

API Documentation with Swagger

Gold - Chapter 5 - Topic 4

Selamat datang di **Chapter 5 Topic 4**
online course **Back End Java** dari
Binar Academy!



Binarian, gimana kabar kamu? □

Pada topic ketiga kamu udah belajar tentang Design Pattern. Pada topik selanjutnya ini, kita bakal mengelaborasi tentang **API Documentation with Swagger**. Mulai dari konsep API, cara membuat API contract, sampai cara mengimplementasikannya.

Yuk, langsung aja kita kepoin~



Dari sesi ini, kita bakal bahas hal-hal berikut:

- Konsep Open API
- Bagaimana cara membuat API Contract
- Penerapan Swagger pada aplikasi



Pada chapter 4 kita udah belajar cara bikin Rest API. Kali ini kita belajar bikin dokumentasi API.

Tapi, kenapa dokumentasi API perlu dibikin?

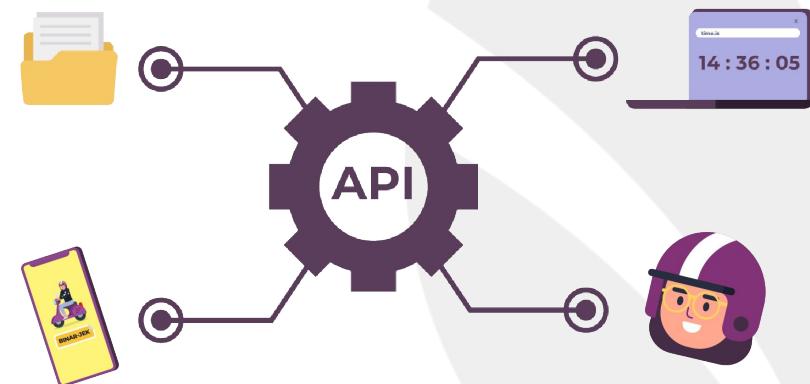
Arti pentingnya bakal kamu temukan setelah slide ini, yuk kita cari bareng!



Pertama, kita perlu akui dulu nih, kalau API itu ternyata keren banget!

Iya, soalnya API bisa melakukan beberapa hal yang berguna banget, kayak:

- Menyimpan, mengubah, dan mengakses data.
- Mengunggah data ke media sosial secara otomatis.
- Menampilkan data secara realtime.
- Mengintegrasikan aplikasi dengan berbagai layanan.



Dibalik itu, ternyata API sengaja dirancang hanya untuk menjadi mak comblang alias **jembanan antar sistem aja**. Bukan jembatan antara manusia dengan sistem.

Oleh karena itu, tampilan visual dari API cuma dirancang untuk menerima **HTTP request**, lalu menampilkan **HTTP response** yang berupa data dalam format **JSON**.

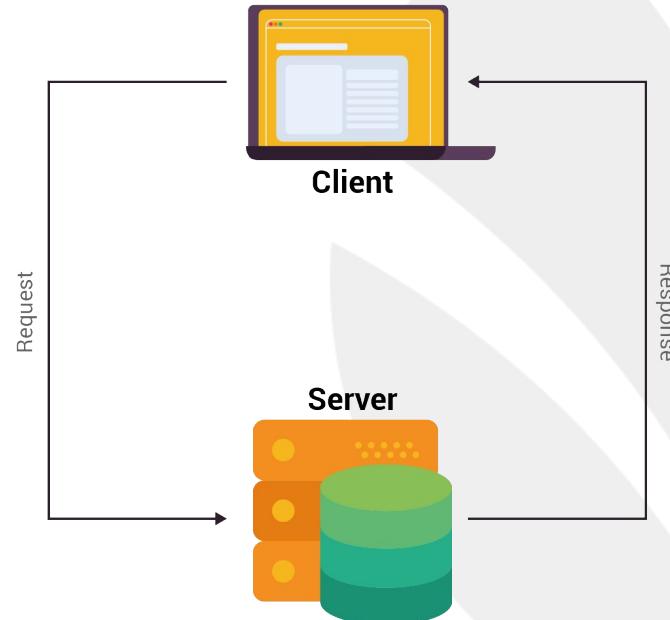


“Kalau cuma jadi jembatan, cara API menyampaikan pesannya gimana dong?”

Request dan response lewat API selalu disampaikan dalam bentuk teks.

Jadi, nggak ada petunjuk visual intuitif bagi kita, para pengguna API.

Tentunya hal ini bakal bikin kita jadi repot banget, apalagi kalau kita nggak berhubungan langsung sama developer API-nya.



Para pengguna API mungkin bakal banyak bertanya deh tentang cara kerjanya ☐

Misalnya pertanyaan kayak gini, nih:

- Apa endpoint yang harus aku akses?
- Parameter apa yang bisa aku pakai?
- Apakah ada lapisan authorization? Gimana cara buat dapat aksesnya?
- Aku melakukan request tapi response yang muncul adalah 400 (bad request), kenapa?

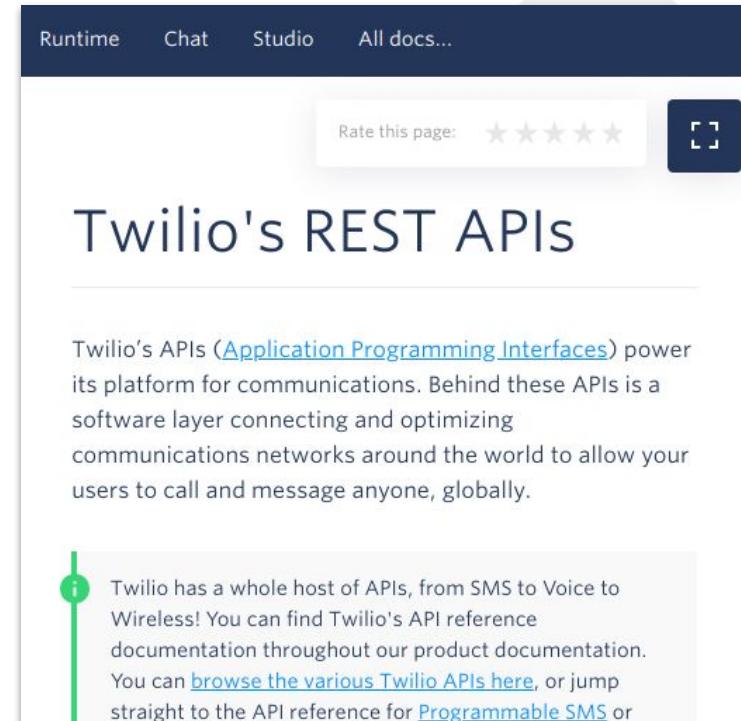
Buat jawab pertanyaan tersebut, maka dibuatlah **dokumentasi API**~



“Dokumentasi API? Maksudnya?”

Dokumentasi API adalah sebuah dokumen yang isinya informasi suatu API, kayak:

- Apa aja Endpoint-nya?
- Apa fungsinya? dan
- Gimana cara pakai-nya?



The screenshot shows a dark-themed API documentation interface. At the top, there are tabs for "Runtime", "Chat", "Studio", and "All docs...". Below the tabs, there's a rating section with a placeholder "Rate this page: ★★★★★" and a "Feedback" button. The main title "Twilio's REST APIs" is displayed prominently. A detailed description follows: "Twilio's APIs ([Application Programming Interfaces](#)) power its platform for communications. Behind these APIs is a software layer connecting and optimizing communications networks around the world to allow your users to call and message anyone, globally." A callout box on the right contains an information icon and text about Twilio's comprehensive API offerings, mentioning SMS, Voice, and Wireless, along with links to browse various APIs and Programmable SMS documentation.

Twilio's APIs ([Application Programming Interfaces](#)) power its platform for communications. Behind these APIs is a software layer connecting and optimizing communications networks around the world to allow your users to call and message anyone, globally.

