

Spring Web (Part 1)

Gold - Chapter 5 - Topic 1

Selamat datang di **Chapter 5 Topic 1**
online course **Back End Java** dari
Binar Academy!



Ciyeet chapter baru~ □

Pada chapter sebelumnya kita belajar tentang Spring framework. Masih sodaraan nih, di chapter baru ini kita bakal mengetahui cara menerapkan **Spring Web** untuk membuat Restful API.

Eitss.. sebelum ke cara buatnya, kayaknya belum afdol kalau kita nggak belajar konsep dasarnya, dulu. Khusus di topik pertama ini, kita bakal mengelaborasi tentang **Spring Web (Part 1)**.

Yuk langsung aja kita kepoin~



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

- Konsep MVC View
- Pengantar Spring Web
- HTTP Concept
- HTTP Method
- REST dan RESTful API
- JSON, XML dan ISO8583
- Struktur JSON
- Cara membuat REST dan RESTful API dengan @Controller annotation
- Cara testing pada API



Spring Web, istilah ini akan berkaitan erat sama yang namanya REST API dan MVC.

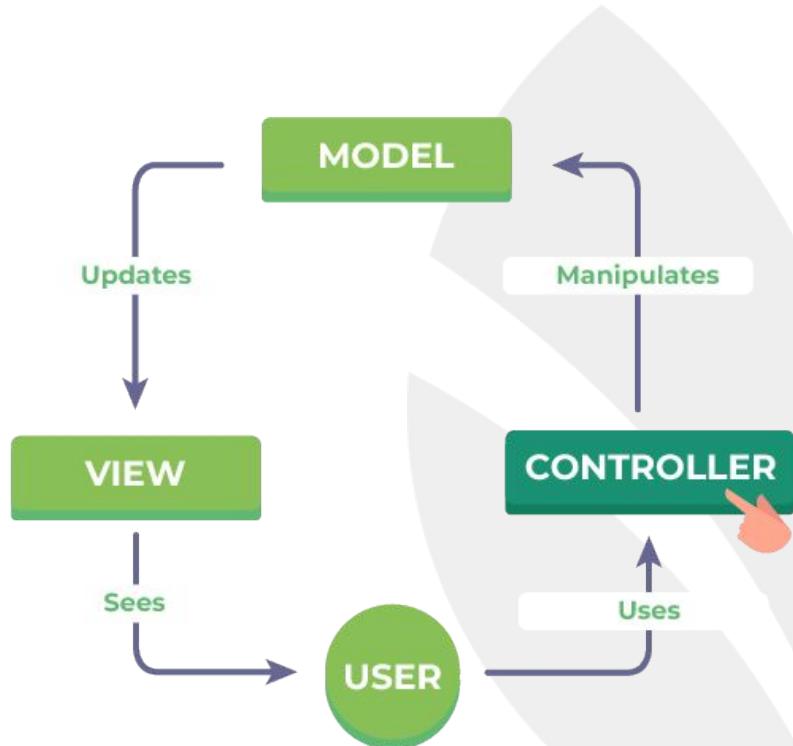
Supaya pemahaman kita clear, gimana kalau kita jabarin **MVC view**, dulu?



MVC pada controller layer~

Pada suatu arsitektur MVC, sebuah controller layer dipakai untuk meng-handle request yang masuk dan memberikan response.

Kalau di topic sebelumnya kita udah bahas secara rinci tentang model layer di mana sebuah business logic dan persistensi data berlangsung, di topic ini, kita bakal fokus pada controller layer ajaaa~

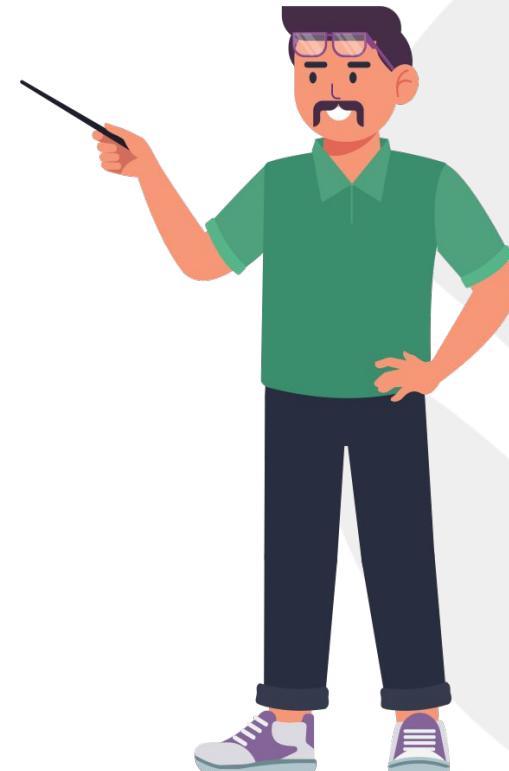


Gimana sih sebuah controller layer berinteraksi dengan view?

Si controller ini bakal membentuk suatu endpoint yang diakses oleh view layer.

Di course ini, controller berinteraksi dengan protokol HTTP dimana controller bakal meneruskan HTTP request yang ada. Sehingga nantinya bisa diproses oleh business logic yang udah dibuat.

Belum selesai sampai situ aja, nanti controller juga bakal memberikan respon kembali yang berasal dari business logic layer tadi.



Sebagaimana namanya, Spring Web itu ada kaitannya sama Spring dan web.

Iya. Spring yang Spring Framework pada chapter sebelumnya itu.

Tapi berhubung ada kata web disini, kira-kira kaitan diantara keduanya itu gimana, ya?



Apa itu Spring Web?

Spring Web merupakan modul dari Spring untuk **menghandle web**, [web-servlet](#) dan [web-portlet](#).

Kombinasi Spring Boot dan Spring Web bisa menampilkan web–servlet dengan konfigurasi yang lebih sederhana dibandingkan dengan Spring MVC.



Selain itu, kalau kita pakai Spring web, nantinya kita jadi bisa bikin REST API dan web-servlet (bentuk MVC).

Meskipun bisa dipakai untuk membuat web-servlet (bentuk MVC), tapi di pembahasan topic ini kita cuma bakal pakai buat bikin REST API aja, yaaa~



Pasti kamu sering dengar istilah HTTP, kan?

Yap, betul banget. HTTP ini sering kita baca dibagian URL ketika mencari sesuatu di google.

Sekarang, kita bakal bahas HTTP secara lengkap di **HTTP Concept**. Yuk!



Buat perkenalan, kita bakal bahas HTTP Server dulu, yaa~

Ada satu fun fact. Ketika kamu lagi cari tahu informasi di google, itu berarti kamu lagi mengirim sebuah request.

Misalnya, kamu mau informasi tentang konser Justin Bieber. Nantinya, google bakal mengirim request pencarian kamu ke server dan hasilnya bakal memunculkan informasi seputar konsernya Justin Bieber.

