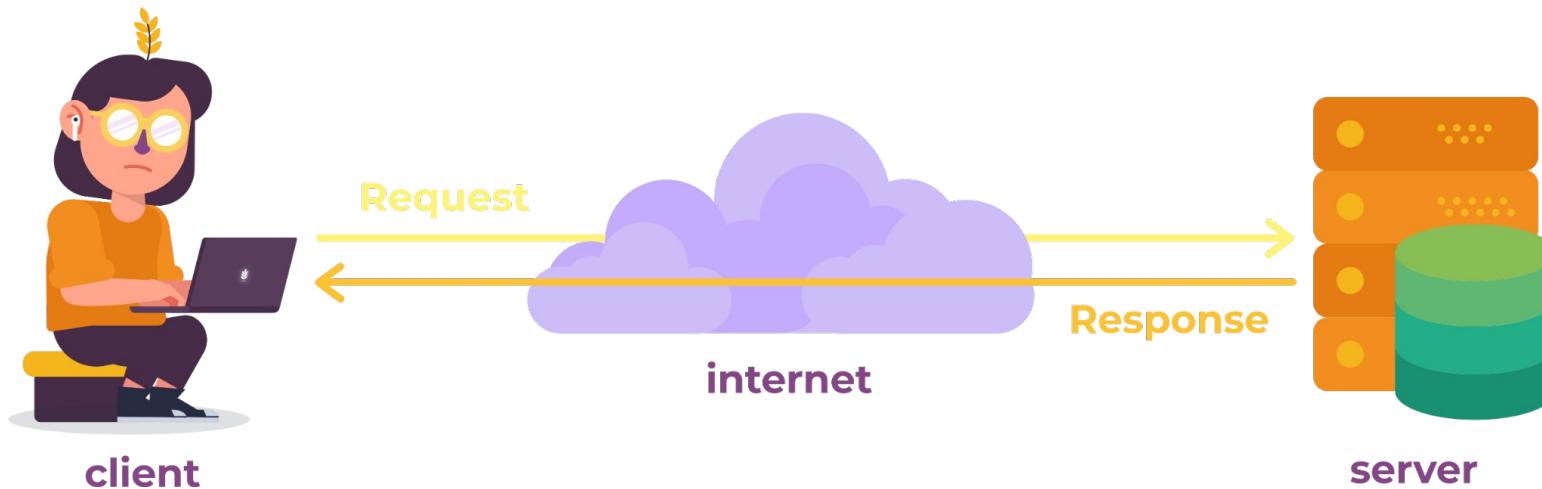
Client-Server Architecture



Client, dan **server** tuh dua komponen yang bisa banget dipisahin loh dari sebuah website yang terhubung melalui internet! Pasti kalian bingung, kok bisa ya?

Di chapter sebelumnya memang sudah dibahas, kalo pengembangan web, hampir semua aplikasi tidak memisahkan antara client dan server-nya.

Tapi, seiring berkembangnya web dan makin kompleks kode yang dikembangkan, maka kita perlu ada pemisahan antara fungsi client dan server.





ALL YOU CAN EAT

Client-Server Architecture

Disinilah konsep Client-Server Architecture muncul. Dimana inti dari arsitektur ini adalah **memisahkan kode untuk client dan kode untuk server.**

Client-server architecture tuh konsepnya mirip kayak makan di AYCE (All You Can Eat). Pihak restoran menyediakan alat masak, meja, dan bahan-bahan makanan mentah. Dan konsumen sendiri yang nantinya bisa ambil dan masak sesuai selera.



Dari analogi tadi, kita bisa simpulkan sesuatu dari konsep Client-Server Architecture, bahwa :

- **Client (Konsumen)** adalah sebuah aplikasi independen yang berjalan tanpa perlu bantuan server untuk mengolah data dan menampilkannya. Tugas client disini hanyalah mengolah data tersebut menjadi tampilan yang bisa dikonsumsi oleh user
- **Server (Pihak Restoran)** adalah sebuah aplikasi yang dijalankan di dalam server yang berfungsi untuk menyediakan data-data (makanan mentah) yang diperlukan oleh client agar dapat diolah menjadi sebuah user interface. Server juga bertanggung jawab untuk mengurus hal-hal terkait logika bisnis dari sebuah website.



URL dari kedua aplikasi tersebut akan berbeda karena client dan server ini dua aplikasi yang berbeda. Sebagai contoh, kita punya aplikasi bernama Instagram, nah URL dari aplikasi tersebut akan dijabarkan sebagai berikut:

- Client : <https://instagram.com>
- Server : <https://api.instagram.com>

Si aplikasi server ini hanya akan diakses oleh aplikasi client, user kita hanya akan mengakses aplikasi client saja. Jadi user ga perlu tau menau tentang cara pake <https://api.instagram.com>



Hmm... Terus kalo client dan servernya terpisah, terus mereka tukeran data gimana ya?