Twilio has a whole host of APIs, from SMS to Voice to Wireless! You can find Twilio's API reference documentation throughout our product documentation. You can [browse the various Twilio APIs here](#), or jump straight to the API reference for [Programmable SMS](#) or

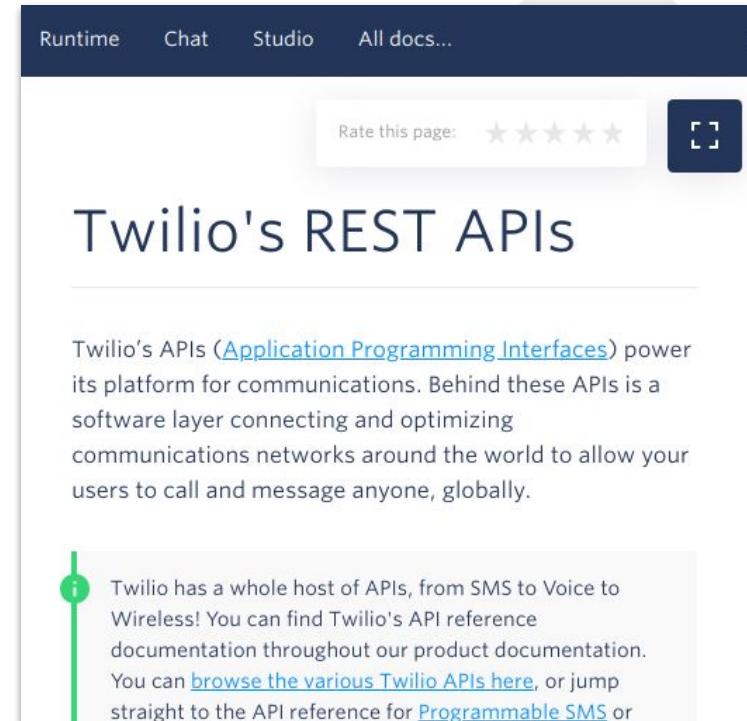
Setelah tahu informasi apa yang disampaikan, berikut adalah contoh dokumentasi API:

- **Twilio Whatsapp API**

[The WhatsApp Business API with Twilio](#)

- **Google Maps API**

[Maps Javascript API](#)



A screenshot of a web page titled "Twilio's REST APIs". The page has a dark blue header with navigation links: "Runtime", "Chat", "Studio", and "All docs...". Below the header is a rating section with a 5-star rating and a "Rate this page" button. The main title "Twilio's REST APIs" is centered above a detailed description of the service. The description states: "Twilio's APIs ([Application Programming Interfaces](#)) power its platform for communications. Behind these APIs is a software layer connecting and optimizing communications networks around the world to allow your users to call and message anyone, globally." At the bottom of the page, there is a callout box with a green border and a green info icon. It contains the text: "Twilio has a whole host of APIs, from SMS to Voice to Wireless! You can find Twilio's API reference documentation throughout our product documentation. You can [browse the various Twilio APIs here](#), or jump straight to the API reference for [Programmable SMS](#) or".

Setelah paham dokumentasi API dan manfaatnya, kita perlu tahu nih struktur dan kriteria dokumentasi API itu apa aja, supaya kita bisa mendefinisikan dokumentasi yang baik itu kayak gimana.

Beli es naik angkot. Let's Cekidot!



Kira-kira struktur dokumentasi API kayak gimana yaaa?

Dokumentasi API ini punya struktur dan kriteria khusus yang bisa kita mengerti sebagai pengguna API.

“Terus strukturnya kayak gimana, dong?”

Struktur dan kriterianya bisa kita simak di slide selanjutnya yaaa~



Berikut adalah struktur dan kriteria dokumentasi API!

- Apakah di dalam dokumentasi tersebut ada **Quick Start Guide**?
- Apakah di dalam dokumentasi tersebut ada **Authentication Guide**?



- Apakah dokumentasi tersebut berisi **Endpoint Definition**?
- Apakah di dalam dokumentasi tersebut ada **Example Response**?
- Apakah di dalam dokumentasi tersebut ada **Code Snippet**?



Well.. kalau kamu penasaran sama istilah baru di pertanyaan tadi, abis ini kita akan langsung jelaskan definisinya.

1. Quick start guide
2. Authentication guide
3. Endpoint definition
4. Example reasons
5. Code snippet

Let's gooo!

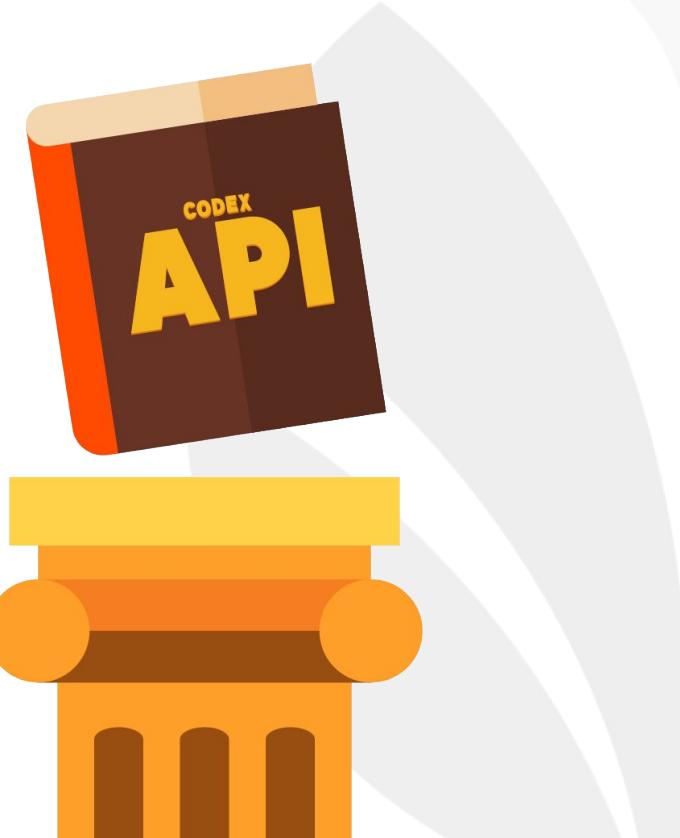


Quick Start Guide

Quick start guide adalah **panduan bagi pengguna API** untuk memulai menggunakan API.

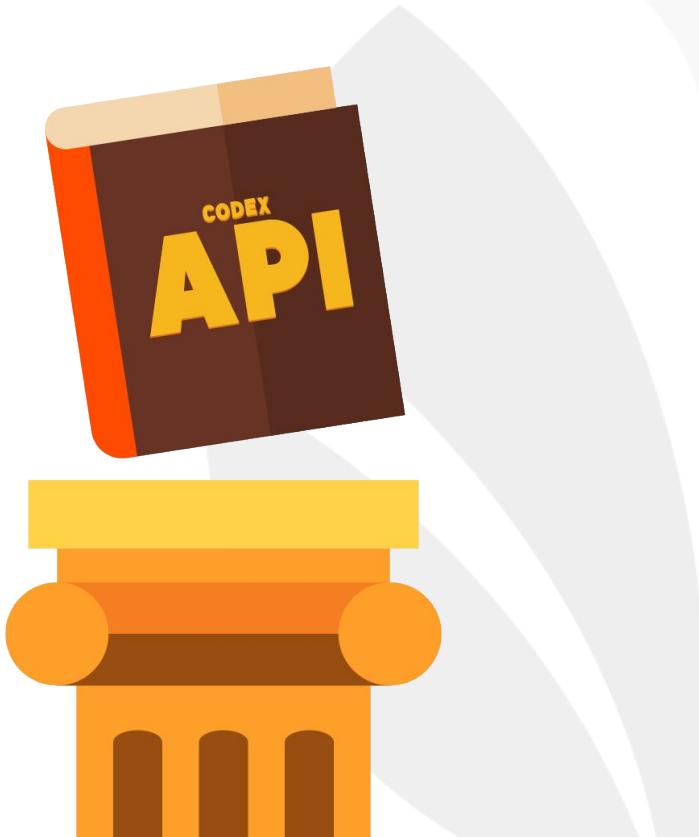
Bagian ini cukup berisiko karena ia harus straightforward untuk bisa memandu developer yang berpengalaman.

Selain itu, guide harus cukup jelas buat memandu developer pemula.



Idealnya, kita bisa konsumsi suatu API ketika udah baca Quick Start Guide ini. Panduannya sendiri berisi tentang:

- Informasi API, baik itu fitur, nama pembuat, hak akses, batasan, disclaimer, dll.
- Cara memahami dokumentasi secara keseluruhan.
- Cara memperoleh credential buat akses API.
- Contoh endpoint yang bisa dicoba secara cepat.