Kayak tanggal konsernya, lokasi konsernya, sampai cara beli tiketnya.



Untuk menghandle request, HTTP server punya dua komponen~

Iya. Jadi, ada dua komponen penting yang perlu kita perhatikan ketika akan meng-handle request, yaitu **request header** dan **body**.

“Hmm itu apa, ya?”

Yuk kita cek penjelasannya!



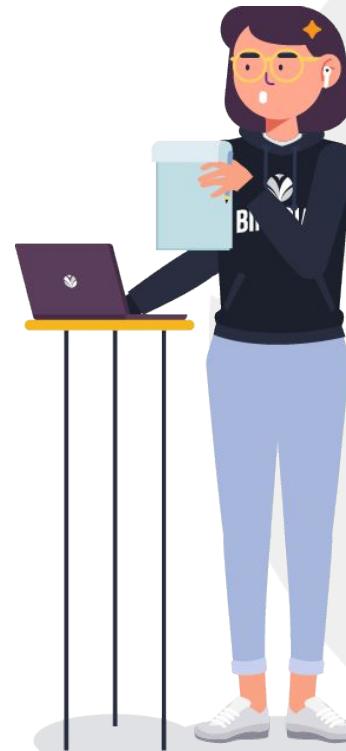
Ini dia penjelasan Headers dan Body pada Request~

- **Headers** berguna untuk memberikan informasi, baik dari client ke server maupun server ke client.

Headers bisa dipakai untuk berbagai macam keperluan, kayak autentikasi dan penjelasan tentang konten yang ada pada body dari request yang dikirimkan.



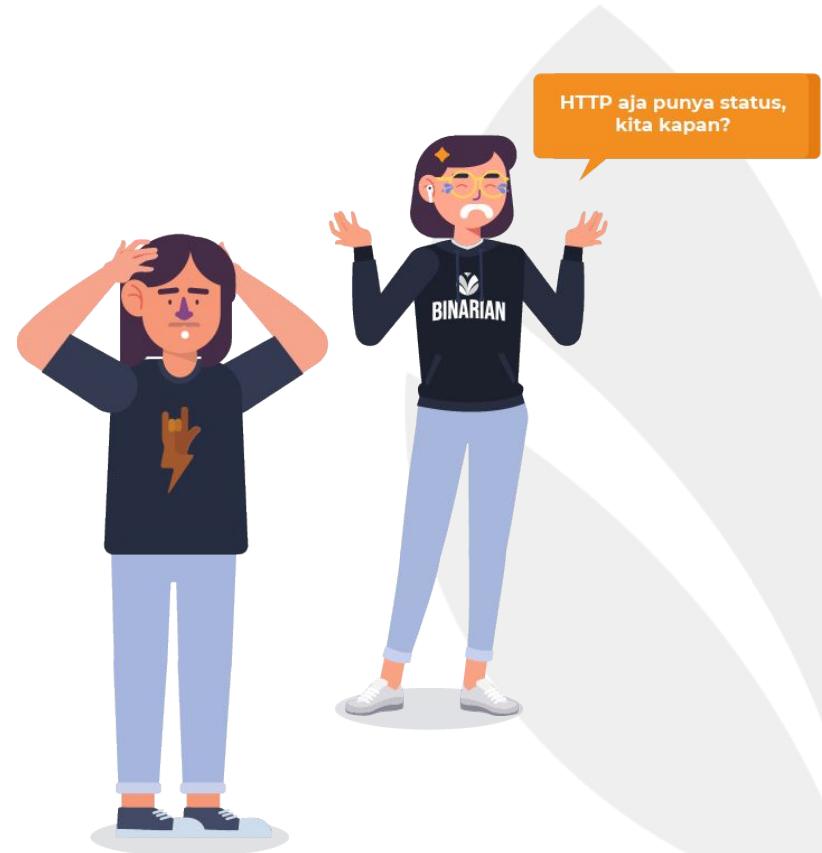
- **Data atau Body** adalah bagian dari request yang menyimpan informasi yang dikirimkan. Body cuma dipakai di method POST, PUT atau PATCH atau DELETE.



Jangan salah, HTTP juga punya status, lho~

Yepp, namanya status kode.

Status kode adalah kode berupa angka dari 100+ hingga 500+ yang bisa menjelaskan status dari response yang dikirimkan oleh server.



HTTP Status Codes

Kalau kamu belum punya gambaran, berikut ini ada penjelasan kodenya:

- 200+ berarti request yang kita kirim berhasil atau sukses.
- 300+ berarti request yang kita kirim dialihkan ke URL lain.
- 400+ berarti terjadi error yang berasal dari client berkaitan dengan request yang dikirim.
- 500+ berarti terjadi error yang berasal dari server.



Udah kenalan sama HTTP Server, sekarang kita bakal kenalan sama **HTTP Method** yang masih punya hubungan sama HTTP Server.

Tapi, hubungan kayak gimana sih yang dimaksud?



Buat melakukan request ke server, ada beberapa jenis method yang bisa kita pakai lho, sob!

Method ini di antaranya yaitu:

- **GET**
- **POST**
- **PUT atau PATCH**
- **DELETE**

HTTP Methods and Their Meaning

Method	Meaning
GET	Read Data
POST	Insert Data
PUT or PATCH	Update data, or insert if a new id
DELETE	Delete data



Beberapa method tersebut menunjukkan tipe dari request yang biasanya dipakai untuk melakukan aksi, yaitu CREATE, READ, UPDATE, DELETE atau yang biasa disebut dengan **CRUD**

HTTP Methods and Their Meaning

Method	Meaning
GET	Read Data
POST	Insert Data
PUT or PATCH	Update data, or insert if a new id
DELETE	Delete data

“Kasih penjelasan dari method - method tadi, dong!”

Okedeh~ kita bahas satu-satu ya!

- **POST**, adalah method yang biasa dipakai untuk membuat data baru di database yang ada di server.

Kita bisa mengirimkan data yang bakal dibikin lewat body dari request.

Inilah yang biasa disebut dengan aksi **CREATE (membuat)**.



- **GET**, adalah method yang biasa dipakai untuk mendapatkan data dari server.

Ketika kita melakukan request dengan method GET, maka server bakal mencari data yang sesuai dengan kebutuhan kita lalu mengembalikan data tersebut lewat response.

Inilah yang biasa disebut dengan aksi **READ (membaca)**.



- **PUT atau PATCH.** adalah method yang dipakai untuk mengupdate data yang sudah ada pada database di server.