Pertanyaan bagus!

Nah, dari sinilah muncul konsep kaidah dalam **mendesain API**. Kira-kira ada apa aja ya?

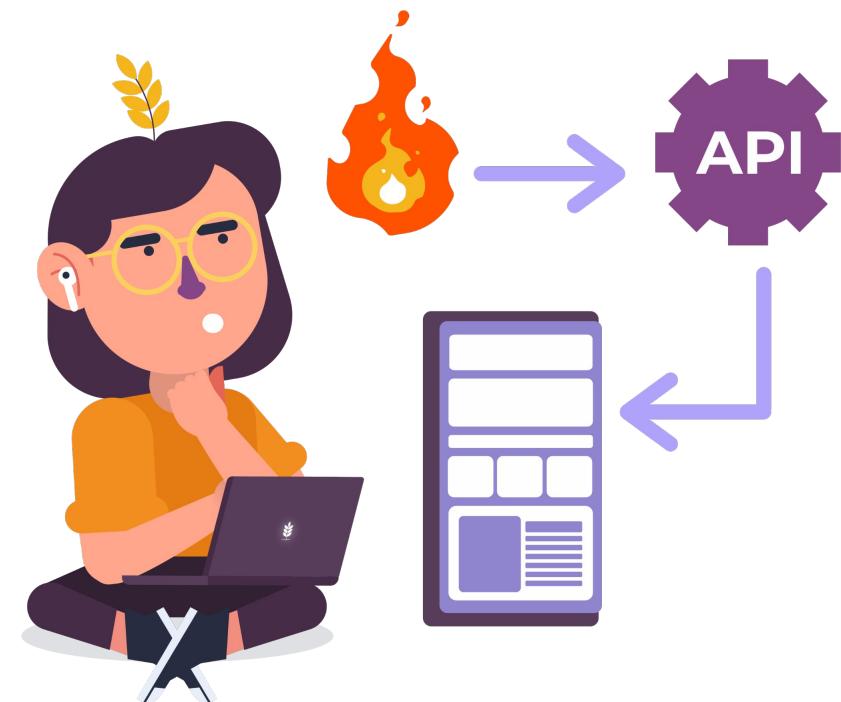


Desain API

Bentar-bentar, sebelum kita bahas tentang kaidah cara mendesain sebuah API, kita perlu bahas dulu apa itu desain API.

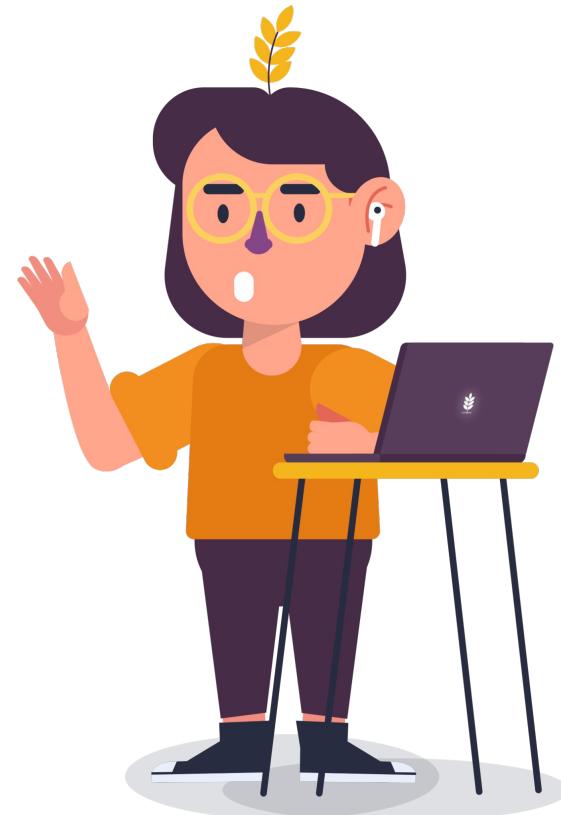
Jadi, **desain API** itu adalah sebuah rancangan tentang operasi apa saja yang bisa dilakukan pada sebuah aplikasi dengan menggunakan fungsi-fungsi tertentu, dan data yang akan dikirim ke dalam aplikasi tersebut akan berbentuk seperti apa dan sebagainya.