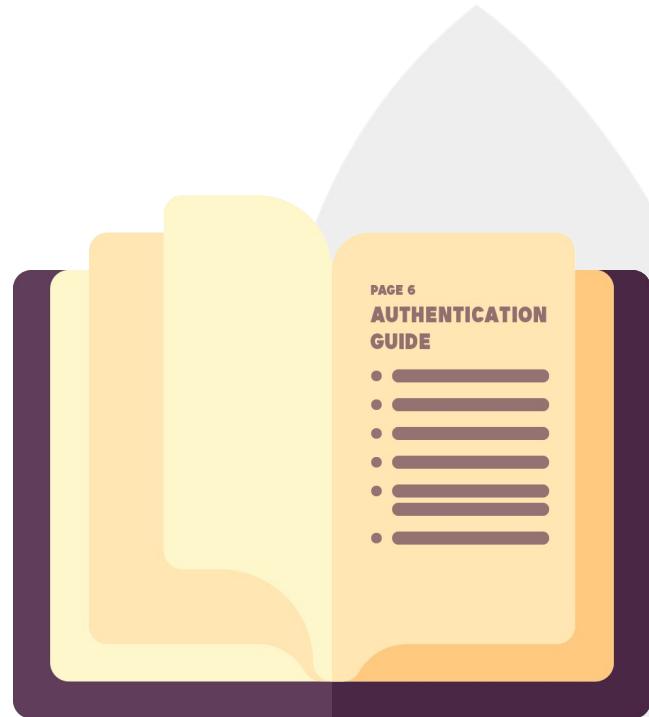


Authentication Guide

Authentication Guide ini berisi tentang gimana kita **mengakses** request pada protected endpoint.

Contohnya, kalau di REST API pakai Token-Based Authentication, berarti di Authentication Guide biasanya berisi tentang gimana cara mendapatkan token dan di mana Client-Side harus mencantumkan token tersebut.

Pada Authentication Guide juga bakal memberi tahu tanda-tanda atau simbol di **Endpoint Definition** yang menandakan bahwa endpoint membutuhkan token.



Endpoint Definition

Di bagian ini endpoint didefinisikan untuk mengakses API melalui **HTTP Request**. Lebih jelasnya, bagian ini meliputi:

HTTP method	Metode HTTP request (GET, POST, PUT, PATCH, HEAD, DELETE, dll)
URL	Alamat web yang dituju
URL parameter	Parameter yang melekat di URL sebagai input
Query parameter	Parameter dalam format key-value yang melekat di URL sebagai input
Request header	Key-value yang disertakan ke dalam header, wajib maupun optional
Request body	Key-value yang disertakan ke dalam body, wajib maupun optional

Berikut adalah contoh dokumentasi API dari **covid19api.com** dengan ringkasan harian kasus Covid-19 memiliki detail sebagai berikut:

- HTTP method: GET
- URL : <https://api.covid19api.com/summary>
- URL parameter : -
- Query parameter : -
- Request header : X-Access-Token
- Request body : -

GET Summary

`https://api.covid19api.com/summary`

A summary of new and total cases per country updated daily.

HEADERS

X-Access-Token

Example Response

Pada bagian ini didefinisikan format **HTTP Response** dari suatu endpoint. Bagian ini berguna banget untuk developer yang membutuhkan nama field secara detail. Bagian ini juga berguna untuk mendefinisikan error handling.

Lebih jelasnya, coba simak tabel berikut yaaaa~

Response Header	Informasi mengenai response
Response Body	Data yang diberikan server
Status Code	Status response, mengikuti standar HTTP status code

Berikut adalah contoh response dari dokumentasi API.

Bisa kamu cek di [Summary API - Covid19](#) juga, ya!

Code Snippet

Yaitu bagian optional untuk **mengimplementasikan request** lewat command line tools atau library HTTP Request.

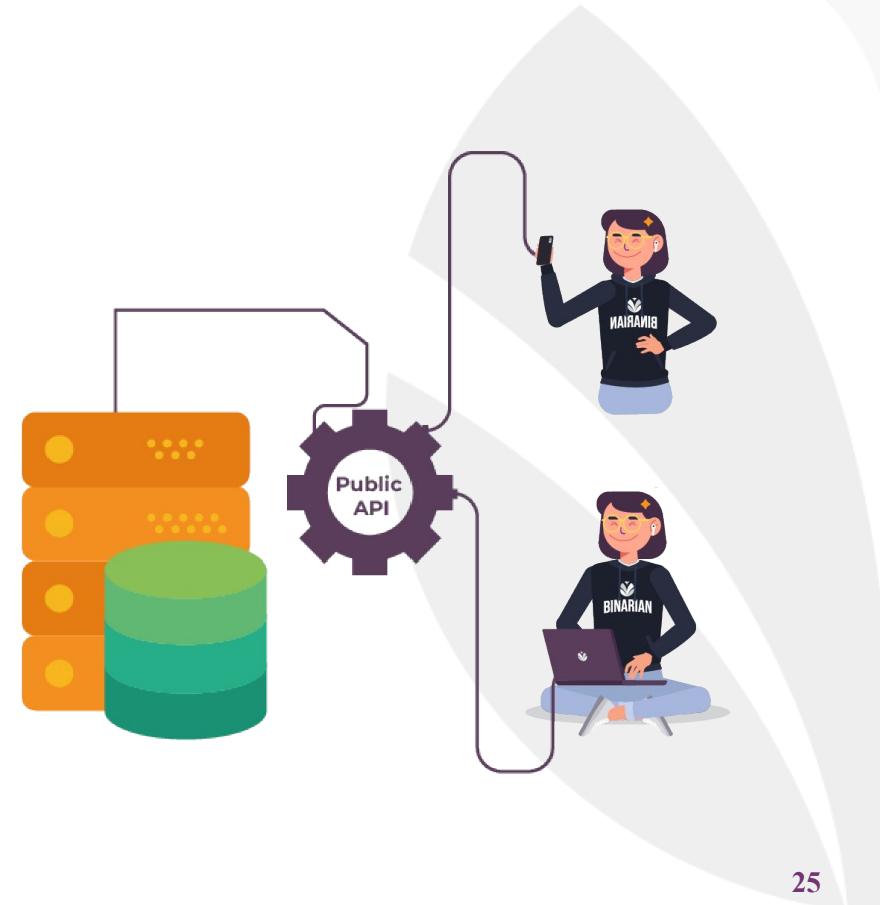
Berikut adalah contoh dengan command line tools (curl). Perhatikan baik-baik ya, Sobat. □

```
curl -X POST \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer <YOUR_ACCESS_TOKEN>' \
  --data '{"email": "mail@example.com", "password": "123456"}' \
  https://api.example.com/v1/auth/login
```

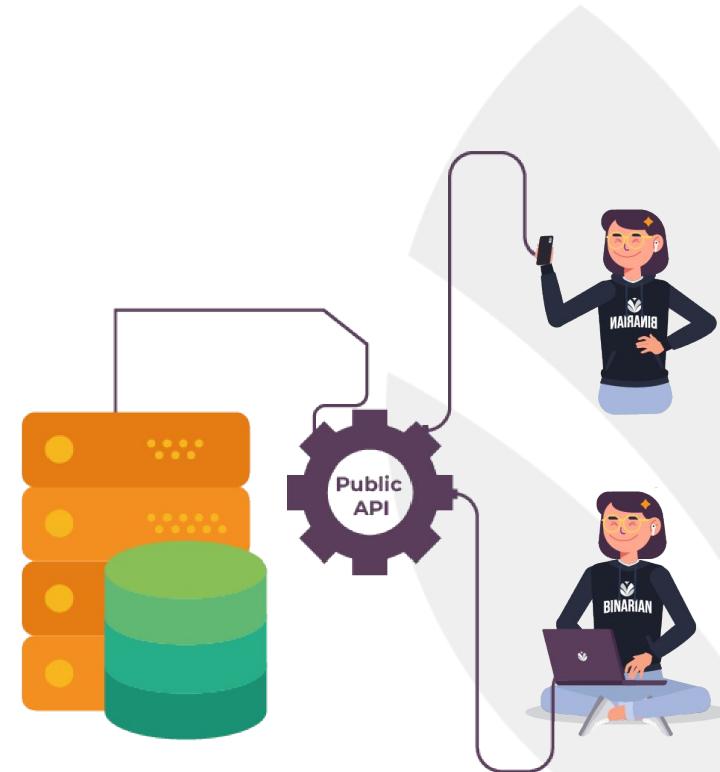
Ternyata ada API yang bisa diakses sama orang atau aplikasi lain, lho!

