



Open API

Gold - chapter 6 - Topic 4

**Selamat datang di Chapter 6 Topik 4 online
course Fullstack Web dari Binar Academy!**

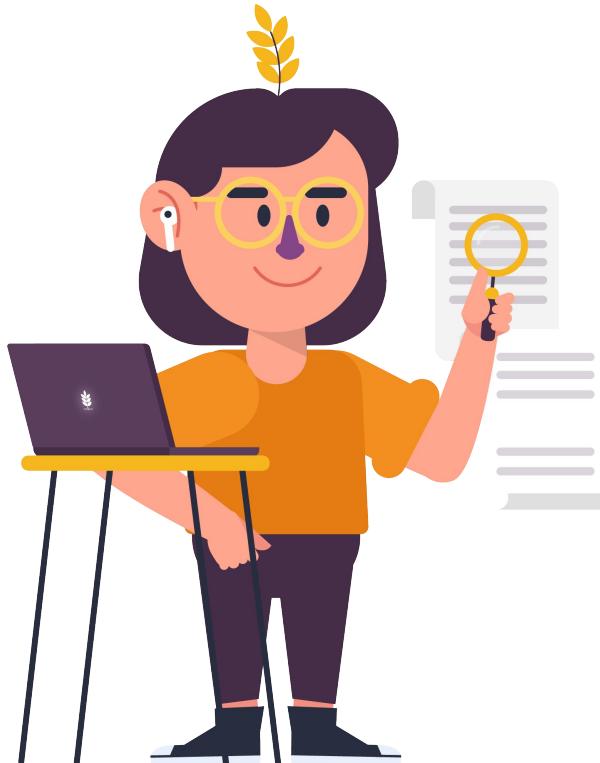




Hari Raya Lebaran makan ketupat, selamat datang di topic 4 !

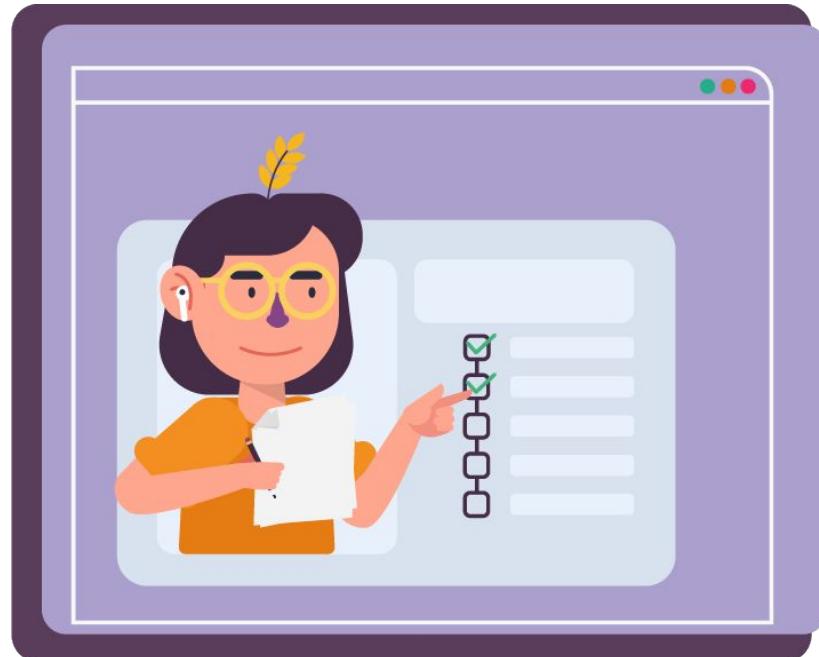
Di topik sebelumnya pada chapter ini, kamu sudah belajar apa tentang authentication & authorization yang membantu kita untuk memasuki laman web yang sudah dibuat.

Nah di topik ke 4 ini, kita akan bahas tentang **Open API** yang akan membantu kita untuk melakukan komunikasi terhadap komputer, maupun sesama developer.



Detailnya, kita bakal bahas hal-hal berikut ini:

- Mengenal konsep open API
- Mengenal struktur open API dan struktur serta kriterianya
- Memahami cara kerja swagger tools dan implementasinya
- Cara integrasi dokumentasi API
- Mock API (Swagger Codegen)





Pernah ga sih kamu berpikir, ketika seorang Backend and Frontend Developer bekerja secara terpisah, gimana cara mereka komunikasikan endpoint-endpoint backend yang sudah dikembangkan?

Nahh.. pembahasan materi ini akan menjawab keresahan kamu tadi. Disini kita akan bahas **Open API** untuk mengintegrasikan komunikasi antar keduanya.

Waaaa kerenn yaa ~



Ketika kita membuat sebuah remote API (Backend Web API) tentu saja kita **perlu mendokumentasikan API tersebut, mulai dari operasi apa saja yang bisa kita jalankan melalui remote API tersebut, dan apa saja parameter yang dibutuhkan.** Hal ini dilakukan agar user remote API dapat menjalankan operasi-operasi dengan baik.

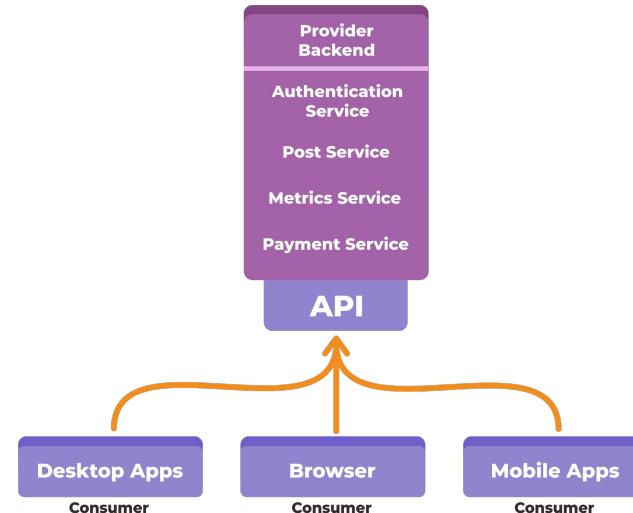
Bentuk dokumentasi itu bisa macam-macam bentuknya, bisa dalam bentuk PDF, README, Website, ataupun bentuk-bentuk dokumen yang lainnya.

Cara mendokumentasikannya pun beragam, salah satunya dengan menggunakan **Open API Specification**.



Kita mulai dari API dulu ya~

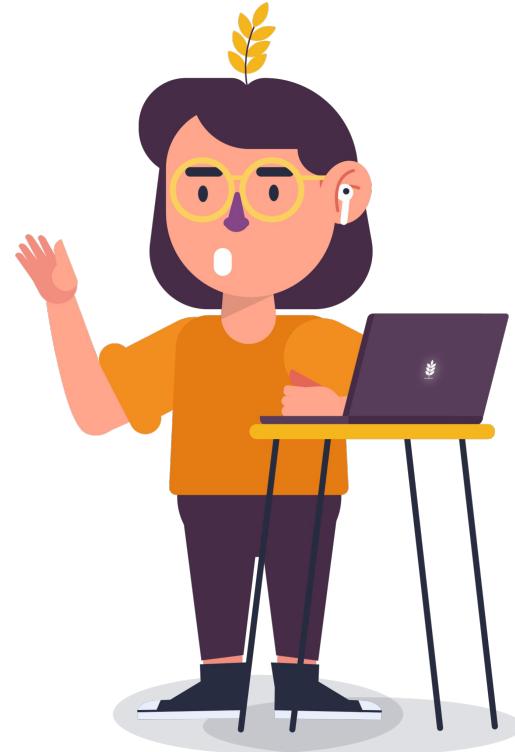
API atau application programming interface adalah **sebuah penghubung antar dua aplikasi**. Sebagai contoh adalah aplikasi frontend dan backend kita. API mempunyai sederet operasi yang bisa dipanggil untuk menjalankan beberapa fungsi dari aplikasi backend kita.