Terus apa aja sih komponen-komponen dalam desain API? 😕



Biasanya yang tercakup dalam Desain API tuh kayak gini :

- **Nama-nama fungsi (Endpoint)** dari sebuah aplikasi tersebut yang dapat diakses oleh aplikasi lain (HTTP Server)
- **Format data** apa yang digunakan untuk bertukar data, sebagai contoh, JSON atau XML
- **Bentuk data** yang dikirim akan seperti apa.



Kaidah dalam Mendesain API

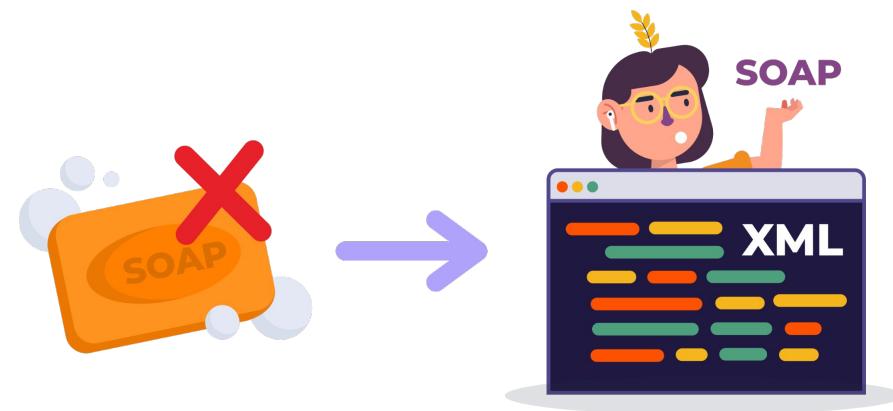
Desain itu luas, teman-teman, maka dari itu ada beberapa kaidah yang bisa kita jadikan pedoman dalam mendesain sebuah API, yaitu:

- **SOAP API**
- **REST API**



Simple Objects Access Protocol (SOAP)

Ini bukan sabun, tapi sebuah kaidah dalam mendesain API. Di dalam SOAP API, antara client dan server akan bertukar data dengan format **XML (Extensible Markup Language)** mirip kayak HTML. Tapi, karena SOAP API ini sudah usang, maka kita gak akan banyak bahas disini ya.



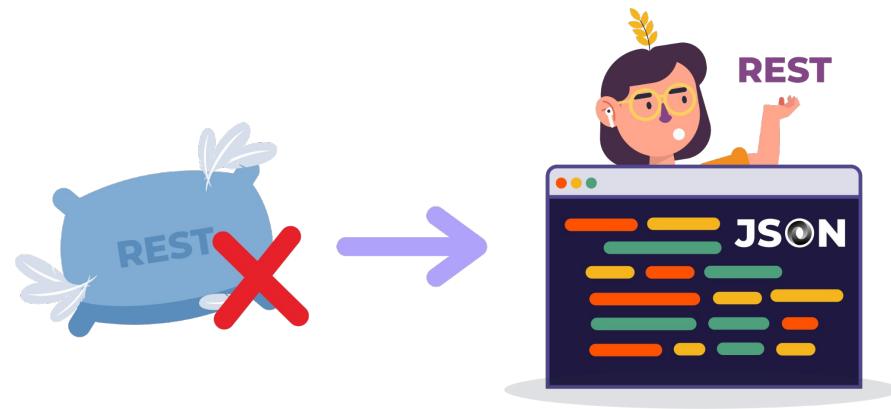


REpresentational State Transfer (REST)

Ini juga bukan bobok atau istirahat ya, tapi ini adalah sebuah kaidah dalam mendesain API juga. Nah, dibandingkan SOAP tadi, REST ini lebih populer dan menjadi standar industri.

Nah, kalo SOAP bertukar data pakai XML, nah kalo REST ini, akan bertukar data pakai JSON (Javascript Object Notation). Itu lho format yang pernah kita bahas di **Chapter 4 Topic 4 (HTTP Server)**.

Nah, belajar konsep REST API tuh penting banget, karena ini akan membantu banget dalam membuat HTTP Server.



Kita tadi udah bahas tentang kaidah dalam mendesain API secara singkat, nah sekarang kita coba bahas lebih detail tentang si REST API ini.





REpresentational State Transfer (REST)

Istilah REST sendiri pertama kali muncul di dalam disertasinya Roy Fielding.

Beliau ini adalah seorang software engineer di tahun 90an, dan beliau membahas detail tentang konsep REST itu sendiri di dalam disertasinya yang berjudul [Architectural Styles and the Design of Network-based Software Architectures](#).

Buat kamu yang penasaran bahas programming dari sudut pandang akademis bisa langsung kepoin linknya ya.