API yang bisa diakses oleh orang atau aplikasi lain disebut sebagai Public API atau Open API.

Open API ini bisa digunakan supaya developer lain mengerti cara menggunakan API yang sebelumnya sudah di dokumentasikan.



Selain melakukan dokumentasi API dengan Postman, ternyata ada interface yang lebih mudah dan paling sering dipakai buat bikin suatu dokumentasi Open API lho, yaitu Swagger.



Sebelum masuk ke cara bikinnya, kita harus tahu dulu nih konsep dari Swagger-nya sendiri, gengs~

Swagger adalah salah satu tools yang mampu untuk mendokumentasikan API.

“Kenapa harus menggunakan tools ini?”



SWAGGER

Ini dia alasannya, bestie~

Karena kalau pakai Swagger, kita bisa **mengintegrasikan UI-nya ke server kita langsung**. Misalnya ke endpoint atau documentation.

Selain bisa dipakai untuk mendokumentasikan API, Swagger juga bisa kita pakai untuk meng-generate Mock API juga, lho!

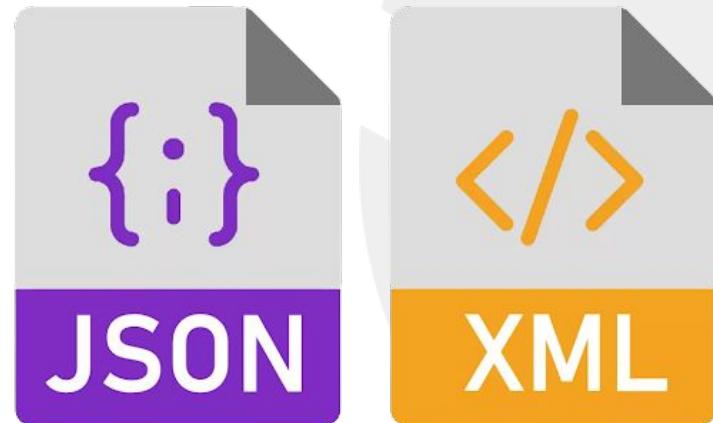
Eits, tapi kita nggak bakal bahas di sini yaa.



Tampilan dari dokumentasi Swagger disebut dengan **Swagger UI**.

Untuk menampilkan Swagger UI yang sesuai dengan ekspektasi kita, Swagger UI bakal baca file **.json** atau **.yml** yang isinya properti-properti yang dibutuhkan Swagger UI untuk lalu dijadikan tampilan di web.

Keren banget, kan?!



Di dalam API, ada beberapa endpoint lho, gengs!

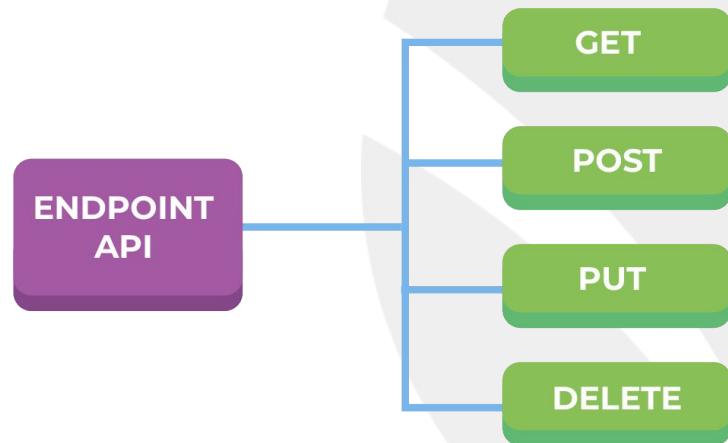
Berikut adalah endpoint-nya:

- **GET** /articles

Response-nya Object Article yang berupa JSON.

- **POST** /articles

Perlu title dan body di dalam request body. Selain itu, response-nya Object Article yang berupa JSON.

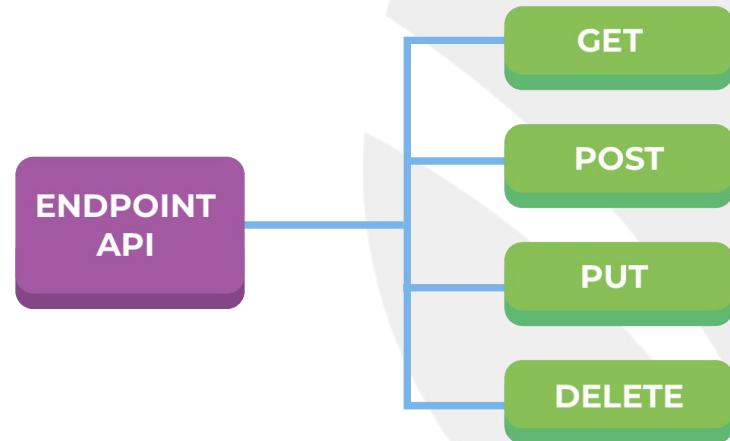


- **PUT** /articles/:id

Perlu title dan body di dalam request body. Selain itu, response-nya Object Article yang berupa JSON.

- **DELETE** /articles/:id

Response-nya berupa message "Artikel berhasil di delete".



Untuk membuat dokumentasi Swagger secara interaktif, kita bisa pakai Swagger Editor, lho!

Serius deh. Nggak bohong!

Swagger Editor adalah sebuah text editor. Jadi, kita bisa mengedit file **.json** atau **.yml** yang nantinya bakal dikonsumsi oleh editor untuk dijadikan preview dari tampilan dokumentasinya.

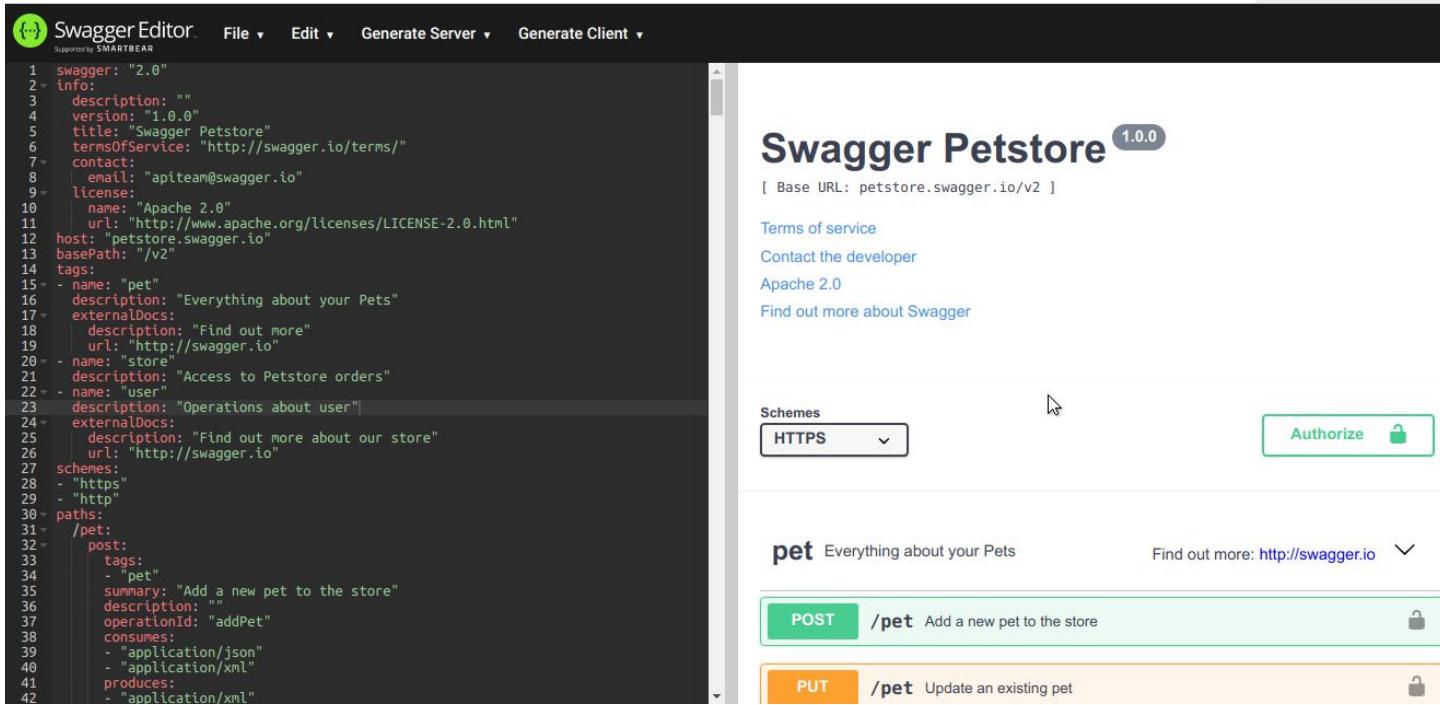


Isi filenya bakal dikonversi sama Swagger jadi halaman HTML yang menampilkan informasi dokumentasi API.