Berhubungan dengan apa yang kita pelajari di materi sebelumnya, API yang kita bicarakan disini adalah **sebuah remote API yang dapat kita panggil menggunakan protokol HTTP atau protokol yang mirip dengan HTTP** (WebSocket dsb).

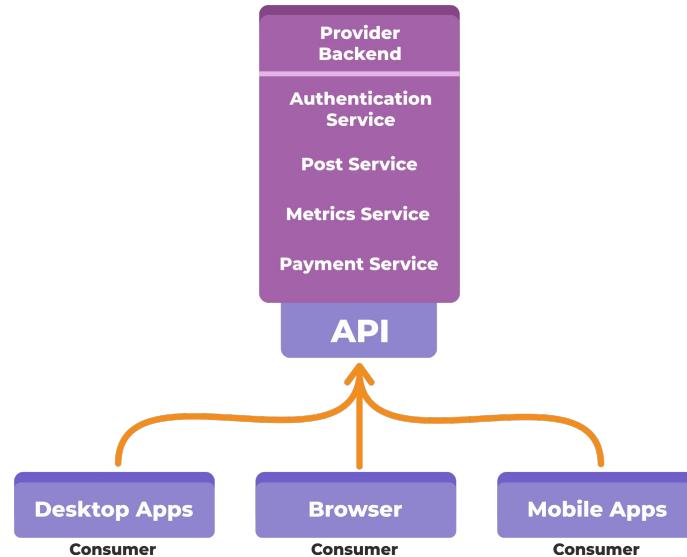
API biasa disebut sebagai **kontrak**, karena API biasa tidak akan pernah rusak. Layaknya sebuah hukum, yang ketika sudah ditetapkan maka hampir tidak akan berubah, kecuali ketika diamandemen.



Untuk merangkum definisi tentang API, secara singkat, kita bisa simpulkan sebagai berikut:

- Pihak yang **menyediakan servis** (seperti: login, register, dan logika bisnis lainnya) melalui API disebut sebagai **Provider** (biasa disebut sebagai Backend)
- Pihak yang **meminta servis** tersebut melalui API disebut sebagai **Consumer** (biasa disebut sebagai Frontend)

Wujud dari API biasanya adalah sebuah *function*, *method* ataupun *endpoint*. Nah **cara mengomunikasikan pemahaman tentang sebuah aplikasi backend Web API** tuh punya *endpoint* apa aja, salah satu solusinya adalah menggunakan **Open API**.

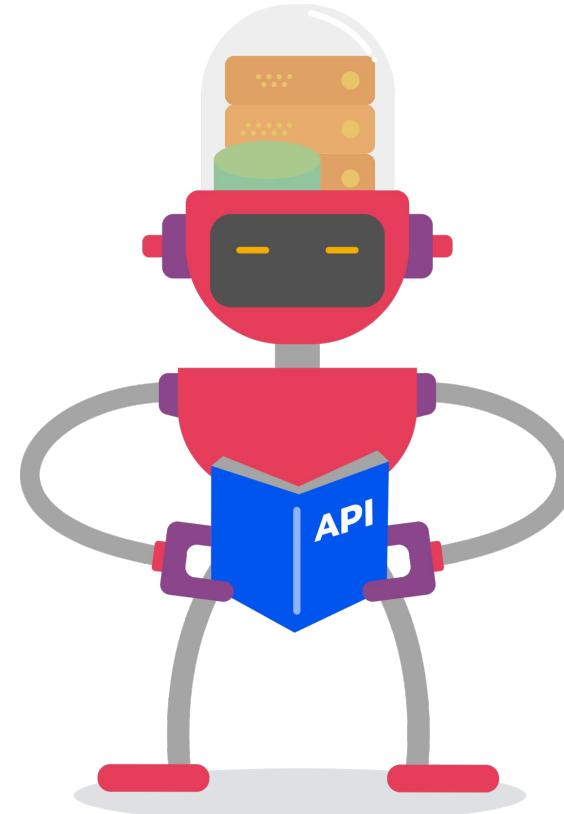




Sab, coba jelaskan Open API dong~

Open API adalah **sebuah deskripsi dari sebuah remote API yang biasanya dapat kita panggil menggunakan protokol HTTP atau WebSocket** (protokol yang mirip dengan HTTP). Deskripsi dengan format JSON atau YAML ini dibaca oleh mesin sehingga dapat diurai dan dibaca sebagai sebuah object di dalam sebuah program.

Ketika kita mendeskripsikan sebuah remote API menggunakan Open API, tidak hanya kita saja manusia yang dapat membacanya, namun mesin juga dapat membacanya.





“Terus apa keuntungan menggunakan Open API, dibanding kita mendokumentasikannya dengan dokumen seperti PDF atau README saja?”

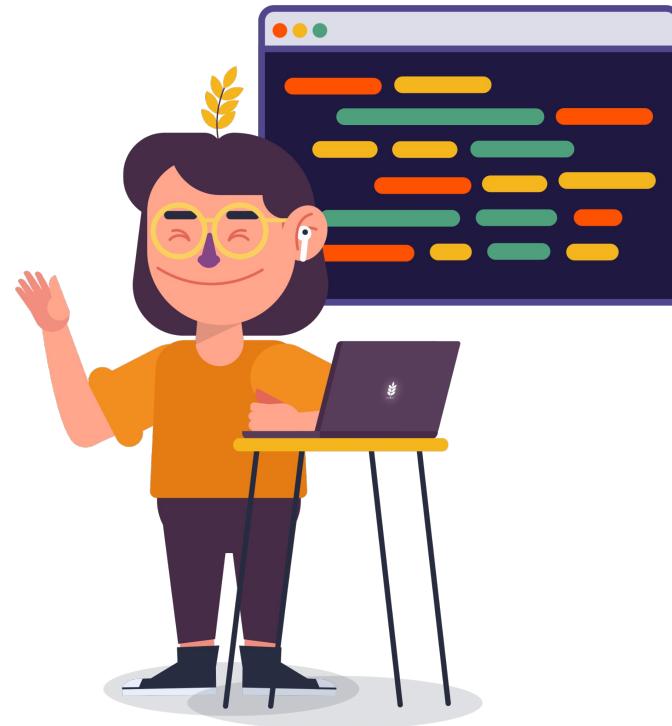
Keuntungan mendeskripsikan remote API menggunakan Open API adalah, kita **dapat menggunakan banyak tools yang dapat membantu kita dalam pengembangan aplikasi, contohnya adalah client codegen.** Codegen dapat membantu kita untuk membuat sebuah *library* atau *SDK (software development kit)* untuk mengonsumsi remote API yang kita buat yang nantinya digunakan oleh aplikasi frontend kita. Tentu saja ini akan **mempercepat proses pengembangan web.**





Untuk melihat lebih jauh lagi tentang tools-tools apa saja yang dapat digunakan dengan Open API bisa lihat [disini](#) ya!

FYI, biasanya bentuk dari Open API adalah file yang memiliki ekstensi .json atau .yaml





Kenapa kita perlu menggunakan Open API?

Inti dari kenapa kita perlu menggunakan Open API adalah komunikasi. **Komunikasi terhadap komputer, maupun sesama developer.** Yap, simpelnya kayak gitu, tapi biar lebih paham lagi sama apa aja kegunaannya, kita coba bahas satu-satu kuy!

API GUIDE

REQUEST URL FORMAT :

`http://nnn.com/<username>/<item ID>`

SERVER WILL RETURN AN XML DOCUMENT WHICH CONTAINS :

- The Requested Data
- Documentation Describing How the Data is Organized Spatially

API KEYS

To obtain API access, contact the X509 Authenticated Server and request an ECDH-RSA TLS Key...



If you do things right, it can take people a while to realize that your “API Documentation” is just Instructions for How to Look at Your Website



1. Perencanaan API akan jauh lebih baik

Mungkin bisa direfleksikan dari proses kalian mengerjakan challenge-challenge sebelumnya, kalian merasa *clueless* ketika ingin membuat sebuah API.