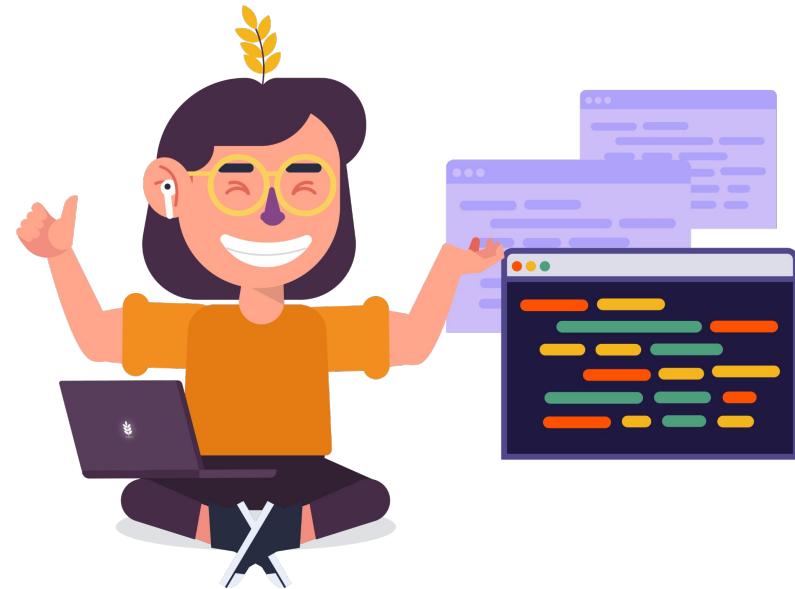


Rest API

HTTP Server yang didesain dengan menggunakan kaidah REST API, dan penerapannya sesuai, maka HTTP Server tersebut dikatakan RESTful.

Oke, to the point aja, ada beberapa poin yang kita perlu ketahui tentang REST API, yaitu:

- Apa itu **resource**?
- Aturan membuat endpoint (**Resource Identifier**)
- Cara **request body** dan **response body**



Apa itu Resource? 🤔

Setiap **informasi yang ada di dalam REST API** bisa kita sebut sebagai Resource. Berikut contoh-contoh resource dalam kehidupan kita sehari-hari:

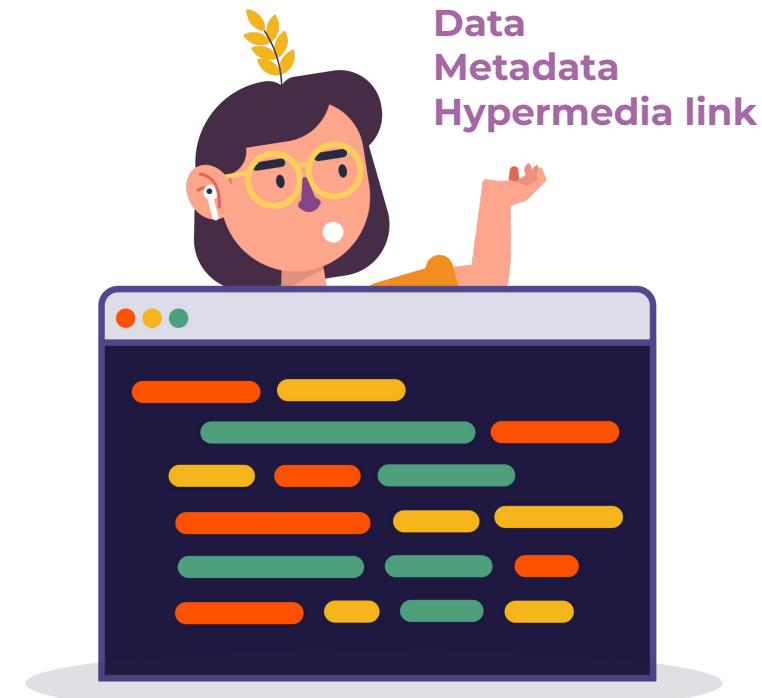
- Feed di Instagram
- Video di Youtube
- Tweet di Twitter





Resource sendiri biasanya terdiri dari 3 hal:

- **Data** itu sendiri, contoh: Gambar, Video, Tweet
- **Metadata** yang mendeskripsikan data tersebut, seperti tanggal, ukuran data tersebut, dan sebagainya
- Dan **hypermedia link** yang membantu client dalam transisi ke dalam state data berikutnya jika akan ada perubahan data.





Hmm... fungsi resource tadi untuk apa ya?

Yah, yang namanya informasi ya bestie. Pastinya kan bisa kita olah. Kita ambil contoh resource feed instagram. Kita bisa apain aja sih si feed itu?