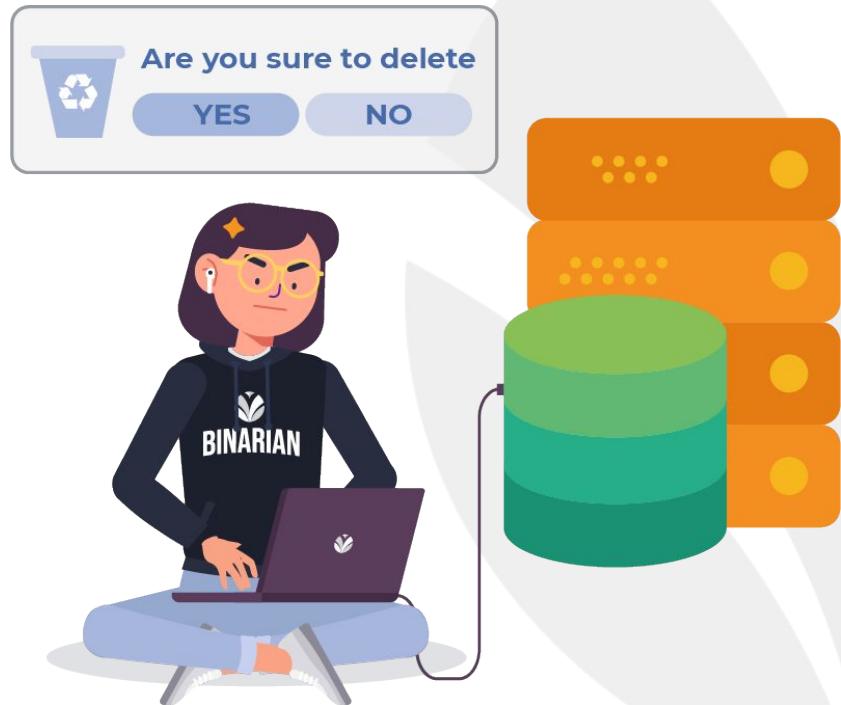
Mirip kayak POST, bedanya method ini biasanya digunakan untuk mengubah data.

Inilah yang biasa disebut dengan aksi **UPDATE (memperbarui).**



- **DELETE**, adalah method yang biasa dipakai untuk menghapus data yang tersedia pada database di server.

Inilah yang biasa disebut dengan aksi **DELETE (menghapus)**.



Bukan cuma HTTP Server dan HTTP Method aja yang berhubungan, kali ini kita bakal bahas dua konsep yang berhubungan juga, yaitu:

REST dan RESTful API.

Iya, yang didepan sempet disebutin itu~



“Apa perbedaan REST API dan RESTful API?”

Hmmm, sebenarnya nggak ada perbedaan berarti sih antara REST API dan RESTful API.

Tapi gini, sebuah API yang memanfaatkan pattern REST bisa dibilang sebagai API yang RESTful.

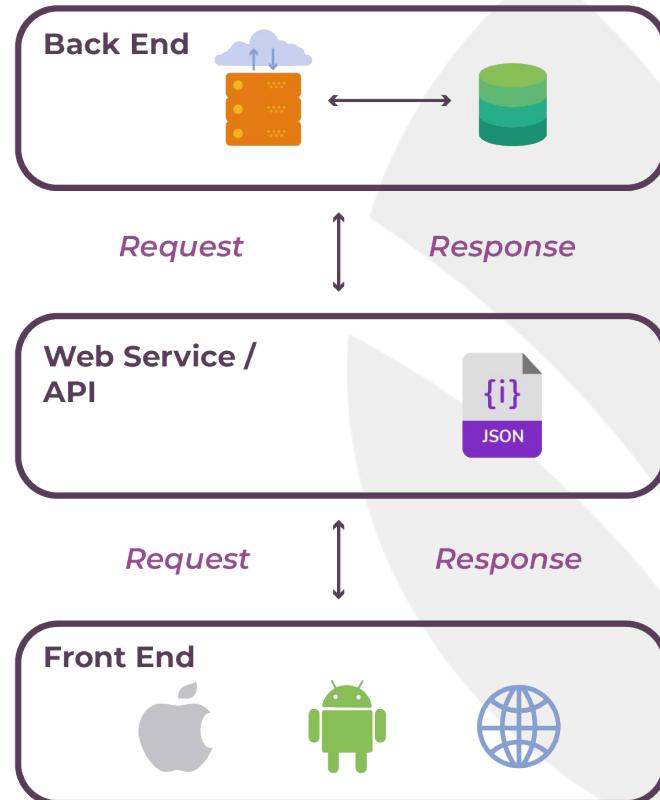
Masih bingung? Oke deh, simak penjelasan jelasnya di slide selanjutnya, ya!



Sebelum kita bahas RESTful API, kita harus paham dulu nih tentang API!

API atau Application Programming Interface adalah kumpulan aturan yang memungkinkan dua atau lebih program untuk berkomunikasi satu sama lain.

API dibuat pada server untuk selanjutnya bisa memungkinkan client berkomunikasi.



“Terus kalau RESTful itu apa dong?”

RESTful atau Representational State Transfer adalah jenis API atau yang menggambarkan bentuk dari API.

REST adalah kumpulan aturan yang diikuti oleh developer untuk membuat API.

Masih belum kebayang?

Oke, kita bakal bahas lebih lanjut tentang RESTful ini~



Ada dua hal penting yang perlu kita tahu sebelum kenal lebih jauh sama RESTful API nih, gengs~

Yaitu, **Request** dan **Response**.

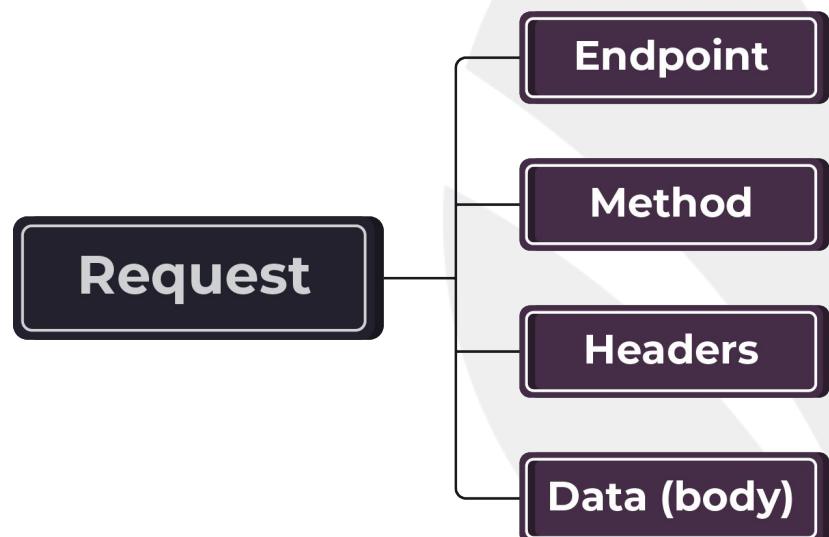
Setiap URL yang ada di RESTful API disebut dengan Request.

Sedangkan Response-nya adalah data yang dikembalikan setelah client melakukan Request.