Nah, dengan membuat Open API Documentation-nya terlebih dahulu, **proses pembuatan remote API (RESTful API) kalian akan jauh lebih terencana.**

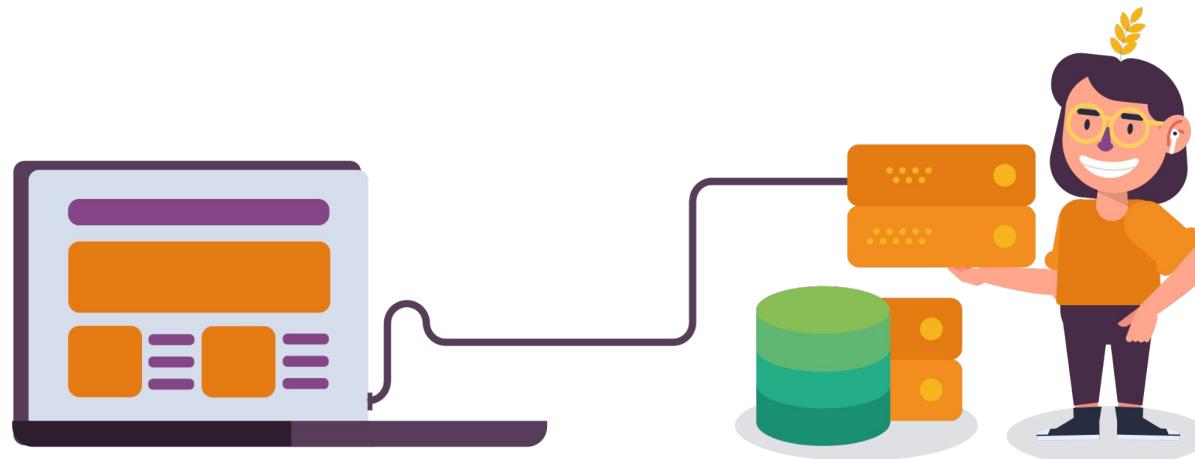




2. Codegen mempermudah pengembangan client side application

Meskipun janjinya API Contract tidak akan merusak implementasi yang sudah ada di Client Side, namun pada kenyataannya terkadang masih ingkar janji layaknya distopia belaka.

Nah, kita bisa gunakan Open API untuk membuat SDK secara otomatis dengan Codegen. Dalam perubahan kontrak di API, **frontend developer (consumer) tidak perlu repot-repot mengimplementasi ulang cara request ke API** kita, mereka tinggal generate ulang SDK tersebut, dan menyesuaikan cara penggunaannya di Client Side Application

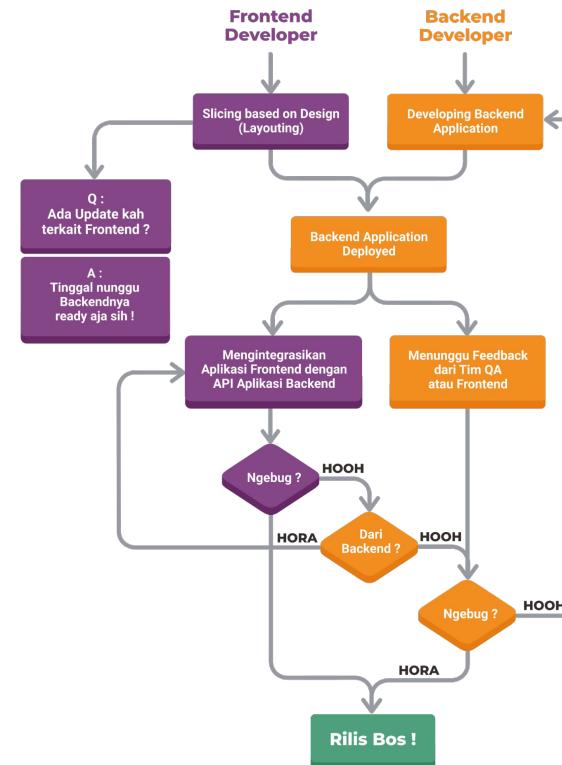


3. Mock API akan mempercepat pengembangan aplikasi kita

Diagram di samping menunjukkan bagaimana software development lifecycle ketika **tidak menggunakan Open API Documentation**. Integrasi aplikasi frontend ke aplikasi backend, akan **sangat bergantung pada ketersediaannya API dari aplikasi backend terlebih dahulu**. Artinya, frontend developer tidak bisa meneruskan pekerjaannya sampai aplikasi backend sudah siap untuk dipakai.

Nah dependensi frontend terhadap backend ini dapat dipecahkan dengan Open API Documentation. Karena **kita bisa membuat kode dengan codegen** untuk membuat sebuah aplikasi backend yang sudah mengimplementasikan API yang sudah didefinisikan pada dokumen Open API. Aplikasi ini biasa disebut sebagai Mock API.

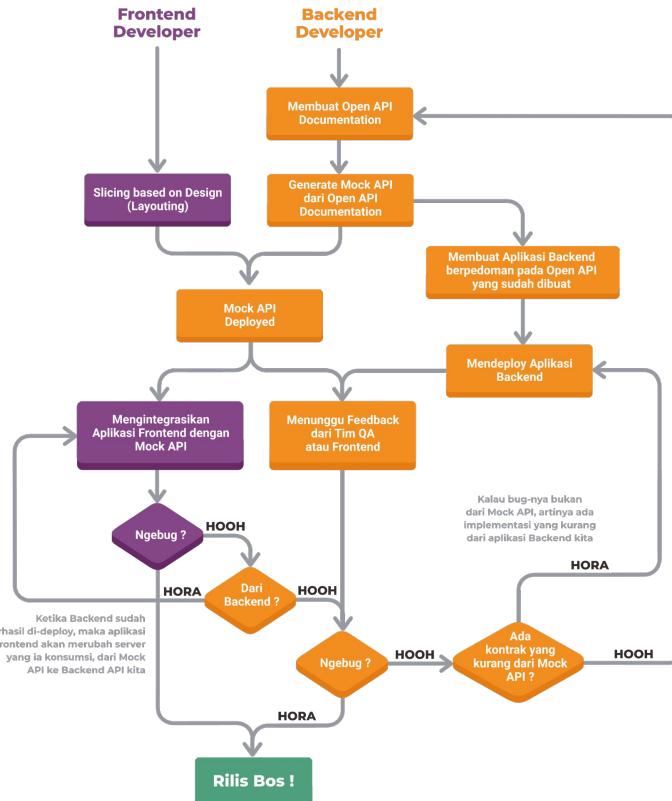
COVENTIONAL SOFTWARE DEVELOPMENT LIFECYCLE



Selagi aplikasi frontend menggunakan Mock API dari backend, maka backend developer sedang membuat implementasi dari API tersebut, dan berjanji tidak akan membuatnya berbeda dari Mock API dari segi interface-nya (cara pakai).

Maka dari itu, **ketika aplikasi backend sudah siap untuk dikonsumsi, aplikasi frontend tinggal mengganti server URL yang akan mereka gunakan untuk memanggil API**, dari yang awalnya server URL ke Mock API, menjadi server URL ke Backend API yang sebenarnya.

Hal ini tentu saja akan mempercepat proses pengembangan aplikasi, karena frontend development tidak lagi dependen dengan siap tidaknya aplikasi backend yang sesungguhnya. Dan, sebagai backend developer, mereka akan mendapatkan feedback lebih cepat dari sisi interface-nya (Cara pakai, dan skema dari response-nya).





Nah, gimana sekarang pemahaman kalian tentang pengertian dan alasan penggunaan Open API. Udah mantap kan? Pastinya udah donggg.

Setelah ini kita akan lihat Mengenal dokumentasi API dan struktur serta kriterianya, letsgooo~





Spesifikasi Open API

First thing first : rules.