Yuk buka link [Swagger Editor](#) buat meluncur ke lokasi~



Berikut adalah tampilan Swagger Editor, gengs!



The screenshot shows the Swagger Editor interface on the left and the generated API documentation on the right.

Swagger Editor (Left):

```
1 swagger: "2.0"
2   info:
3     description: ""
4     version: "1.0.0"
5     title: "Swagger Petstore"
6     termsOfService: "http://swagger.io/terms/"
7     contact:
8       email: "apiteam@swagger.io"
9     license:
10    name: "Apache 2.0"
11    url: "http://www.apache.org/licenses/LICENSE-2.0.html"
12  host: "petstore.swagger.io"
13  basePath: "/v2"
14  tags:
15    - name: "pet"
16      description: "Everything about your Pets"
17    externalDocs:
18      description: "Find out more"
19      url: "http://swagger.io"
20    - name: "store"
21      description: "Access to Petstore orders"
22    - name: "user"
23      description: "Operations about user"
24    externalDocs:
25      description: "Find out more about our store"
26      url: "http://swagger.io"
27  schemes:
28    - "https"
29    - "http"
30  paths:
31    /pet:
32      post:
33        tags:
34          - "pet"
35          summary: "Add a new pet to the store"
36          description: ""
37          operationId: "addPet"
38          consumes:
39            - "application/json"
40            - "application/xml"
41          produces:
42            - "application/xml"
```

Swagger Petstore (Right):

1.0.0 [Base URL: petstore.swagger.io/v2]

Terms of service
Contact the developer
Apache 2.0
Find out more about Swagger

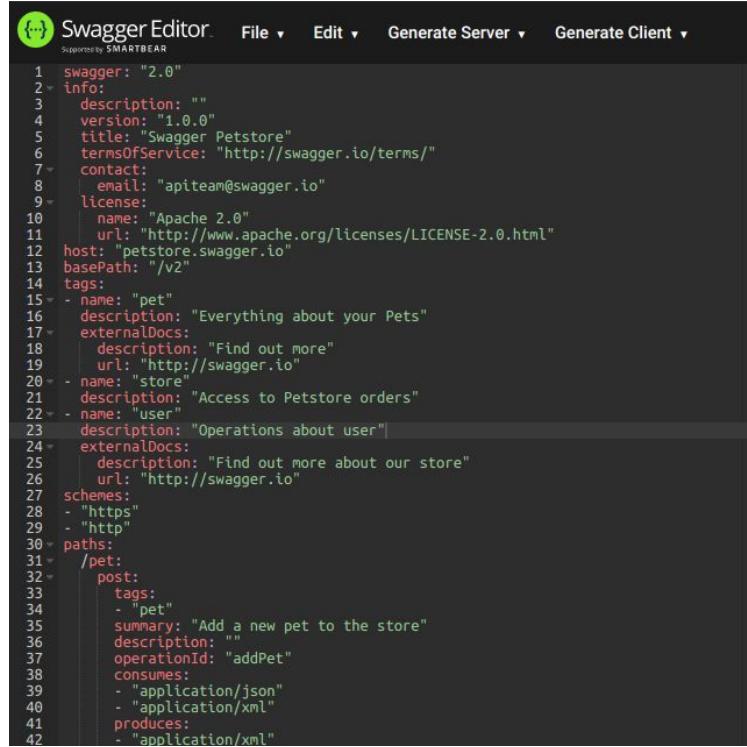
Schemes: HTTPS ▾ Authorize 🔒

pet Everything about your Pets Find out more: <http://swagger.io> ▾

POST /pet Add a new pet to the store 🔒

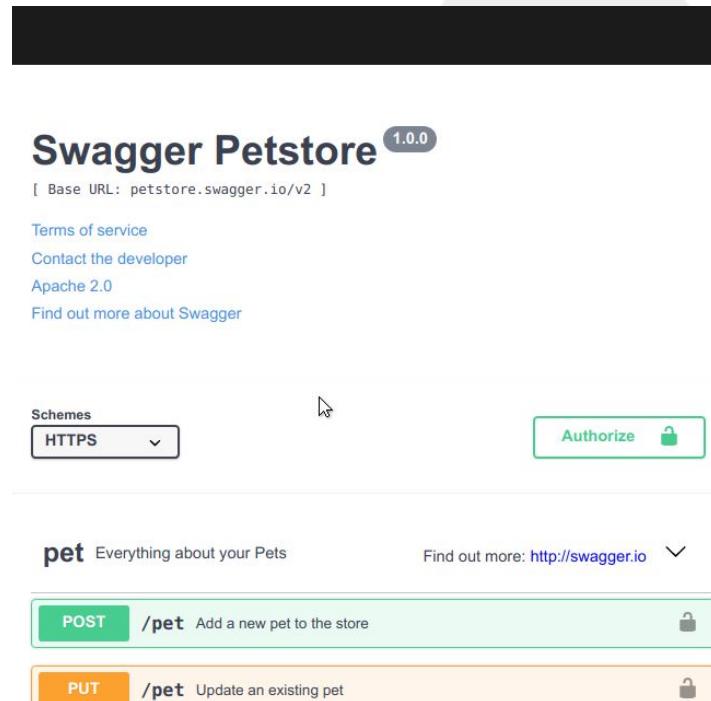
PUT /pet Update an existing pet 🔒

Kolom ini menunjukkan definisi dokumentasi. Coba perhatikan baik-baik ya, bestie~

A screenshot of the Swagger Editor interface. The title bar says "Swagger Editor" and "Supported by SMARTBEAR". The main area shows a large block of JSON code representing an API definition. The code includes sections for info, contact, license, tags, paths, and schemes. The "paths" section contains a "post" operation for the "/pet" endpoint, which has a summary of "Add a new pet to the store", an operationId of "addPet", consumes "application/json" and "application/xml", produces "application/xml", and a tag of "pet".

```
1  swagger: "2.0"
2  info:
3    description: ""
4    version: "1.0.0"
5    title: "Swagger Petstore"
6    termsOfService: "http://swagger.io/terms/"
7    contact:
8      email: "apiteam@swagger.io"
9    license:
10      name: "Apache 2.0"
11      url: "http://www.apache.org/licenses/LICENSE-2.0.html"
12    host: "petstore.swagger.io"
13    basePath: "/v2"
14    tags:
15      - name: "pet"
16        description: "Everything about your Pets"
17        externalDocs:
18          description: "Find out more"
19          url: "http://swagger.io"
20      - name: "store"
21        description: "Access to Petstore orders"
22      - name: "user"
23        description: "Operations about user"
24        externalDocs:
25          description: "Find out more about our store"
26          url: "http://swagger.io"
27    schemes:
28      - "https"
29      - "http"
30    paths:
31      /pet:
32        post:
33          tags:
34            - "pet"
35          summary: "Add a new pet to the store"
36          description: ""
37          operationId: "addPet"
38          consumes:
39            - "application/json"
40            - "application/xml"
41          produces:
42            - "application/xml"
```

Kalau kolom yang ini menunjukkan tampilan UI yang otomatis dihasilkan Swagger.