Request dibangun dengan empat hal yang penting banget lho, yaitu:

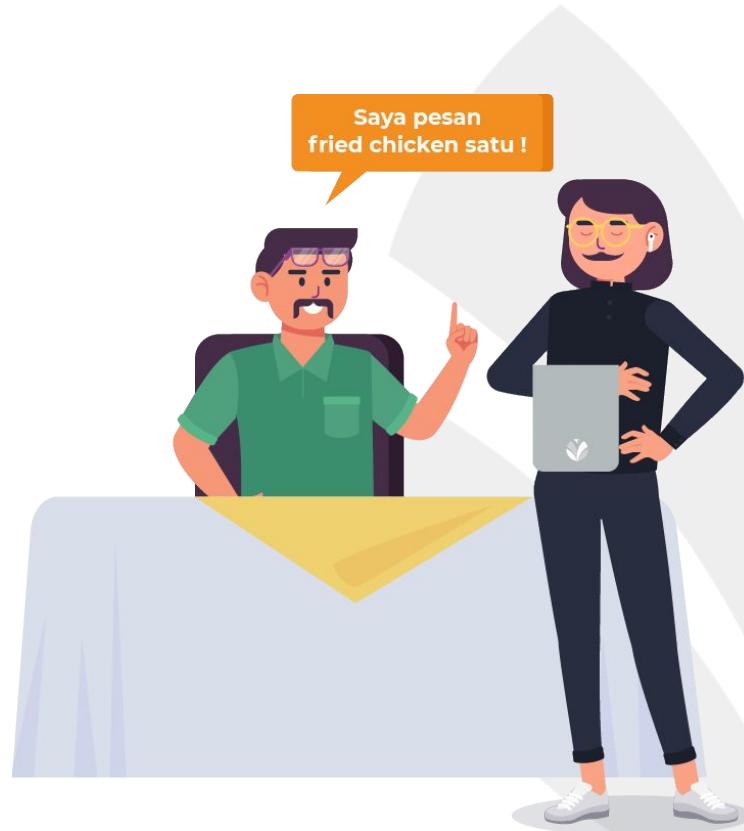
- **Endpoint**
- **Method**
- **Headers**
- **Data (body)**



Biar lebih paham, coba simak analogi berikut ya!

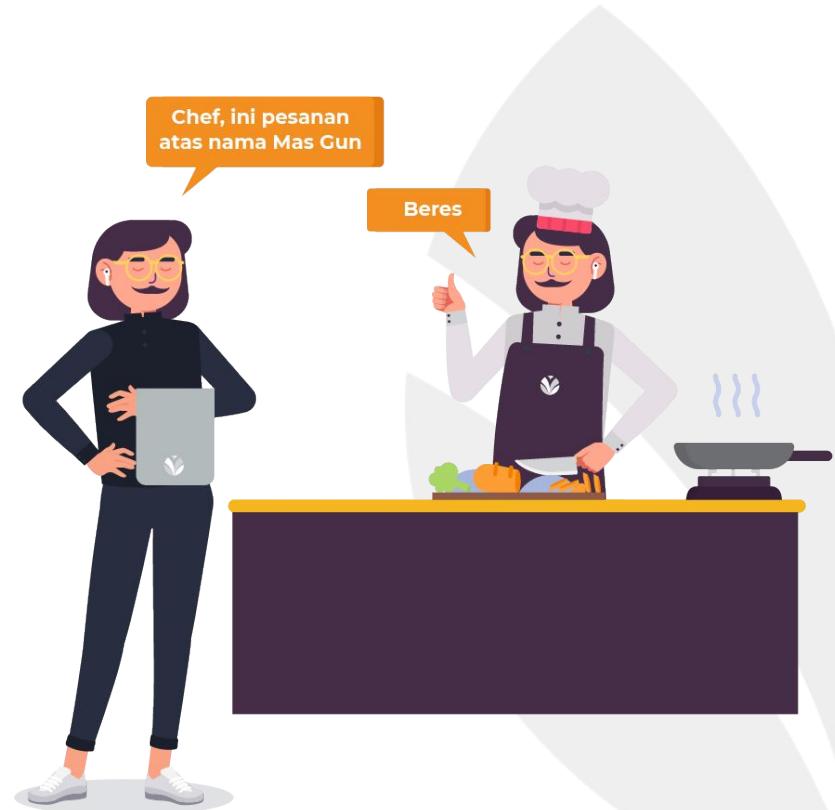
Misalnya kamu adalah pemilik restoran fried chicken. Lalu, mas Gun datang ke restoran kamu untuk mencicipi fried chicken ciri khas restoran kamu.

Waktu sampai di lokasi, mas Gun disambut oleh pelayan yang bertugas menghampiri Mas Gun sambil memberikan buku menu.



Terus nih, ketika mas Gun udah selesai pilih menu, si pelayan akan pergi ke dapur untuk menginformasikan koki tentang menu apa aja yang dipesan mas Gun.

Setelah itu, mas Gun cuma tinggal duduk manis sambil nunggu pesanannya diantar aja sama pelayan.

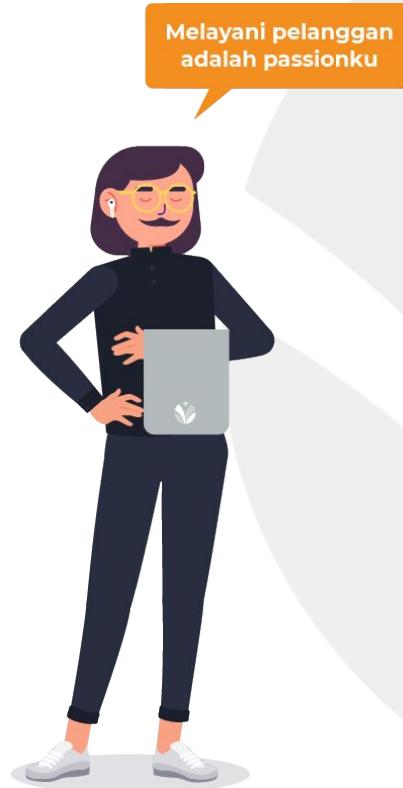


Kita balik lagi ke tugas pelayan yang memberikan buku menu sekaligus antar makanan ke mas Gun~

Kenapa waiters memberikan buku menu?

Karena restoran udah punya menu, dapur, koki dan bahan masakan sendiri yang dibuat sesuai standar restoran.

Kamu nggak mungkin minta mas Gun buat ngeracik sendiri fried chicken-nya, kan? Bisa-bisa nanti Mas Gun merubah tatanan dapur restoran.