Dalam mendeskripsikan sebuah remote API ada kaidah-kaidah yang wajib kita ikuti di dalam menulis dokumennya.

Dokumen Open API ini berwujud json atau yaml. Artinya, deskripsi dari remote API kita nanti akan berbentuk object, yang memiliki atribut-atribut yang berkaitan dengan remote API.

Apa saja sih atributnya? Dan seperti apa struktur data dari object tersebut? Cus kita lanjut ~

Document Open API





Oh iya! FYI, materi-materi dan dokumen dibawah ini akan selalu berdasarkan Open API Specification versi 3.0.0, jika referensi ke versi 2.0.0 dibutuhkan, silahkan bisa kamu eksplor secara mandiri [disini](#) ya!





Struktur dokumen Open API

Seperti yang sudah disebutkan pada bagian sebelumnya, dokumen **Open API ini bermaksud agar bisa dibaca oleh mesin**, maka dari itu, kita perlu menyimpan dokumen ini dalam bentuk yang dikenali mesin, yaitu dalam bentuk JSON dan YAML.

Karena bentuknya JSON atau YAML, maka dari itu pastinya data yang kita simpan pada file tersebut akan berupa sebuah object, struktur dari object tersebut merepresentasikan atau mendeskripsikan remote API kita, mulai dari apa sih nama remote API kita, siapa maintainernya, lisensinya dan sebagainya.





Syntax dari dokumen Open API

Seperti yang sudah disebutkan diatas, dokumen ini ditulis menggunakan YAML atau JSON, berikut ini contoh kode YAML

```
# Anything after a hash sign is a comment
anObject:
  aNumber: 42
  aString: This is a string
  aBoolean: true
  nothing: null
  arrayOfNumbers:
    - 1
    - 2
    - 3
```



Dan berikut contoh kode JSON

Pilihlah salah satu tipe dokumen ketika menulis dokumentasi menggunakan Open API, tapi yang sangat dianjurkan adalah dengan menggunakan YAML, karena lebih manusiawi untuk dibaca.

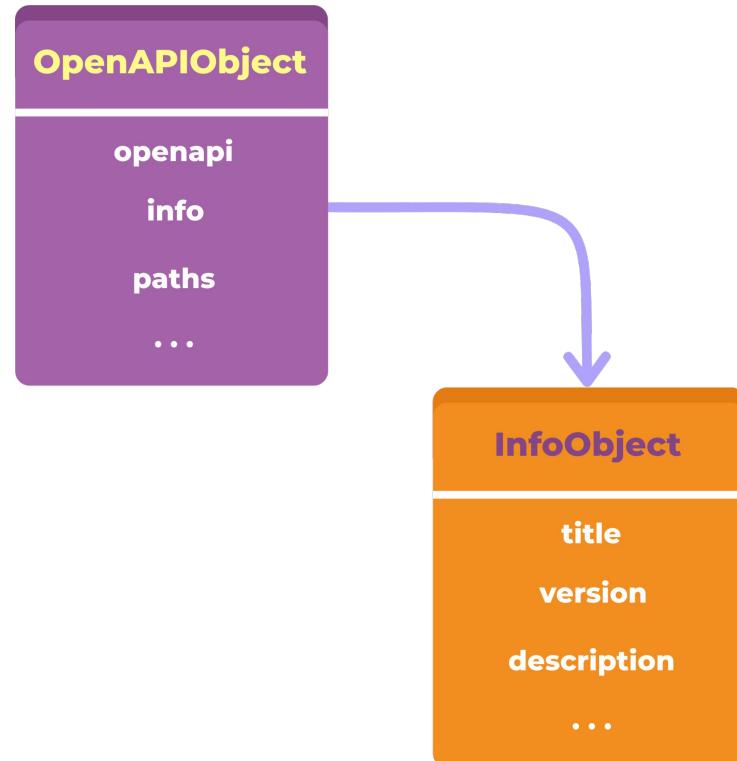
```
{  
  "anObject": {  
    "aNumber": 42,  
    "aString": "This is a string",  
    "aBoolean": true,  
    "nothing": null,  
    "arrayOfNumbers": [  
      1,  
      2,  
      3  
    ]  
  }  
}
```



The Open API Object

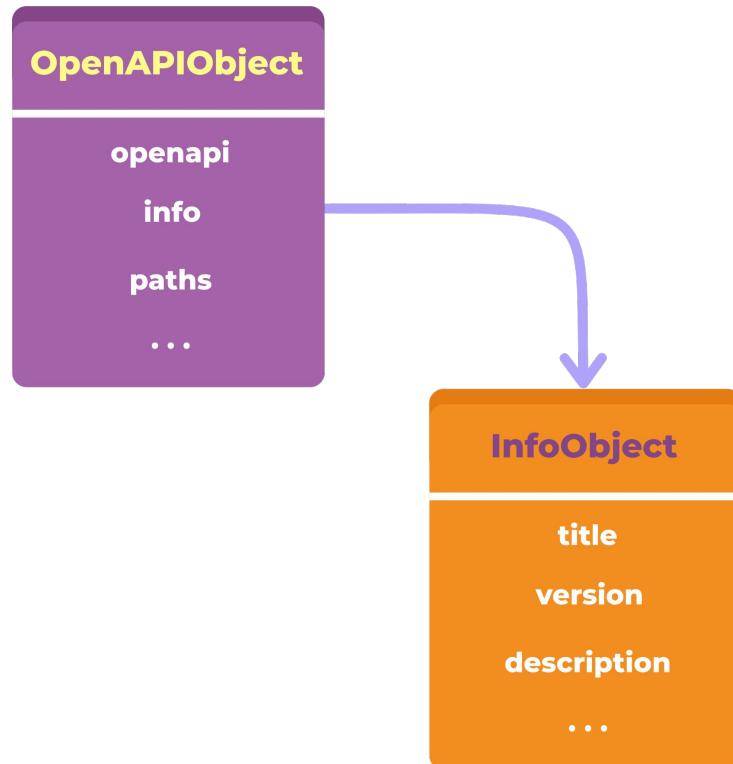
Root object atau object dengan level tertinggi di dalam dokumen ini kita sebut dengan Open API Object, dimana object ini mendefinisikan metadata dari remote API yang ingin kita dokumentasikan. Gambaran atribut dari objectnya bisa dilihat dari gambar disamping ya.

Tapi biar lebih jelas, kita bahas satu per satu yuk!





- **openapi** dipakai untuk memberi tau, kita ingin menggunakan spesifikasi versi keberapa dari Open API Specification.
- **info** adalah sebuah object yang berisi informasi terkait API yang dideskripsikan, berisi siapa author-nya, nama api-nya apa dan sebagainya.
- **paths** adalah sebuah object yang digunakan untuk mendeskripsikan semua rute yang ada di dalam API yang dideskripsikan.