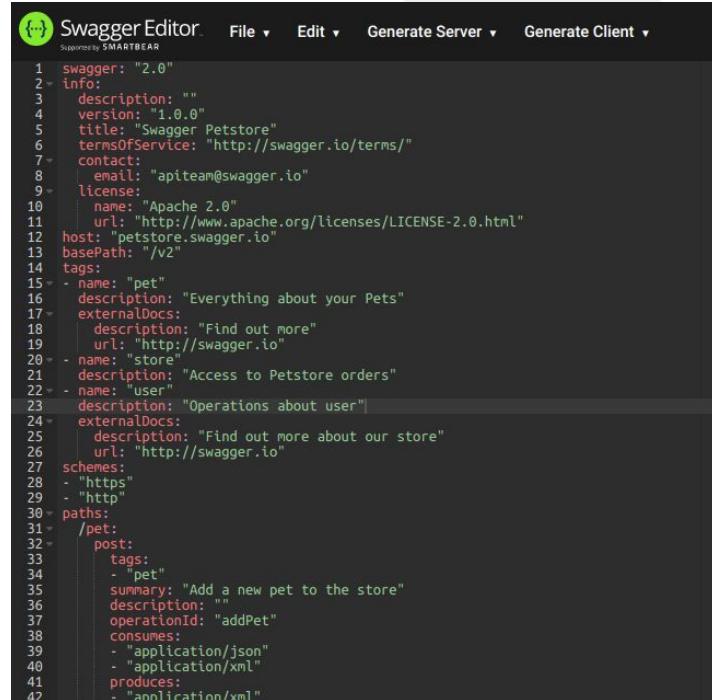


The screenshot shows the Swagger Petstore API documentation. At the top right, there's a dark red rectangular button with white text. Below it, the title "Swagger Petstore" is displayed in a large, bold, dark blue font, with a "1.0.0" badge next to it. A note "[Base URL: petstore.swagger.io/v2]" is shown in a smaller font. To the right of the title are links for "Terms of service", "Contact the developer", "Apache 2.0", and "Find out more about Swagger". On the left, there's a dropdown menu labeled "Schemes" with "HTTPS" selected. On the right, there's a green "Authorize" button with a lock icon. The main content area shows the "pet" resource with two operations: a green "POST /pet" button for adding a new pet to the store, and an orange "PUT /pet" button for updating an existing pet. Both buttons have a lock icon to their right. The "pet" resource is described as "Everything about your Pets".

Kalau kita perhatikan pada gambar, ada file **.yml** yang punya properti terdefinisi, lho~

Properti tersebut antara lain:

- **Swagger**
- **Info**
- **Host**
- **basePath**
- **Tags**
- **Paths**
- **securityDefinition**

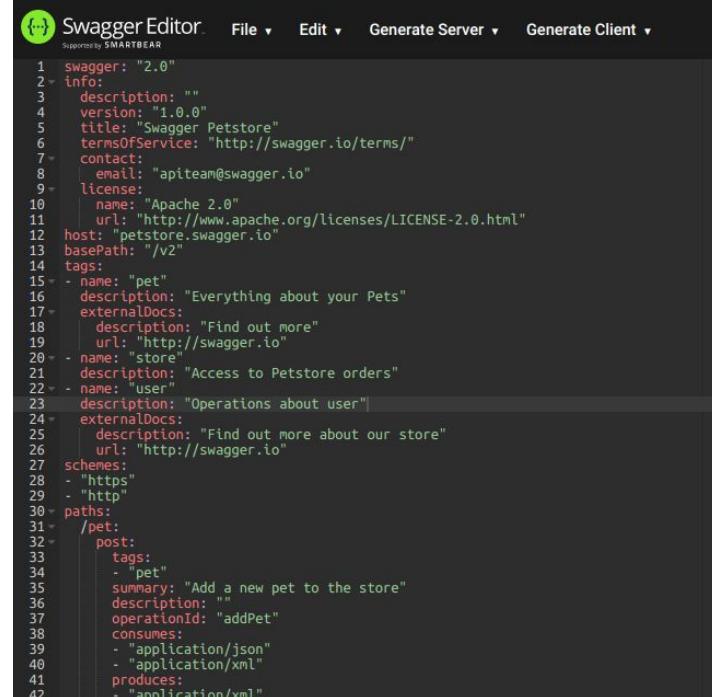


The screenshot shows the Swagger Editor interface with a dark theme. At the top, it says "Swagger Editor" and "Supported by SMARTBEAR". Below that is a navigation bar with "File", "Edit", "Generate Server", and "Generate Client". The main area displays a block of YAML code representing an API definition. The code includes sections for info, host, basePath, paths, and securityDefinitions. The "paths" section contains a "post" operation for the "/pet" endpoint, which has a summary of "Add a new pet to the store" and consumes "application/json" and produces "application/xml".

```
1  swagger: "2.0"
2  info:
3    description: ""
4    version: "1.0.0"
5    title: "Swagger Petstore"
6    termsOfService: "http://swagger.io/terms/"
7    contact:
8      email: "apiteam@swagger.io"
9    license:
10   name: "Apache 2.0"
11   url: "http://www.apache.org/licenses/LICENSE-2.0.html"
12  host: "petstore.swagger.io"
13  basePath: "/v2"
14  tags:
15    - name: "pet"
16      description: "Everything about your Pets"
17      externalDocs:
18        description: "Find out more"
19        url: "http://swagger.io"
20    - name: "store"
21      description: "Access to Petstore orders"
22    - name: "user"
23      description: "Operations about user"
24      externalDocs:
25        description: "Find out more about our store"
26        url: "http://swagger.io"
27  schemes:
28    - "https"
29    - "http"
30  paths:
31    /pet:
32      post:
33        tags:
34        - "pet"
35        summary: "Add a new pet to the store"
36        description: ""
37        operationId: "addPet"
38        consumes:
39        - "application/json"
40        - "application/xml"
41        produces:
42        - "application/xml"
```

Selain itu, dari gambar di samping, kita bisa tahu kalau indentasi dan spacing penting banget dalam file .yml.

Jadi, pastikan indentasinya udah tepat, ya!

A screenshot of the Swagger Editor interface. The title bar says "Swagger Editor" and "Supported by SMARTBEAR". The main area shows a block of YAML code representing an API specification. The code defines an info object with version 1.0.0, title "Swagger Petstore", and terms of service. It includes contact information, a license (Apache 2.0), and two tags: "pet" and "store". The "pet" tag has a description and an external doc pointing to a URL. The "store" tag also has a description and an external doc. A schemes section lists "https" and "http". The paths section starts with a "/pet" endpoint, which has a post operation. This operation has a summary ("Add a new pet to the store"), a description, an operation ID ("addPet"), and consumes ("application/json" and "application/xml"). It also produces ("application/xml").

```
1 swagger: "2.0"
2   info:
3     description: ""
4     version: "1.0.0"
5     title: "Swagger Petstore"
6     termsOfService: "http://swagger.io/terms/"
7     contact:
8       email: "apiteam@swagger.io"
9     license:
10       name: "Apache 2.0"
11       url: "http://www.apache.org/licenses/LICENSE-2.0.html"
12     host: "petstore.swagger.io"
13     basePath: "/v2"
14   tags:
15     - name: "pet"
16       description: "Everything about your Pets"
17       externalDocs:
18         description: "Find out more"
19         url: "http://swagger.io"
20     - name: "store"
21       description: "Access to Petstore orders"
22     - name: "user"
23       description: "Operations about user"
24       externalDocs:
25         description: "Find out more about our store"
26         url: "http://swagger.io"
27   schemes:
28     - "https"
29     - "http"
30   paths:
31     /pet:
32       post:
33         tags:
34           - "pet"
35         summary: "Add a new pet to the store"
36         description: ""
37         operationId: "addPet"
38         consumes:
39           - "application/json"
40           - "application/xml"
41         produces:
42           - "application/xml"
```

Swagger

Yaitu versi dari Swagger yang dipakai.

Coba perhatikan contoh di samping ya, bestie!