- Kita bisa **buat** feed baru
- Kita bisa **edit** feed yang barusan kita post
- Kita bisa **lihat** feed yang barusan kita post
- Kita bisa **hapus** feed yang barusan kita post
- Feed kita bisa **lihat** sama user instagram lain

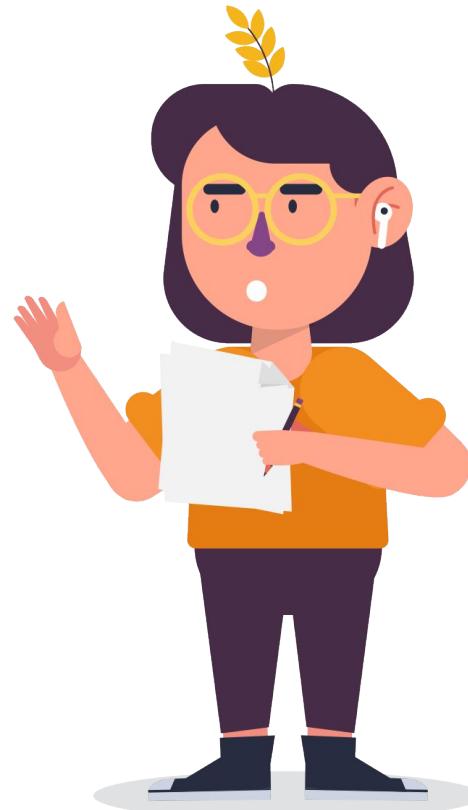
Apapun lah, yang penting data itu dipakai untuk diolah menjadi sesuatu.



Penamaan Resource

Cara penamaan resource itu juga ada aturannya lho! Ini masuk ke dalam cakupan REST API juga. Jadi, buat menamakan sebuah resource tuh aturannya kayak gini:

1. Nama resource dianjurkan menggunakan bahasa Inggris formal
2. Nama resource harus jamak kalo misal datanya lebih dari satu
3. Nama resource harus self-descriptive



Berikut nih, contoh-contoh resource yang digunakan dalam REST API :

- **Feeds**
- **Tweets**
- **Posts**
- **Todos**
- **Users**
- **Videos**





Resource Identifier

Agar client bisa dapetin data dari sebuah API perlu yang namanya **resource identifier**. Resource identifier ini kalo dalam konteks HTTP Server, dia akan berbentuk URL atau URI.

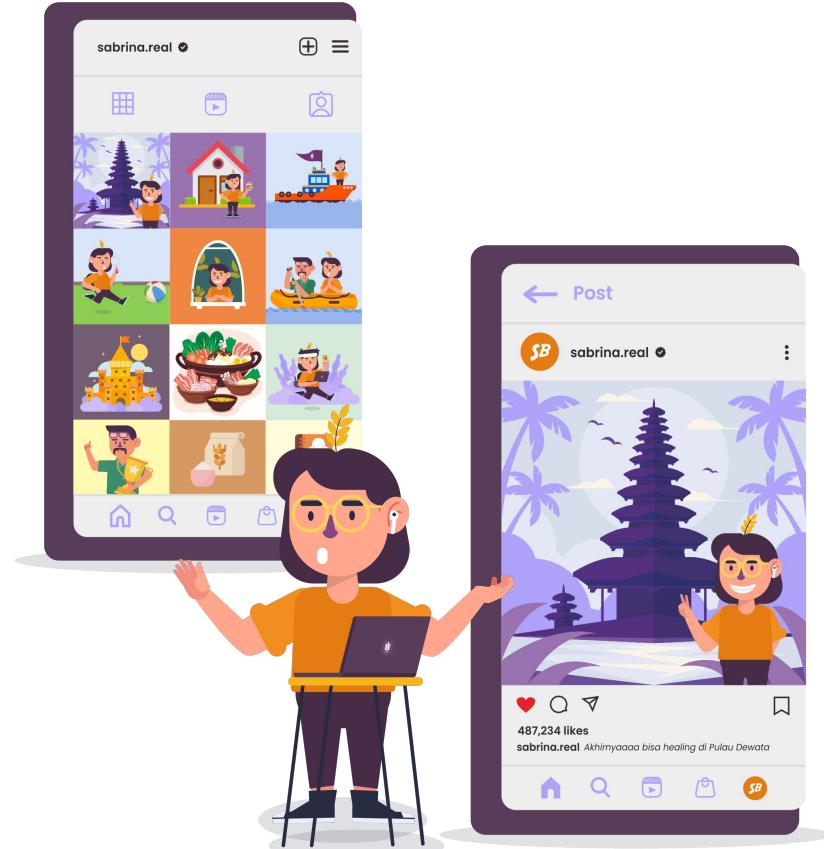
Resource identifier ini biasanya dibagi menjadi dua, yaitu :

1. **Collection**
2. **Resource**.