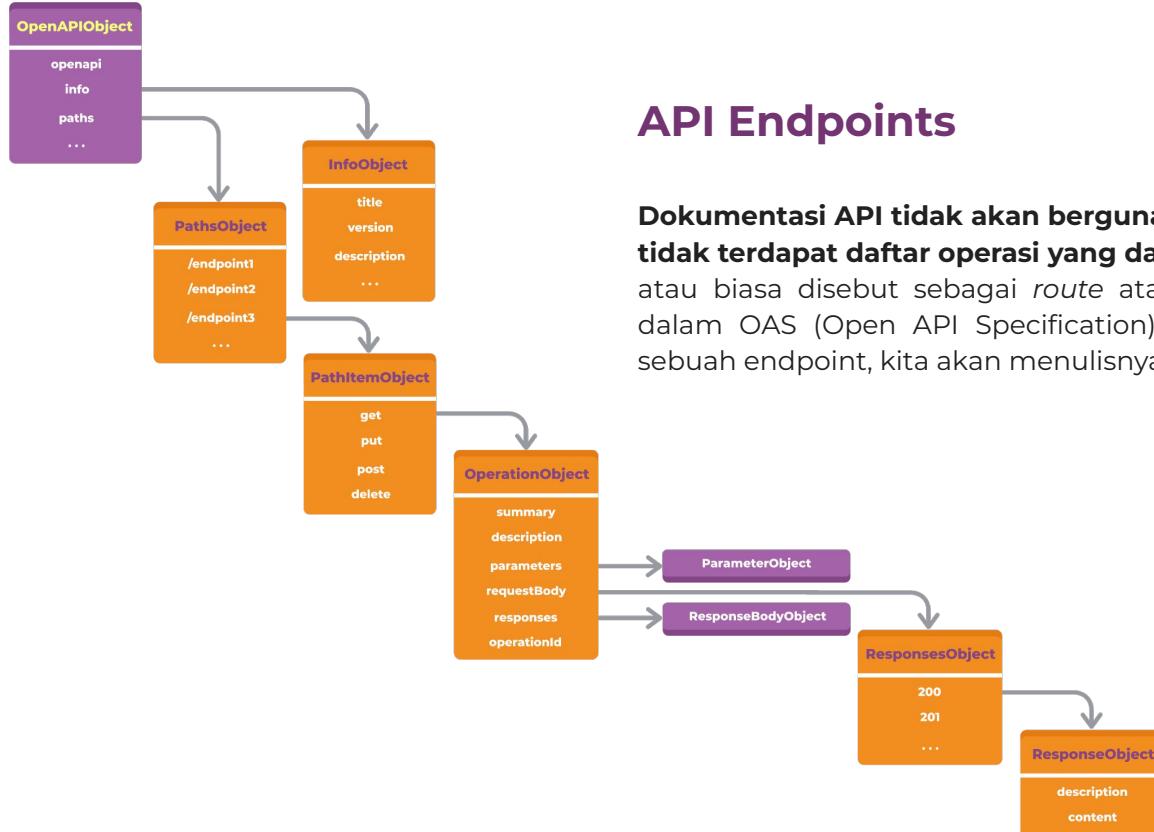


Berikut contoh dokumentasi minimal yang ditulis menggunakan YAML:

Dokumen tersebut sudah valid. Namun tidak memberi tahu kita lebih tentang apa saja yang API yang disediakan, dan bagaimana cara memanggilnya. Maka dari itu kita perlu mendefinisikan operasi apa saja yang dapat dilakukan, bahasan ini akan kita lanjut pada bagian berikutnya.



```
openapi: 3.0.0
info:
  title: Medium Blogging API
  version: 0.0.1
paths: {} # No endpoints defined
```



API Endpoints

Dokumentasi API tidak akan berguna kalau di dalam dokumentasi tersebut tidak terdapat daftar operasi yang dapat dilakukan (**endpoint**). API Endpoint atau biasa disebut sebagai *route* atau *operation* disebut sebagai **Paths** di dalam OAS (Open API Specification), maka dari itu, untuk mendefinisikan sebuah endpoint, kita akan menulisnya dibawah object bernama paths.



The Paths Object

Paths adalah sebuah **atribute dengan tipe data object, dimana tiap atribut dari object tersebut adalah sebuah string yang menandakan sebuah endpoint dari suatu API.** Maka tiap atribut dari paths harus dimulai dengan / sebagai prefiks dari endpoint.

Kalau kamu mau lihatcontoh Open API Object dengan paths, cek gambar disamping ya ~

Pada setiap atribut di dalam paths adalah sebuah PathItemObject, dimana di dalam atribut tersebut, adalah sebuah object yang mana atribut-atributnya adalah sebuah metode HTTP, baik itu GET, POST, PUT, maupun DELETE.

```
● ● ●  
openapi: 3.0.0  
info:  
  title: Medium Blogging API  
  version: 0.0.1  
paths:  
  /posts: {} # We can conclude that this API has path named /posts  
  /posts/{id}: {} # Path with path parameter
```



The Path Item Object

Path Item Object adalah sebuah **object yang berada pada sebuah atribut dari object paths, atribut dari object ini adalah metode HTTP**, baik itu GET, POST, PUT, PATCH, maupun DELETE. Tiap atribut itulah yang menandakan adanya operasi di dalam sebuah API kita.

Dari dokumen diatas, dapat kita simpulkan bahwa *Medium Blogging API* memiliki dua *path* yaitu /posts dan /posts/{id}, dimana, /posts, memiliki 2 operasi, yaitu GET, dan POST, dan untuk /posts/{id}, memiliki 3 operasi, yaitu DELETE, PUT, dan GET.

Dokumen disamping masih belum valid, karena tiap operasi wajib berisi OperationObject, yang mana memiliki beberapa atribut yang wajib ada, seperti responses dan sebagainya.

```
openapi: 3.0.0
info:
  title: Medium Blogging API
  version: 0.0.1
paths:
  /posts:
    get: {}
    post: {}
  /posts/{id}:
    get: {}
    put: {}
    delete: {}
```



Response Body Message

Setiap operasi wajib mendeskripsikan hasil dari operasi tersebut, baik itu berupa *No Content* maupun ada konten. Untuk mendeskripsikan sebuah response dari sebuah operasi, kita perlu menambahkan atribut bernama responses ke dalam OperationObject, dimana OperationObject ini berada di dalam PathItemObject.





The Responses Object

Setiap operasi tentunya memiliki beberapa jenis response, seperti 422, 400, 500 atau 200, semua response tentu dibedakan berdasarkan *HTTP Status Code*-nya, maka dari itu **pada OAS, setiap operasi wajib mencantumkan setidaknya satu jenis response menggunakan *HTTP Status Code* yang ditulis di dalam sebuah string sebagai nama atribut.**

Dokumen diatas masih **belum valid**, karena tiap atribut di dalam ResponsesObject, wajib memiliki atribut yang bernama description, yang mana akan kita bahas pada bagian berikutnya.

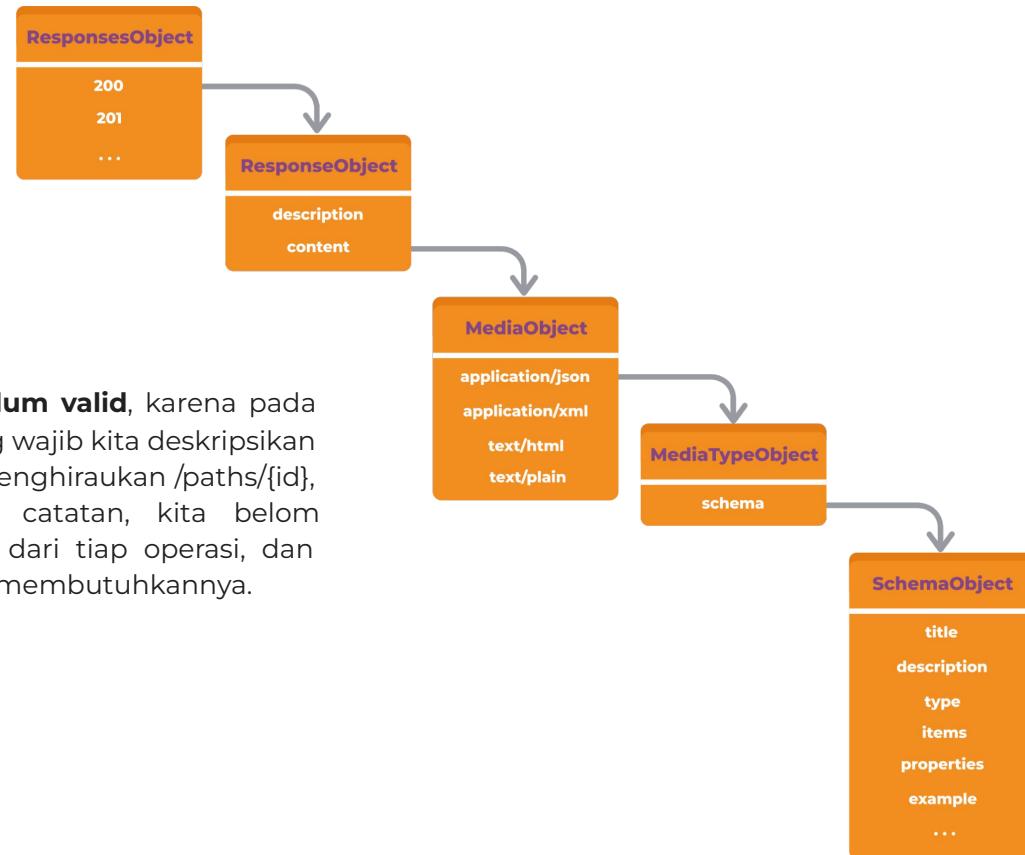
```
openapi: 3.0.0
info:
  title: Medium Blogging API
  version: 0.0.1
paths:
  /posts:
    get:
      responses:
        "200": {}
        post:
          responses:
            "201": {}
            "400": {}
            "401": {}
            "422": {}
            "500": {}
  /posts/{id}:
    get:
      responses:
        "200": {}
        "403": {}
        "404": {}
        put:
          responses:
            "200": {}
            "400": {}
            "403": {}
            "404": {}
            "422": {}
        delete:
          responses:
            "204": {}
            "403": {}
            "404": {}
```