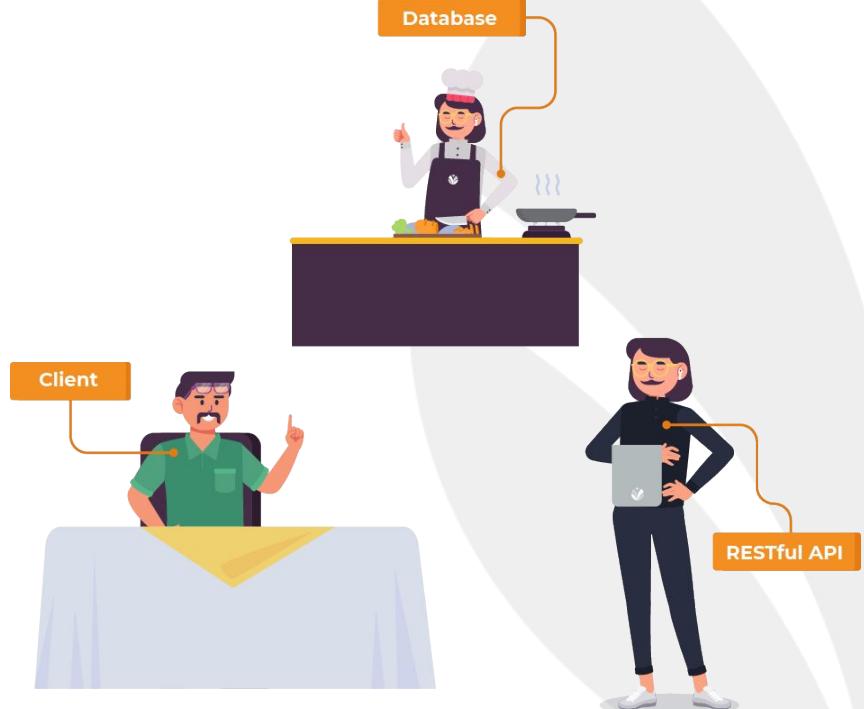


Dari analogi tadi, waiters berperan sebagai RESTful API lho, gengs!

Waiters atau pelayan ini diibaratkan sebagai RESTful API yang berperan menjadi **jembatan antara database dengan client** (program yang bakal mengkonsumsi data).

Jadi, client bisa berkomunikasi sama server, mengambil data, dan memanipulasi data yang ada di database sesuai sama aturan-aturan yang udah diberikan.

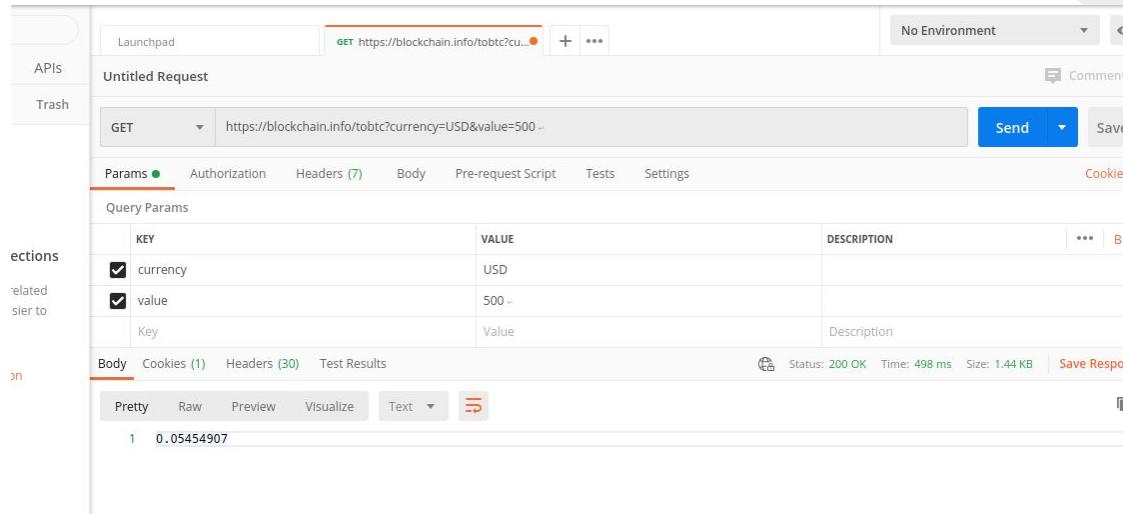
Tanpa memungkinkan mereka untuk merusak atau merubah sesuatu di luar aturan yang berlaku.



RESTful API ternyata berhubungan sama Endpoint, lho!

Iya, jadi endpoint atau route adalah URL atau alamat request yang biasanya bakal tampak kayak dibawah ini:

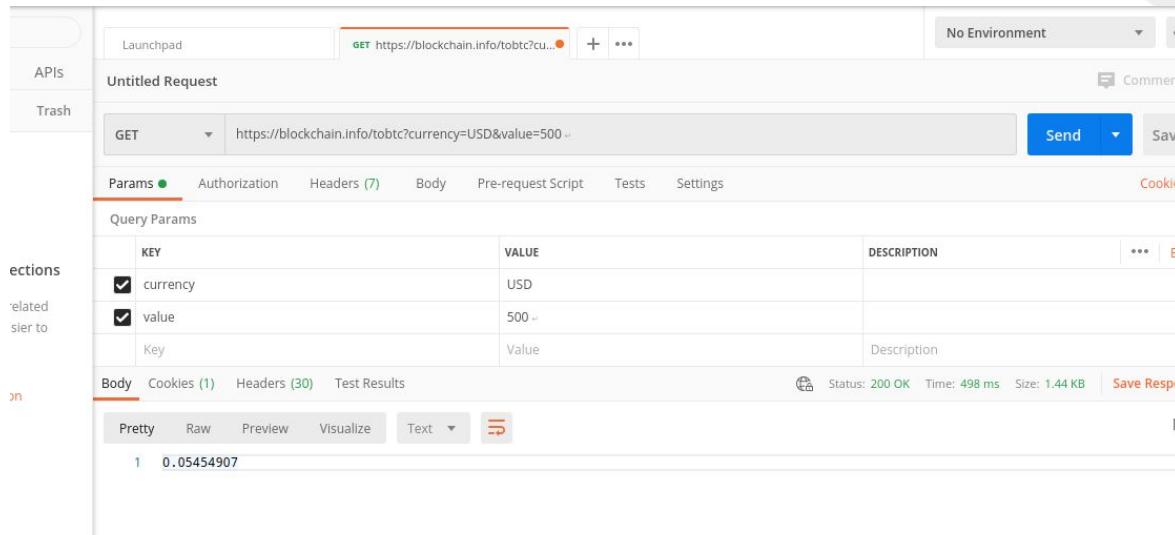
`https://blockchain.info/tobtc?currency=USD&value=500`



The screenshot shows the Postman application interface. On the left, there's a sidebar with 'APIs' and 'Trash' buttons. The main area has tabs for 'Launchpad', 'Untitled Request', and 'No Environment'. Below these are buttons for 'Comment', 'Send', and 'Save'. A dropdown menu shows 'GET' selected. The URL field contains `https://blockchain.info/tobtc?currency=USD&value=500`. The 'Params' tab is active, showing two entries: 'currency' with value 'USD' and 'value' with value '500'. Other tabs include 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. At the bottom, there are tabs for 'Body', 'Cookies (1)', 'Headers (30)', and 'Test Results'. The status bar at the bottom right shows 'Status: 200 OK', 'Time: 498 ms', 'Size: 1.44 KB', and 'Save Response'. Below the status bar, there are buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', 'Text', and a copy icon.