The diagram illustrates the relationship between the Swagger version and the generated API documentation. A small dark box at the top contains three colored dots (red, yellow, green) and the text "swagger: \"2.0\"". A vertical arrow points downwards from this box to a larger dark box below it. This larger box also contains three colored dots and is labeled "Info" on its right side. Inside this "Info" box, a detailed Swagger 2.0 schema is shown:

```
swagger: "2info:
description: "this is API"
version: "1.0.0"
title: "Nama API Kamu"
termsOfService: "https://websitekamu/linkKeTermOfService"
contact:
  email: "kamu@mail.com"
license:
  name: "Apache 2.0"
  url: "http://www.apache.org/licenses/LICENSE-2.0.html"
```

Info

Yaitu informasi tentang dokumentasi API. Contohnya sama kayak di slide sebelumnya~

```
swagger: "2.0"
info:
  description: "this is API"
  version: "1.0.0"
  title: "Nama API Kamu"
  termsOfService: "https://websitekamu/linkKeTermOfService"
  contact:
    email: "kamu@mail.com"
  license:
    name: "Apache 2.0"
    url: "http://www.apache.org/licenses/LICENSE-2.0.html"
```

Host

Berisi informasi host (alamat RESTful API).

Karena nantinya kita bakal pasang Swagger ini langsung ke API, maka kita bisa hapus properti ini.

Coba lihat gambar di samping, ya!



basePath

Yaitu namespace dari endpoint API kita. Fungsinya untuk mengidentifikasi API kalau ada banyak endpoint.

Biasanya REST API pakai namespace/API, tapi nggak wajib, kok. Jadi **property ini bisa dihilangkan**.

Meskipun begitu, kalau API kita pakai namespace /api, maka tambahkan properti ini dengan nilai /api, ya.



Tags

Yaitu list dari kategori-kategori di API kita.

Contohnya, kita punya resource articles, nantinya kita bakal kelompokkan endpoint-endpoint dari resource tersebut ke dalam sebuah tags.

Di samping adalah contohnya ya, gengs~

```
tags:  
  - name: "Article"  
    description: "Article Resources"
```



```
basePath: "/api"
```

Paths

Yaitu definisi endpoint yang mau dibikin dan meliputi banyak properti lain. Kalau yang ini contoh untuk **GET/articles**, yaaa~

```
● ● ●  
paths:  
  "/articles":  
    get:  
      tags:  
        - "Article"  
      summary: "List all available articles"  
      description: "described hit"  
      produces:  
        - "application/json"  
      responses:  
        "200":  
          description: "Success"  
          schema:  
            type: array
```

```
● ● ●  
  items:  
    type: object  
    properties:  
      id:  
        type: integer  
        example: 1  
      title:  
        type: string  
        example: "Lorem Ipsum"  
      body:  
        type: string  
        example: "Dolor sit amet"
```

Nah kalau yang ini, contoh untuk **POST/articles**.

Request ini pakai Content-Type: application/json, gengs.

```
post:  
  tags:  
    - "Article"  
  summary: "Add new article"  
  description: "Create new"  
  parameters:  
    - in: "body"  
      name: "body"  
      description: "Article object"  
      required: true  
      schema:  
        type: object  
        properties:
```

```
      title:  
        type: string  
        example: Lorem Ipsum  
      body:  
        type: string  
        example: Dolor sit amet  
      consumes:  
        - "application/json"  
      produces:  
        - "application/json"  
      responses:  
        "201":  
          description: "Successfully create new article"
```

```
      schema:  
        type: object  
        properties:  
          id:  
            type: integer  
            example: 1  
          title:  
            type: string  
            example: "Lorem Ipsum"  
          body:  
            type: string  
            example: "Dolor sit amet"  
        "400":  
          description: "Failed to create new article"  
          # Disinilah kita mendefinisikan bentuk object dari response-nya  
          schema:  
            type: object  
            properties:  
              message:  
                type: string  
                example: "Failed to create new article"
```

securityDefinition

Yaitu definisi authentication dalam API kita.

Ada beberapa definition di Swagger nih sob, yaitu: Basic Authentication, OAuth, dan API key. Karena disini authentication kita adalah token-based, jadi kita pakai definisi API-KEY, ya.



```
securityDefinitions:
```

```
Token:
```

```
  type: apiKey
```

```
  in: header
```

```
  name: Authorization # Nama key dari header yang kita gunakan untuk mengirim token
```

Di samping merupakan penambahan security layer di endpoint (**POST/articles**), gengs. Coba dilihat yaa~

```
post:  
  tags:  
    - "Article"  
  summary: "Add new article"  
  description: "Create new article"  
  
  # layer security di endpoint definition  
  security:  
    - Token: []  
  
  parameters:  
    - in: "body"  
      name: "body"  
    # ...
```

Masuk ke pembahasan yang lebih mendalam lagi, kita bakal kupas tuntas tentang **API Contract**, nih.

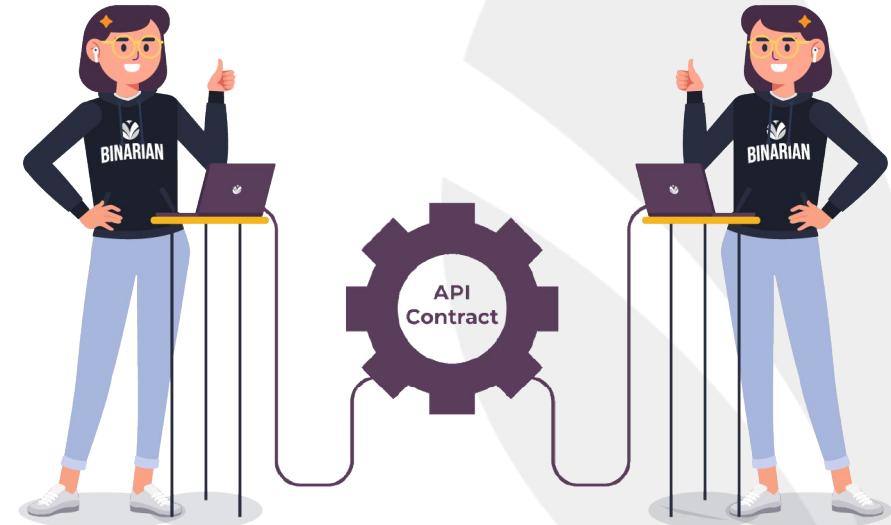
API Contract ini berisi sebuah kesepakatan pas kita mau bikin API.

Kesepakatan yang dimaksud tuh kayak gimana yaaa?



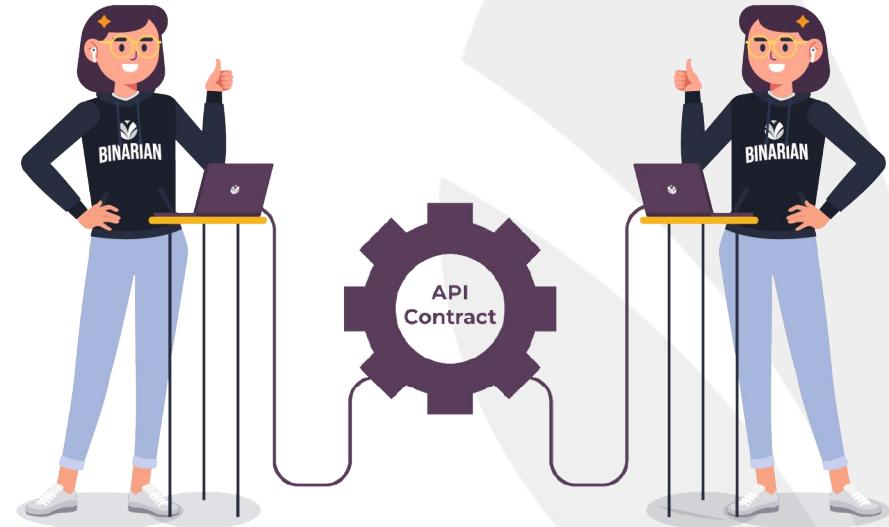
API Contract dibuat biar bisa saling berkomunikasi~

Kalau kita mau kerja sama dengan developer lain dan development-nya dilakukan secara bersamaan, contohnya front-end dan back-end, maka diperlukan kesepakatan ketika merancang API supaya bisa saling berkomunikasi, nih.



Hal ini diperlukan karena setiap aplikasi punya kebutuhannya masing-masing.

Oleh karena itu, kesepakatan ini disebut dengan **API contract**.



Sebuah API contract bisa aja berupa dokumentasi kayak pakai swagger, lho~

Konsepnya tuh gini, kita nggak perlu selalu pakai swagger pas melakukan API contract. Tapi, ketika melakukan API contract ada beberapa hal yang harus selalu diperhatikan.