The Response Object

Response Object adalah **sebuah object yang digunakan dalam OAS untuk mendefinisikan seperti apa sih response dari operasi yang dijalankan.** Object ini memiliki atribut yang wajib ada di dalamnya, yaitu description, yang mana hanyalah deskripsi dari sebuah response, namun tidak menutup kemungkinan juga kita akan menambahkan skema dari response body dari sebuah operasi.

```
openapi: 3.0.0
info:
  title: Medium Blogging API
  version: 0.0.1
paths:
  /posts:
    get:
      responses:
        "200":
          description: New post is created
```



Gambar pada slide sebelumnya itu masih **belum valid**, karena pada *path /paths/{id}*, terdapat *path parameter*, yang wajib kita deskripsikan di tiap operasi dibawahnya. Namun kita kita menghiraukan */paths/{id}*, maka dokumen ini sudah valid, dengan catatan, kita belum mendeskripsikan, wujud dari *response body* dari tiap operasi, dan wujud dari *request body* dari tiap operasi yang membutuhkannya.



The Media Object

MediaObject adalah **object yang memiliki atribut dalam bentuk sebuah Content-Type dari sebuah response**, berikut adalah contoh daftar Content-Type yang merupakan nama atribut dari MediaObject:

- application/json
- application/xml
- text/html
- text/plain

Atribut-atribut tersebut memiliki nilai berupa MediaTypeObject.



```
openapi: 3.0.0
info:
  title: Medium Blogging API
  version: 0.0.1
paths:
  /posts:
    get:
      responses:
        "200":
          description: List of post is retrieved
          content:
            application/json: {}
```



The Media Type Object

MediaTypeObject adalah sebuah object yang mendefinisikan bentuk response-nya nanti seperti apa, object ini memiliki atribut-atribut sebagai berikut:

- schema yang berisi definisi data, atribut ini wajib ada ketika kita membuat sebuah MediaTypeObject
- example yang berisi data
- examples yang berisi data jika bentuk datanya berupa array
- encoding yang berisi encoding dari data tersebut

Atribut schema adalah sebuah SchemaObject, yang mana atribut ini digunakan untuk mendefinisikan sebuah bentuk data dari sebuah response.

```
openapi: 3.0.0
info:
  title: Medium Blogging API
  version: 0.0.1
paths:
  /posts:
    get:
      responses:
        "200":
          description: List of post is retrieved
          content:
            application/json:
              schema: {}
```



The Schema Object

SchemaObject adalah sebuah object yang dibuat berdasarkan JSON Schema untuk mendefinisikan bagaimana bentuk data dari sebuah *object*. Karena atribut ini *compatible* dengan JSON Schema, maka dari itu, ketika kita mendefinisikan sebuah skema di dalam Open API Documentation, kita bisa pasang beberapa tools yang dapat memvalidasi sebuah *object* (parameter) apakah sudah sesuai dengan skema atau belum.

Dokumen di samping adalah simplifikasi dari definisi yang dibutuhkan sebenarnya, but you get the idea, namun kita masih perlu menambahkan bagaimana cara melakukan *request* yang baik dan benar yang mana hal ini akan dibahas pada bagian berikutnya.

```
paths:
  /posts:
    post:
      responses:
        "500":
          description: Cannot create post because of an error on the server
          content:
            application/json:
              schema:
                type: object
                properties:
                  status:
                    type: string
                    example: "ERROR"
                  data:
                    type: object
                    properties:
                      name:
                        type: string
                        example: "INTERNAL_SERVER_ERROR"
                      message:
                        type: string
                        example: "Something is error"
                      stack:
                        type: string
                        example: "app/controllers/postController.js:64"
```



Request Body and Parameter

Setiap kita menjalankan suatu operasi, tentunya ada beberapa operasi yang memerlukan input dari yang meminta operasi tersebut dijalankan, layaknya sebuah *function* di dalam sebuah program, ketika kita memanggil sebuah *function* terkadang *function* tersebut meminta beberapa parameter. Begitu juga dengan sebuah API, setiap operasi di API, terkadang perlu parameter yang harus dikirim oleh *consumer* dari API tersebut.





The Parameter Object

Parameter object adalah **sebuah object yang kita gunakan untuk mendefinisikan parameter yang dapat kita kirim pada sebuah operasi**. Biasanya, object ini digunakan untuk mendefinisikan *path parameter* dan *query parameter* jika dalam sebuah endpoint membutuhkan kedua jenis parameter tersebut.

```
paths:  
  /posts:  
    get:  
      parameters:  
        - in: query  
          name: status  
          description: The status of Posts  
          required: false  
          schema:  
            type: string  
            enum:  
              - DRAFT  
              - PUBLISHED  
        - in: query  
          name: per_page  
          description: How much post will be listed on each page  
          required: false  
          schema:  
            type: integer  
            example: 10  
  /posts/{id}:  
    get:  
      parameters:  
        - in: path  
          name: id  
          description: The status of Posts  
          required: true  
          schema:  
            type: string
```



The Request Body Object

Selain *parameters*, kita juga dapat mendefinisikan sebuah *parameter* lain yaitu *request body*. *Request body* hanya akan dipakai di dalam 3 jenis operasi saja yaitu POST, PUT, dan PATCH. *Object* ini disimpan di dalam atribut yang bernama *requestBody* di dalam sebuah *Operation Object*.

```
paths:  
  /posts:  
    post:  
      requestBody:  
        description: Payload for making a Post  
        required: true  
        content:  
          application/json:  
            schema:  
              type: object  
              properties:  
                title:  
                  type: string  
                  example: Industrial Society  
                body:  
                  type: string  
                  example: Lorem ipsum dolor sit amet.
```



Components

Ada beberapa object yang kita buat berulang-ulang, ketika kita mendefinisikan sebuah *Open API documentation*. Seperti bentuk *response*, *request body*, dan sebagainya. Untuk mengurangi perulangan dalam membuat *object* yang sama, kita dapat menggunakan components, dimana components ini akan diletakkan pada *Open API Object* dengan atribut bernama components. Ada 4 jenis components, yaitu:

- **schemas**
- **requestBodies**
- **responses**
- **securitySchemes**





1. The Schema Object

Schema Object adalah sebuah *object* atributnya ialah jenis *schema* yang dapat kita gunakan ketika kita membutuhkan *schema* di dalam sebuah *operation*.