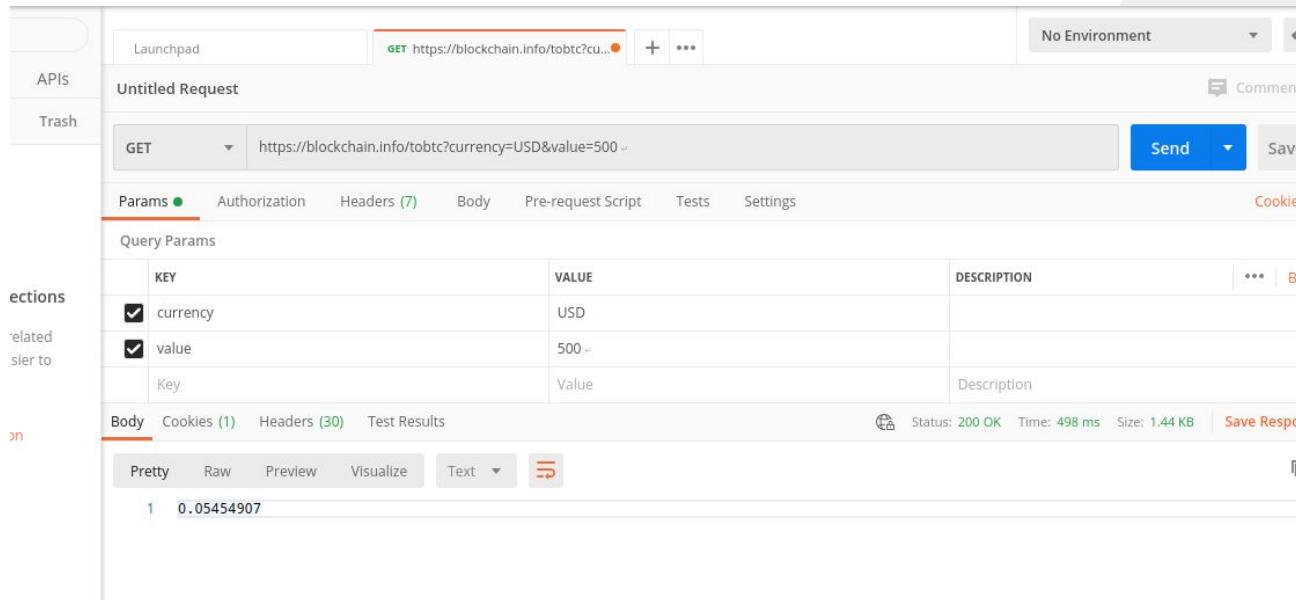
URL sebelumnya adalah contoh REST API yang bisa dipakai untuk mendapatkan harga bitcoin saat ini. Yes, pada contoh kali ini, kita pakai contoh harga bitcoin.

Kita bisa memodifikasi 2 parameter yang terdiri dari, currency dan value. Yaitu untuk mendapatkan data yang sesuai sama kebutuhan kita.



The screenshot shows the Postman application interface. On the left, there's a sidebar with sections like 'APIs' and 'Trash'. The main area has tabs for 'Launchpad', 'Untitled Request', and 'No Environment'. The 'Untitled Request' tab is active, showing a GET request to `https://blockchain.info/tobtc?currency=USD&value=500`. Below the URL, there are buttons for 'Send' and 'Save'. Underneath the URL input, there are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Params' tab is selected, showing 'Query Params' with two entries: 'currency' set to 'USD' and 'value' set to '500'. There's also a 'Cookies' tab. At the bottom, there are tabs for 'Body', 'Cookies (1)', 'Headers (30)', and 'Test Results'. The 'Body' tab is selected, showing the response: '1 0.05454907'. Other tabs show 'Status: 200 OK', 'Time: 498 ms', and 'Size: 1.44 KB'. There are also 'Pretty', 'Raw', 'Preview', 'Visualize', 'Text', and 'JSON' buttons at the bottom of the body section.

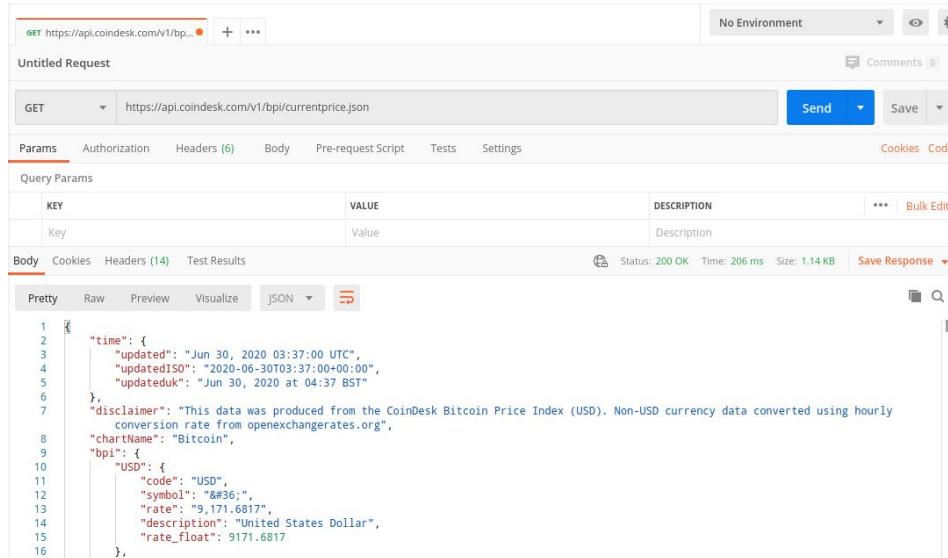
Sekarang kita coba lakukan request ke endpoint-nya pakai [Postman](#). Cukup ketikan URL tadi ke dalam kolom URL, lalu tekan tombol **Send**. Perhatikan gambar dibawah, ya!



The screenshot shows the Postman interface with the following details:

- Launchpad** tab is selected.
- Untitled Request** is the current request name.
- Method**: GET
- URL**: <https://blockchain.info/tobtc?currency=USD&value=500>
- Send** button is highlighted.
- Params** tab is active, showing two query parameters:
 - currency** (Value: USD)
 - value** (Value: 500)
- Body**, **Cookies (1)**, **Headers (30)**, and **Test Results** tabs are also present.
- Status**: 200 OK
- Time**: 498 ms
- Size**: 1.44 KB

Kalau koneksi internet kamu aman, kamu bakal dapat response dari server dimana API tersebut dibuat. Response ini berupa harga bitcoin saat ini yang sesuai dengan parameter yang dimasukkan, yaitu mata uang USD pakai jumlah 500.