Berikut adalah hal-hal yang perlu diperhatikan~

1. Request Body

Berisi pertanyaan seperti, apakah ada field yang wajib (mandatory) atau nggak (optional), atau field tersebut punya tipe data apa.

2. Request Header

Berisi pertanyaan seperti, apakah ada header mandatory atau optional, apakah ada header yang berfungsi sebagai authorization atau nggak. Hal ini bakal menentukan API bisa diakses secara public atau nggak



Hal yang lainnya...

3. Parameter yang dibutuhkan
4. URL
5. Contoh keseluruhan API

Baik dari body maupun header.



LANJUTTT



Binarian, nggak kerasa ya kita udah sampai di penghujung materi! □

Sebagai penutup topic, kita bakal menyempurnakan pemahaman kita dengan belajar **Implementasi Swagger**.

Go! Go! Go!

“Kasih tahu dong caranya kayak gimana?!”

Untuk melakukan dokumentasi API menggunakan Swagger, kita bakal pakai dependency dari [springdoc](#).

Kalau pakai springdoc, bukan cuma bisa mengatur Swagger Editor aja, tapi juga bisa menampilkan swagger UI.

Untuk melihat dokumentasi dari springdoc secara lengkap, bisa kamu pelajari [di sini](#), yaaaa~



Ada lagi. Untuk menambahkan swagger di aplikasi yang udah dibikin, kita cuma perlu menambahkan dependency disamping pada pom.xml.

Kalau pakai `springdoc-openapi-ui`, secara otomatis seluruh API di aplikasi bakal terdeteksi.

Keren, khannn?!



```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-ui</artifactId>
</dependency>
```

Berikut adalah buat mengecek API!



```
hostname/v3/api-docs/  
hostname/v3/api-docs.yaml
```

Kalau untuk melakukan customisasi pada path dengan mengatur properties,
berikut adalah contohnya~



```
springdoc.api-docs.path=/my-api
```

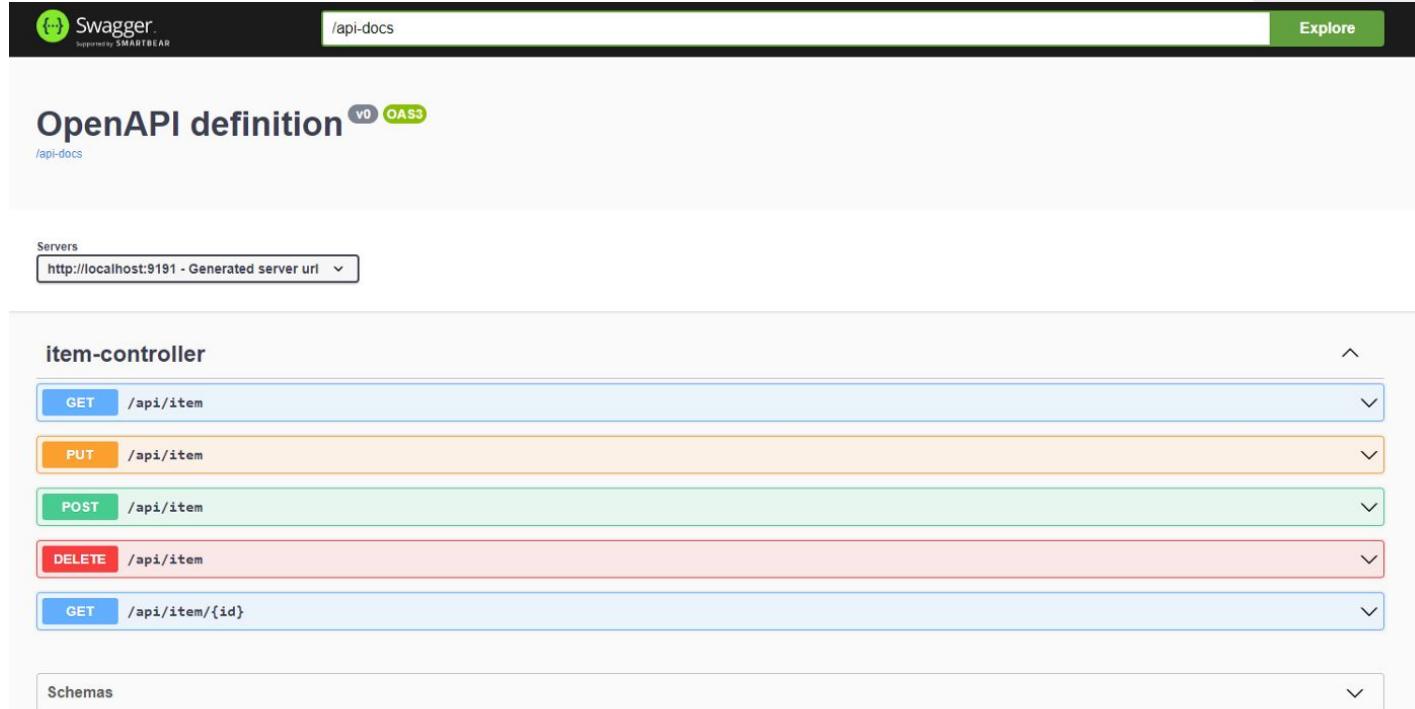
Kalau yang ini untuk mengecek UI-nya



Terus, kalau mau melakukan customisasi pada path UI dengan mengatur properties,
berikut adalah contohnya~



Berikut contoh tampilan default Swagger UI saat sudah menambahkan dependency springdoc



The screenshot shows the Swagger UI interface for an API. At the top, there's a navigation bar with the Swagger logo, the URL '/api-docs', and a green 'Explore' button. Below the navigation, the title 'OpenAPI definition' is displayed, followed by 'v0 OAS3' and the URL '/api-docs'. A 'Servers' section shows a single entry: 'http://localhost:9191 - Generated server url'. The main content area is titled 'item-controller'. It lists several API endpoints with their methods and URLs: 'GET /api/item' (blue background), 'PUT /api/item' (orange background), 'POST /api/item' (green background), 'DELETE /api/item' (red background), and 'GET /api/item/{id}' (light blue background). Each endpoint has a dropdown arrow to its right. At the bottom, there's a 'Schemas' section with a dropdown arrow.

Kita juga bisa melakukan kustomisasi tampilan dengan menambahkan konfigurasi berikut, lho~

```
@Bean
public OpenAPI customOpenAPI(@Value("Binar API")
                               String appDescription,
                               @Value("v1.0.0")
                               String appVersion) {
    return new OpenAPI()
        .info(new io.swagger.v3.oas.models.info.Info()
            .title("Open API")
            .version(appVersion)
            .description(appDescription)
            .termsOfService("http://swagger.io/terms/")
            .license(new License()
                .name("Apache 2.0").
                url("http://springdoc.org"))));
}
```

Secara lengkap penggunaan springdoc swagger beserta customisasi-nya bisa kamu simak di referensi berikut, yaaa!

[Documenting a Spring REST API](#)



Materi udah selesai, nih.

Yippie!! selamat ya kita udah super produktif hari ini. Selanjutnya ada latihan dan penugasan yang melatih keterampilan kamu.

Kita ke latihan dulu yaitu, silakan kamu **buat dokumentasi API dengan Swagger**.

Kalau udah, hasil penggeraan kamu bisa didiskusikan dengan teman sekelas atau fasilitator. Selamat mencoba, Binarian~



Eitss.. belum selesai. Selain latihan, ada juga penugasan alias PR buat kamu.

Silakan **ubah dan buatlah dokumentasi API dari bentuk Swagger ke bentuk file/dokumen**. Referensi dokumennya bisa kamu temukan di internet.

Nanti diskusikan hasilnya bersama facilitator dan teman sekelas pada pertemuan berikutnya, okey?

Selamat mengerjakan □





Last banget! Sabrina punya pertanyaan diskusi nih buat kamu.

Menurutmu, seberapa penting sih seorang Back End Engineer harus mendokumentasikan API?

Terus, apa aja ya kemungkinan yang bakal terjadi kalo kita nggak mendokumentasikan API pada project yang kita buat? Boleh dijawab yaa □

Nah, selesai sudah pembahasan kita di Chapter 5 Topic 4 ini.

Selanjutnya, kita bakal bahas tentang **Working With Media and PDF** alias cara download dan upload foto, video, serta bikin file PDF.

Penasaran kayak gimana? Yuk, langsung ke topik selanjutnya~