Kalo dari analogi feeds instagram tadi, feeds ini bisa dilihat sebagai daftar di halaman home instagram, dan feeds dilihat sebagai halaman sendiri di dalam instagram.

Feeds di halaman home inilah yang bisa kita sebut sebagai **Collection**, dan yang di halaman feed itu sendiri yang kita sebut sebagai **Resource**.



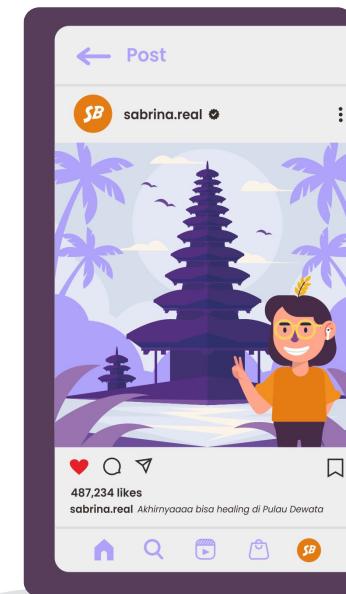
Berdasarkan pembagian itulah, resource identifier dari feeds dapat kita desain sebagai berikut:

- **/feeds** → Daftar feeds, misal daftar feeds-nya Sabrina
- **/feeds/{id}** → Feeds secara spesifik, misal feednya Sabrina waktu di Bali

/feeds inilah yang kita sebut sebagai Collection URL, dan **/feeds/{id}** kita sebut sebagai Resource URL



/feed



/feed/{id}

Nah, pembahasan resource identifier tadi dipakai untuk menentukan object dari sebuah resource di dalam server.

Pembahasan berikutnya tentang **request method** akan bahas apa yang akan dilakukan pada object tadi

Kalau udah mulai pusing, gapapa kita bahas pelan-pelan aja! Yuk ~





Resource Methods

Resource method akan **menentukan aksi apa yang akan dilakukan pada sebuah resource**. Berikut ini resource method yang umum dipakai :

- **Create**
Menambahkan entri baru pada resource tersebut
- **Update**
Mengupdate suatu entity di dalam resource tersebut
- **List**
Menampilkan daftar entri yang terdapat dalam suatu resource
- **Get**
Menampilkan spesifik entri di dalam sebuah resource
- **Delete**
Menghapus sebuah entri di dalam sebuah resource





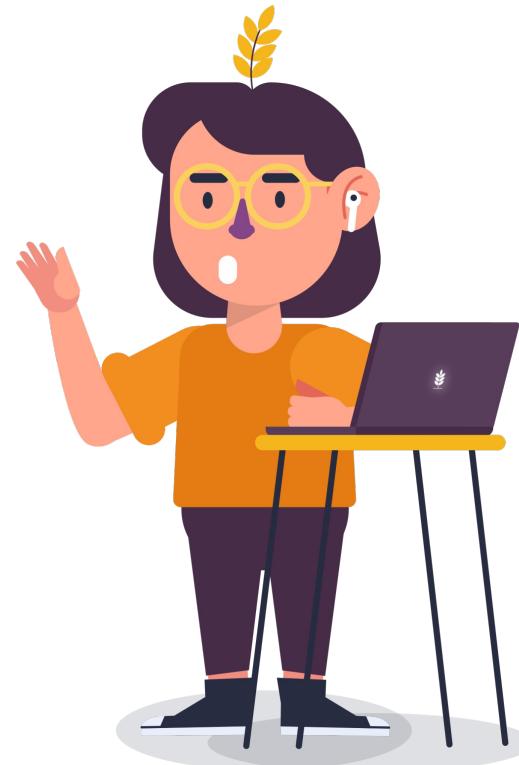
Standard Method	HTTP Mapping	HTTP Request Body	HTTP Response Body	Contoh
<u>List</u>	GET <collection URL>	N/A	Resource* list	GET /feeds
<u>Get</u>	GET <resource URL>	N/A	Resource*	GET /feeds/{id}
<u>Create</u>	POST <collection URL>	Resource	Resource*	POST /feeds
<u>Update</u>	PUT atau PATCH <resource URL>	Resource	Resource*	PUT /feeds/{id} atau PATCH /feeds/{id}
<u>Delete</u>	DELETE <resource URL>	N/A	N/A	DELETE /feeds/{id}



Terus kalo misal ada aksi yang diluar dari 5 methods tersebut gimana? 🤔

Sebenarnya sangat fleksibel, dan ada banyak standar yang bisa kita pakai, namun sangat direkomendasikan untuk mengikuti kaidah dari [Google API Design](#).