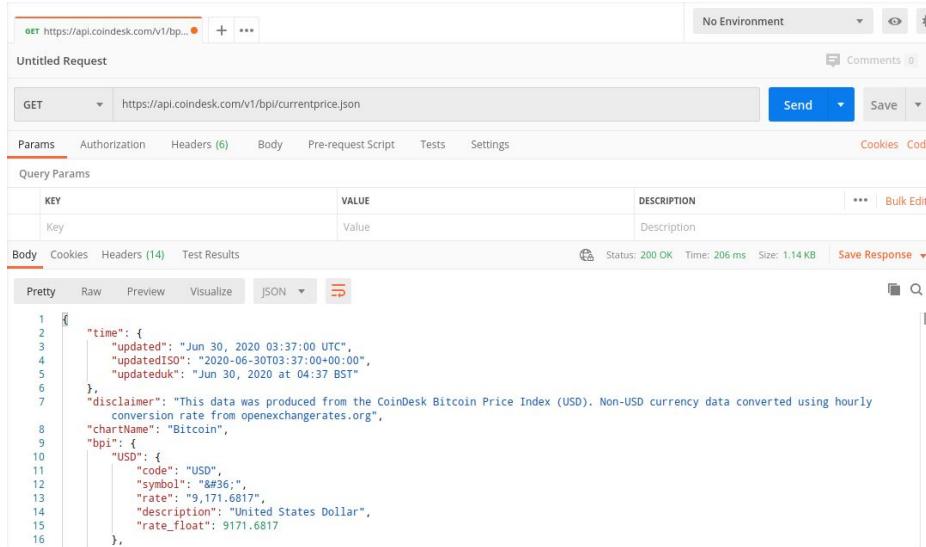


The screenshot shows the Postman application interface. A GET request is being made to <https://api.coindesk.com/v1/bpi/currentprice.json>. In the 'Params' tab, there is a single entry for 'amount' with a value of '500'. The response status is 200 OK, and the response body is displayed below:

```
1 {
2     "time": {
3         "updated": "Jun 30, 2020 03:37:00 UTC",
4         "updatedISO": "2020-06-30T03:37:00+00:00",
5         "updateduk": "Jun 30, 2020 at 04:37 BST"
6     },
7     "disclaimer": "This data was produced from the CoinDesk Bitcoin Price Index (USD). Non-USD currency data converted using hourly conversion rate from openexchangerates.org",
8     "chartName": "Bitcoin",
9     "bpi": {
10         "USD": {
11             "code": "USD",
12             "symbol": "&#36;",
13             "rate": "9,171.6817",
14             "description": "United States Dollar",
15             "rate_float": 9171.6817
16         }
17     }
18 }
```

Endpoint sebelumnya itu hanya mengembalikan response sederhana berupa harga dari bitcoin saat ini. Biasanya nih, buat data yang lebih kompleks, endpoint bakal mengembalikan response pakai format JSON. Ini contohnya:

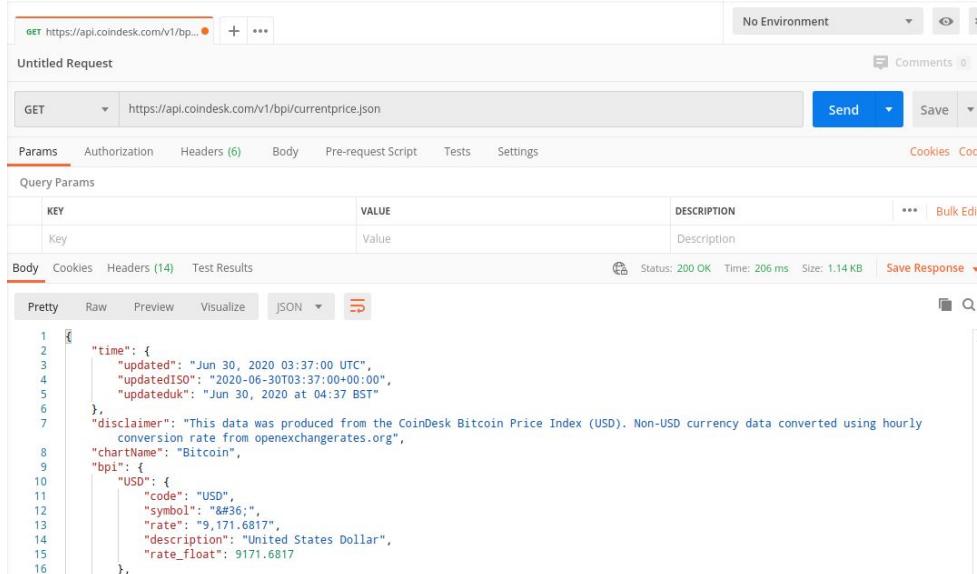
<https://api.coindesk.com/v1/bpi/currentprice.json>



The screenshot shows the Postman application interface. A GET request is made to <https://api.coindesk.com/v1/bpi/currentprice.json>. The response status is 200 OK, with a time of 206 ms and a size of 1.14 KB. The response body is displayed in JSON format:

```
1 {
2     "time": {
3         "updated": "Jun 30, 2020 03:37:00 UTC",
4         "updatedISO": "2020-06-30T03:37:00+00:00",
5         "updatedUK": "Jun 30, 2020 at 04:37 BST"
6     },
7     "disclaimer": "This data was produced from the CoinDesk Bitcoin Price Index (USD). Non-USD currency data converted using hourly conversion rate from openexchangerates.org",
8     "chartName": "Bitcoin",
9     "bpi": {
10         "USD": {
11             "code": "USD",
12             "symbol": "&#36;",
13             "rate": "9,171.6817",
14             "description": "United States Dollar",
15             "rate_float": 9171.6817
16         },
17     }
}
```

JSON atau Javascript Object Notation adalah format yang biasa dipakai untuk mengirim dan menerima data lewat RESTful API. Dimana response yang dikembalikan oleh endpoint tersebut adalah contoh data yang berformat JSON.



The screenshot shows the Postman application interface. At the top, there is a header bar with a 'GET' button, the URL 'https://api.coindesk.com/v1/bpi...', and a 'Send' button. Below the header, the title 'Untitled Request' is displayed. Underneath the title, the method 'GET' and URL 'https://api.coindesk.com/v1/bpi/currentprice.json' are specified. To the right of the URL are 'Send' and 'Save' buttons. Below the URL, there are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Headers' tab is currently selected. In the 'Headers' section, there is a table with columns 'KEY', 'VALUE', and 'DESCRIPTION'. There is one row in the table with the key 'Key' and value 'Value'. Below the headers, there are tabs for 'Body', 'Cookies', and 'Headers (14)'. The 'Body' tab is selected. On the right side of the body panel, it shows 'Status: 200 OK', 'Time: 206 ms', and 'Size: 1.14 KB'. Below this, there are buttons for 'Save Response' and a 'Pretty' button. The main content area displays the JSON response in a 'Pretty' format. The JSON structure includes a 'time' object with 'updated', 'updatedISO', and 'updateduk' properties, a 'disclaimer' string, a 'chartName' string, a 'bpi' object containing a 'USD' object with 'code', 'symbol', 'rate', 'description', and 'rate_float' properties. The JSON is numbered from 1 to 16.

```
1   {
2     "time": {
3       "updated": "Jun 30, 2020 03:37:00 UTC",
4       "updatedISO": "2020-06-30T03:37:00+00:00",
5       "updateduk": "Jun 30, 2020 at 04:37 BST"
6     },
7     "disclaimer": "This data was produced from the CoinDesk Bitcoin Price Index (USD). Non-USD currency data converted using hourly conversion rate from openexchangerates.org",
8     "chartName": "Bitcoin",
9     "bpi": {
10       "USD": {
11         "code": "USD",
12         "symbol": "&#36;",
13         "rate": "9,171.6817",
14         "description": "United States Dollar",
15         "rate_float": 9171.6817
16       },
17     }
18   }
```

Sipp deh!