Dokumen di samping, sudah menggunakan **#/components/schemas/Post**, untuk mendefinisikan skema dari array di dalam.

```
components:  
schemas:  
  Post:  
    type: object  
    properties:  
      id:  
        type: string  
        example: "bdaf6814-2e99-40d5-9c15-e0238a90c886"  
      title:  
        type: string  
        example: Industrial Society and It's Future  
      body:  
        type: string  
        example: Lorem ipsum dolor sit amet, consectetur adipiscing elit.  
      authorId:  
        type: string  
        example: "196508c4-495f-4d2e-898b-776fb3c1e8b4"  
      authorName:  
        type: string  
        example: "Ted Kaczynski"  
      authorAvatar:  
        type: string  
        example: "https://placekitten.com/200/200"  
      createdAt:  
        type: string  
        example: "2021-07-30 17:00:00 +0700"  
      updatedAt:  
        type: string  
        example: "2021-07-30 17:00:00 +0700"
```



2. The Request Bodies Object

Request Bodies Object adalah **sebuah object yang digunakan untuk mendefinisikan sebuah request body yang dapat kita panggil pada sebuah operasi.** Request Bodies juga dapat menggunakan schemas.

```
components:  
  requestBodies:  
    LoginRequest:  
      description: "Payload for Login Request"  
      content:  
        application/json:  
          schema:  
            type: object  
            properties:  
              email:  
                type: string  
                example: sabrina@binar.co.id  
              password:  
                type: string  
                example: awurenwae
```



3. The Response Object

Response Object adalah sebuah **object** yang kita gunakan untuk membuat sebuah definisi **response** dengan berbagai jenis tipe media yang bisa didefinisikan.

```
components:  
responses:  
  LoginResponse:  
    description: "Payload on Successfully Logged In"  
    content:  
      application/json:  
        schema:  
          type: object  
          properties:  
            accessToken:  
              type: string  
            example: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...."
```



4. The Security Schemes Object

Security Schemes Object digunakan untuk mendefinisikan sebuah security yang nantinya akan digunakan di dalam operasi. Untuk detail terkait ini dapat dilihat pada tautan berikut:

<https://swagger.io/docs/specification/authentication/>

Di dalam bagian ini, kita akan membuat sebuah securitySchemes yang memiliki Bearer Auth di dalamnya.

Ketika kita memasang security di dalam sebuah *operation*, maka operasi tersebut akan membutuhkan sebuah access yang memadai, seperti *JWT* atau apapun yang digunakan oleh API untuk mengautentikasi sebuah *request*.

```
● ● ●

paths:
  "/api/v1/whoami":
    get:
      security:
        - bearerToken: []
components:
  securitySchemes:
    bearerToken:
      name: "Authorization"
      type: http
      scheme: "bearer"
```



API Servers

Di dalam dokumentasi Open API, kita dapat mendefinisikan apa saja sih server yang menjalankan API tersebut. Untuk mendefinisikannya, kita dapat menggunakan atribut bernama servers, di dalam *Open API object*.



```
openapi: 3.0.0
info:
  title: Medium Blogging API
  version: 0.0.1
servers:
  - description: Mock Server
    url: https://mock.medium.com
  - description: "Staging Server"
    url: https://staging.medium.com
  - description: "Beta Server"
    url: https://beta.medium.com
  - description: "Production Server"
    url: https://medium.com
paths: {}
```



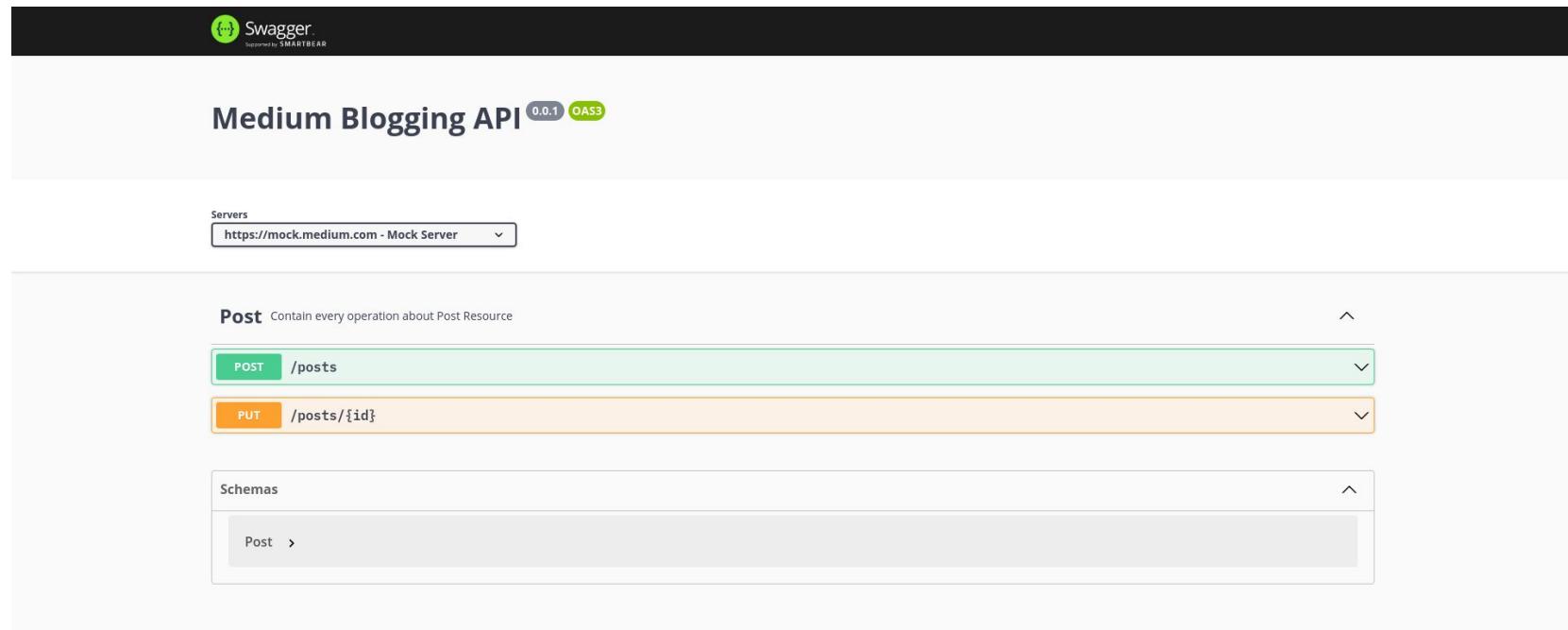
Banyak banget ya tadi struktur dan kriteria Open API. Teorinya udah oke kan? Biar makin paham, langsung gasss yuk ke praktiknya buat mengimplementasi Open API documentation~





Menampilkan Open API Documentation

Berikut ini adalah sebuah Open API Specification yang akan kita gunakan untuk menampilkan sebuah UI dari Open API Specification kita. Kalian bisa mendownload [Open API Specs ini](#) sebagai contoh. **Swagger UI**



The screenshot shows the Swagger UI interface displaying the "Medium Blogging API". At the top, there's a header with the Swagger logo and "Supported by SMARTBEAR". Below the header, the API title is "Medium Blogging API" with version "0.0.1" and "OAS3" status. A "Servers" dropdown menu is set to "https://mock.medium.com - Mock Server". The main content area shows the "Post" resource, which contains operations for creating and updating posts. The "POST /posts" operation is highlighted in green, while the "PUT /posts/{id}" operation is shown in orange. Below the operations, there's a "Schemas" section with a "Post" entry. The entire interface has a clean, modern design with a light gray background and distinct color coding for different sections.



Untuk menampilkan dokumentasi Open API ini sangat tergantung terhadap bahasa yang digunakan dari sebuah server. Untuk express, kita dapat menggunakan library **swagger-ui-express**.