Jika ada **custom method, maka method tersebut harus berupa kata kerja**, sebagai contoh, kita ingin membuat aksi search pada suatu collection atau resource, maka dari itu, nama method tersebut harusnya **search**.





Custom Methods	HTTP Mapping	HTTP Request Body	HTTP Response Body	Contoh
<u>search</u>	GET <collection URL>:search	N/A	Resource* list	GET /feeds:search
<u>clone</u>	POST <resource URL>:clone	N/A	Resource*	POST /feeds/{id}:clone
<u>publish</u>	PUT <resource URL>:publish	N/A	Resource*	PUT /feeds/{id}:publish
<u>archive</u>	PUT <resource URL>:archive	N/A	Resource*	PUT /feeds/{id}:archive

Dari tadi perasaan kita ngomongin konsepnya mulu, yuk sekarang kita langsung praktik penerapan REST API ke Express biar makin cepat pahamnya ~





TOKO BUKU SAY



Rest API ke Express

Buat menerapkan REST API ke dalam Express.js kita akan pakai contoh studi kasus ya~

Bayangin kamu punya toko buku, dan kamu ingin membuat katalog untuk buku-buku yang kamu jual ke dalam website toko buku-mu.

Dari sini kita bisa petakan, apa saja resource yang ada di dalam website toko buku tersebut, dan aksi apa saja yang bisa kita lakukan terhadap resource tersebut.



TOKO BUKU SAY



Book Resource

Resource yang terdapat di dalam website toko buku tersebut adalah **katalog buku**, atau kalau kita ingin namakan sesuai dengan kaidah REST API, maka resource tersebut akan bernama **books**.

Books ini, nantinya akan memiliki atribut title, cover image, synopsis, author, publisher, dan price.





Atribut	Tipe Data	Contoh
Title	String	Industrial Society And Its Future
Cover Image	String	https://placeimg.com/640/480/any
Synopsis	String	Lorem Ipsum is simply dummy text of the printing and typesetting industry.
Author	String	Theodore John Kaczynski
Publisher	String	F.C
Price	Integer	0



Book Resource Methods

Dari studi kasus tadi, kita juga bisa petakan aksi-aksi apa saja yang bisa kita lakukan terhadap Book Resource. Diantaranya :

- Create book
- Update book
- Get book
- List books
- Delete book

Get Book
Update Book
List Book
Create Book
Delete Book





URL	Method	Action	Description
/books/{id}	GET	Get book	Endpoint untuk mengambil data dari satu buku berdasarkan ID.
/books/{id}	PUT	Update book	Endpoint untuk mengupdate data dari satu buku berdasarkan ID.
/books/{id}	DELETE	Delete book	Endpoint untuk menghapus data dari satu buku berdasarkan ID.
/books	GET	List books	Endpoint untuk mengambil daftar buku yang ada di dalam database atau server.
/books	POST	Create book	Endpoint untuk menambahkan buku ke dalam database atau server.



```
npm init -y  
npm install --save express
```

Skuy terapin Rest API ke dalam Express.Js ~

Oke, desain API kita setidaknya sudah jadi, sekarang saatnya kita implementasikan desain tersebut ke dalam Express.js

Pertama, seperti biasa kita inisialisasikan project node.js dan install express.



Kedua, buat server sederhana dengan Express.js.

Jangan lupa juga untuk pasang JSON parser middleware dengan menggunakan **express.json()**. Middleware ini digunakan untuk membaca request yang dikirim dengan format JSON.



```
const express = require("express");
const port = process.env.PORT || 8000;
const app = express();

app.use(express.json());

// Endpoint-endpoint kita
// akan ditulis disini

app.listen(port)
```



```
● ● ●

const books = [];

class Book {
  constructor(params) {
    this.title = params.title;
    this.coverImage = params.coverImage;
    this.synopsis = params.synopsis;
    this.publisher = params.publisher;
    this.author = params.author;
    this.price = params.price;
  }

  static create(params) {
    const book = new this(params);

    books.push(book);

    return book;
  }

  static find(id) {
    const book = books.find((i) => i.id ===
Number(id));
    if (!book) return null;

    return book;
  }
}
```

Ketiga, buatlah sebuah class yang merepresentasikan sebuah resource.

Class ini harus mengimplementasikan method untuk membuat entri baru, mengupdate entri yang sudah ada, mencari specific entri, menghapus specific entri, dan menampilkan semua entri yang ada di dalam resource tersebut.

Source code dari class Book dapat [dilihat disini](#).



Keempat, buatlah endpoint untuk kelima aksi dari resource book.

Kode disamping adalah **implementasi dari Get book dan List books**, yang mana response dari kedua endpoint tersebut adalah JSON, bahkan jika ada error pun, response-nya juga tetap JSON.



```
// GET /books (List books)
app.get("/books", (req, res) => {
  const books = Book.list();
  res.status(200).json(books);
}

// GET /books/:id} (Get book)
app.get("/books/:id", (req, res) => {
  const book = Book.find(req.params.id);

  if (!book) res.status(404).json({
    error: "Book not found!"
  });

  res.status(200).json(book);
})
```



```
// POST /books (Create book)
app.post("/books", (req, res) => {
  const book = Book.create(req.body);
  res.status(201).json(book);
})

// PUT /books/{id} (Update book)
app.put("/books/:id", (req, res) => {
  const book = Book.find(req.params.id);
  if (!book) return res.status(404).json({
    error: "Book not found!"
  })

  book.update(req.body);

  res.status(200).json(book);
})
```

Kode disamping adalah **implementasi dari aksi create dan update book**. Response dari kedua endpoint tersebut adalah data book itu sendiri dalam bentuk JSON.

```
{
  "id": 1,
  "title": "Industrial Society And Its Future",
  "coverImage": "https://placeimg.com/640/480/any",
  "synopsis": "Lorem ipsum",
  "publisher": "F.C",
  "author": "Theodore Kaczynski",
  "price": 0
}
```



```
// DELETE /books/{id} (Delete book)
app.delete("/books/:id", (req, res) => {
  const book = Book.find(req.params.id);
  if (!book) return res.status(404).json({
    error: "Book not found!"
  });

  book.delete();

  res.status(204).end();
})
```

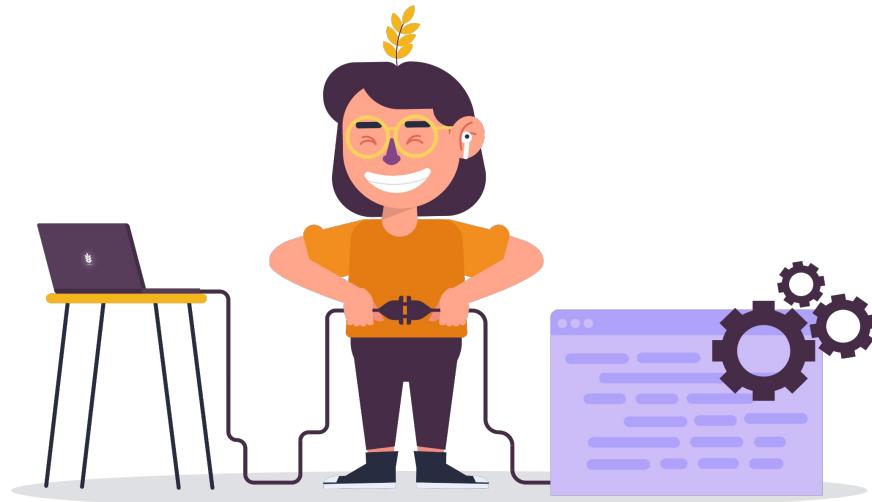
Kode disamping adalah **implementasi dari Delete book.** Yang dalam kaidah REST API, jika kita menghapus sebuah data dan berhasil, maka response-nya harus No Content atau tidak ada response body di dalamnya.



Semua endpoint sudah jadi, dan tugas kalian selanjutnya adalah melakukan testing pada masing-masing endpoint. Dari tes yang dilakukan kita bisa mengetahui apakah endpoint sudah berjalan dengan baik atau belum. Testing ini bisa dengan menggunakan Postman. Berikut daftar-daftar endpoint yang bisa kalian coba :

- POST /books (Create book)
- GET /books (List books)
- GET /books/1 (Get book)
- PUT /books/1 (Update book)
- DELETE /books/1 (Delete book)

Source code dari API tadi dapat kalian download [disini](#).



Kalau kamu ingin eksplor lebih Desain API lain, yang umum digunakan di industri beserta best practice-nya kamu bisa klik link dibawah ini lho !

- [JSON:API](#)
- [JSend](#)
- [Google API Design](#)



Referensi buat kamu yang suka kepo-kepo~

- [RESTful API](#)
- [JSON:API](#)
- [JSend](#)
- [Google API Design](#)
- [RESTful API Node.js](#)





Gimana, sekarang udah tau kan gimana cara mendesain API dengan baik dan benar?

Selanjutnya kita bakal bahas gimana cara menyimpan data di dalam HTTP Server, biar datanya kesimpul meskipun server kita mati.

Terima Kasih!



Next Topic

loading...