Kode di samping membutuhkan file bernama `swagger.json`, ini adalah nama dari file Open API Documentation yang sudah kalian buat, namun dalam format JSON. Kalau kalian ingin menggunakan versi YAML, maka ada beberapa library yang perlu kita instal agar kita dapat menggunakan file YAML kita.



```
const express = require('express');
const swaggerUi = require('swagger-ui-express');
const app = express();
const swaggerDocument = require('./swagger.json');

app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument));
```



Maka dokumentasi API-mu akan dapat diakses melalui server express pada path **/api-docs**.

```
const express = require('express');
const YAML = require('yamljs');
const swaggerUi = require('swagger-ui-express');

const app = express();
const swaggerDocument = YAML.load('./openapi.yaml');
const port = 3000;

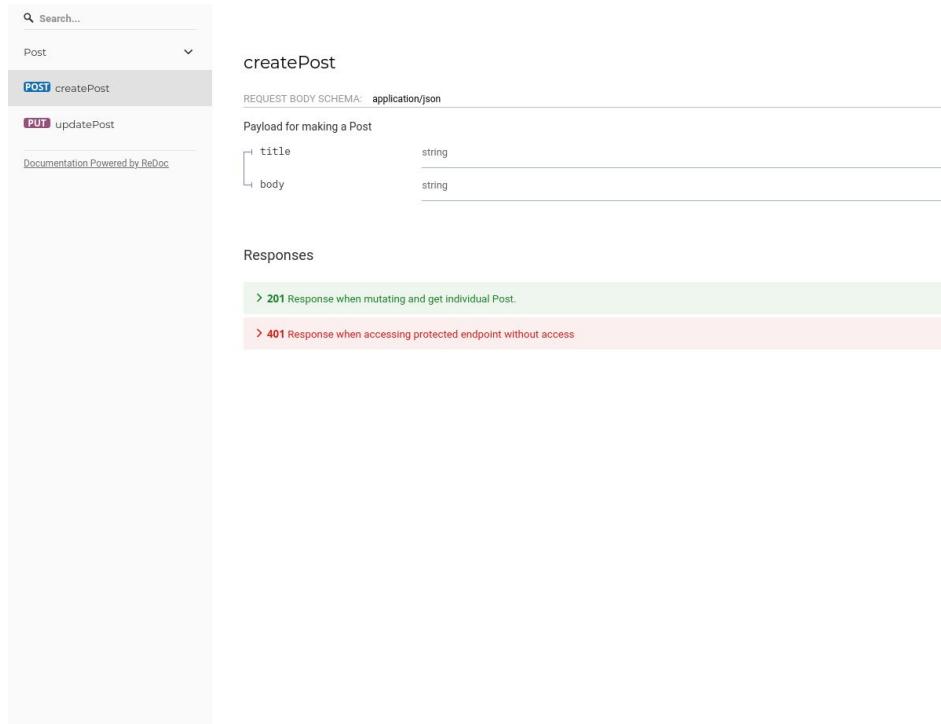
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument));

app.listen(port, () => console.log(`Example app listening on port ${port}!`));
```



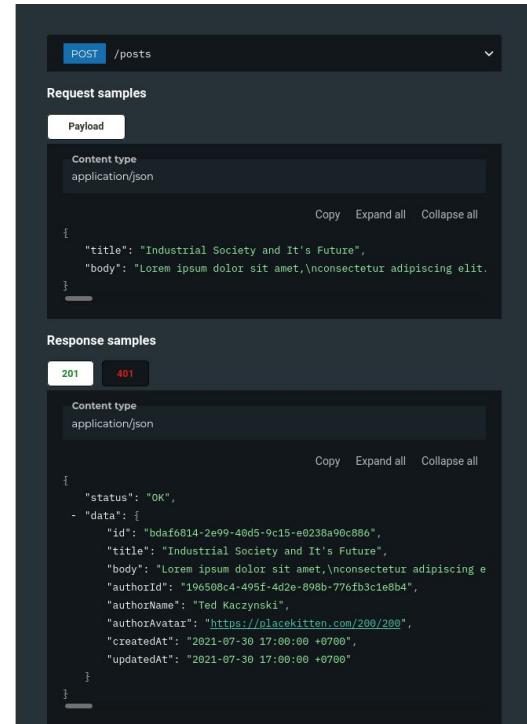
Redoc

Untuk menggunakan ReDoc di dalam express, kita hanya perlu menginstal library bernama [redoc-express](#). Dan cara menggunakannya cukup sederhana.



The screenshot shows the ReDoc interface for a 'Post' resource. On the left, there's a sidebar with a search bar and dropdown menus for 'Post' and 'createPost'. Below it are links for 'createPost' (highlighted in blue) and 'updatePost'. A note says 'Documentation Powered by ReDoc'.

The main area is titled 'createPost'. It shows a 'REQUEST BODY SCHEMA: application/json'. Below this is a 'Payload for making a Post' section with two fields: 'title' (string) and 'body' (string). Underneath is a 'Responses' section with two items: a green box for '201 Response when mutating and get individual Post.' and a pink box for '401 Response when accessing protected endpoint without access'.



The screenshot shows the ReDoc interface for a 'posts' endpoint. At the top, it says 'POST /posts'. Below this is a 'Request samples' section with a 'Payload' tab selected. It shows a 'Content type: application/json' field with a JSON payload: { "title": "Industrial Society and It's Future", "body": "Lorem ipsum dolor sit amet,\nconsectetur adipiscing elit."}. There are 'Copy', 'Expand all', and 'Collapse all' buttons.

Below this is a 'Response samples' section with tabs for '201' and '401'. The '201' tab is selected, showing a 'Content type: application/json' field with a JSON response: { "status": "OK", "data": { "id": "bdaf6814-2e99-40d5-9c15-e0238a90c886", "title": "Industrial Society and It's Future", "body": "Lorem ipsum dolor sit amet,\nconsectetur adipiscing elit.", "authorID": "19e508c4-495f-4d2e-898b-776fb3c1e8b4", "authorName": "Ted Kaczynski", "authorAvatar": "https://blacekitten.com/200/200", "createdAt": "2021-07-30 17:00:00 +0700", "updatedAt": "2021-07-30 17:00:00 +0700" } }. There are also 'Copy', 'Expand all', and 'Collapse all' buttons.



Ketika kalian membuka halaman /docs, maka kalian akan melihat dokumentasi API kalian dengan UI dari Redoc.

```
const express = require('express');
const redoc = require('redoc-express');
const YAML = require('yamljs')
const openAPIDocument = YAML.load('./openapi.yaml')

const app = express();
const port = 3000;

// serve your swagger.json file
app.get('/docs/swagger.json', (req, res) => {
    res.status(200).json(openAPIDocument)
});

// define title and specUrl location
// serve redoc
app.get(
    '/docs',
    redoc({
        title: 'API Docs',
        specUrl: '/docs/swagger.json'
    })
);

app.listen(port, () => console.log(`Example app listening on port ${port}!`));
```



Membuat Mock API untuk Early Development

Untuk membuat Mock API sangatlah mudah, kalian hanya perlu:

1. Buka editor.swagger.io
2. Paste Open API Documentationmu
3. Generate Server
4. Pilih bahasa nodejs-server
5. Download
6. Extract
7. npm start

Server yang kamu jalankan akan dapat memenuhi kebutuhan kontrak dari API yang sudah kamu definisikan, hal ini sangat berguna bagi developer frontend, karena mereka sudah bisa menggunakan Mock API kita untuk mengimplementasikan integrasi ke API.





Open API Documentation sangat berguna dalam proses development, kita dapat membuat Mock API secara otomatis dan efisien. Open API Documentation dapat menggantikan kebutuhan adanya API di dalam proses pengembangan website.

Dan juga, **Open API documentation sangat berguna untuk mendokumentasikan setiap operasi yang dapat kita lakukan dengan menggunakan API yang kita dokumentasikan.**



Referensi dan bacaan lebih lanjut~

- [Open API Initiative](#)
- [Swagger](#)





Nah, selesai sudah pembahasan kita di Chapter 6 Topic 4 ini.

Setelah ini, kita akan lanjut ke **Chapter 7**.

See youuu~



Terima Kasih!



Chapter ✓

completed