Karena tadi udah disinggung tentang JSON. Berarti sekarang waktunya kita cari tahu lebih dalam tentang **JSON, XML dan ISO8583** (**kurang XML dan ISO8583**).

Berangkatttt~



“Jadi JSON itu apa sih?”

JSON adalah kependekan dari **JavaScript object notation**, yaitu format yang dipakai untuk menyimpan dan mentransfer data.

JSON sendiri terdiri dari dua struktur nih gengs, yaitu:

- Kumpulan value yang saling berpasangan. Dalam JSON, contohnya adalah object.
- Daftar value yang berurutan, seperti array.



Setidaknya ada enam jenis data yang bisa dipakai sebagai value JSON, gengs~

Berikut adalah jenis data yang dimaksud:

- **String**
- **Object**
- **Array**
- **Boolean**
- **Number**
- **Null**

Kita bahas satu per satu yuk!



String

String adalah data yang terdiri dari karakter unicode, kayak “Anton” pada contoh di bawah ini.

```
“nama”:“Anton”
```



Object

Object adalah sepasang key dan value.

Kayak contoh syntax tadi, object dibuka dan ditutup pakai kurung kurawal. Kalau ada lebih dari satu object, maka masing-masing dipisahkan oleh koma dan spasi.

```
“karyawan”: {“nama”：“Anton”, “asal”：“Bandung”}
```

Pada contoh di atas, semua yang ada di dalam kurung kurawal adalah object yang merupakan value dari “karyawan”

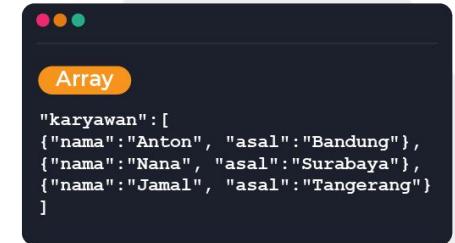


Array

Array adalah kumpulan object. Kumpulan ini dibuka dan ditutup dengan kurung siku [].

Contohnya adalah sebagai berikut:

```
"karyawan": [  
    {"nama": "Anton", "asal": "Bandung"},  
    {"nama": "Nana", "asal": "Surabaya"},  
    {"nama": "Jamal", "asal": "Tangerang"}  
]
```

A screenshot of a dark-themed code editor window. At the top, there's a yellow header bar with the word "Array". Below it is the code:

```
"karyawan": [  
    {"nama": "Anton", "asal": "Bandung"},  
    {"nama": "Nana", "asal": "Surabaya"},  
    {"nama": "Jamal", "asal": "Tangerang"}  
]
```

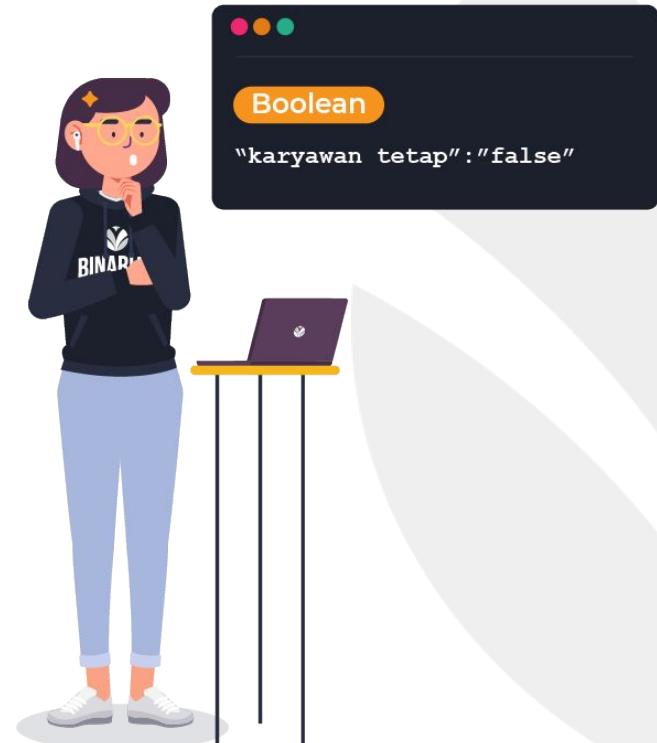
The code uses curly braces for objects and square brackets for the array itself, with commas separating the elements.

Boolean

Boolean adalah jenis data yang cuma berisi pernyataan benar atau salah.

Kamu tinggal memasukkan true atau false, kayak contoh berikut ini:

“karyawan tetap”：“false”



Number

Kayak namanya, jenis data ini cuma berupa angka. Dengan catatan, angkanya harus merupakan integer atau angka bulat. Artinya, 21,8 atau $\sqrt{2}$ nggak bisa dijadikan sebagai value.

Contoh penggunaannya kayak di bawah ini:

“usia”:“29”



Null

Kalau sebuah key nggak punya value, kamu bisa mengetikkan null.

Contohnya kayak code berikut, gengs.

```
“golonganDarah”:”null”
```



“Kalau XML itu apa, sih?”

Sebenarnya kita udah kenal file XML, kayak yang di pom.xml gitu~

XML sendiri merupakan singkatan dari **Extensible Markup Language**. Dimana sebuah REST API bisa pakai xml di body-nya.

Mirip kayak field yang ada di JSON, kalau XML ini menggunakan tag.



**E X TENSIBLE
M ARKUP
L ANGUAGE**

The word "EML" is written vertically in a large, bold, dark purple font. Each letter is enclosed in a yellow square. The letters are arranged as follows: "E" is at the top, "X" is in the middle row, "TENSIBLE" is to the right of "X"; "M" is in the middle row, "ARKUP" is to the right of "M"; "L" is at the bottom, "ANGUAGE" is to the right of "L".

Kalau JSON bisa bikin array, XML juga nggak mau kalah, nih. XML juga bisa bikin array, gengs!

Hal ini bisa ditunjukkan pada tag <**dependecies**> yang terdiri dari object <dependency> pada pom xml.

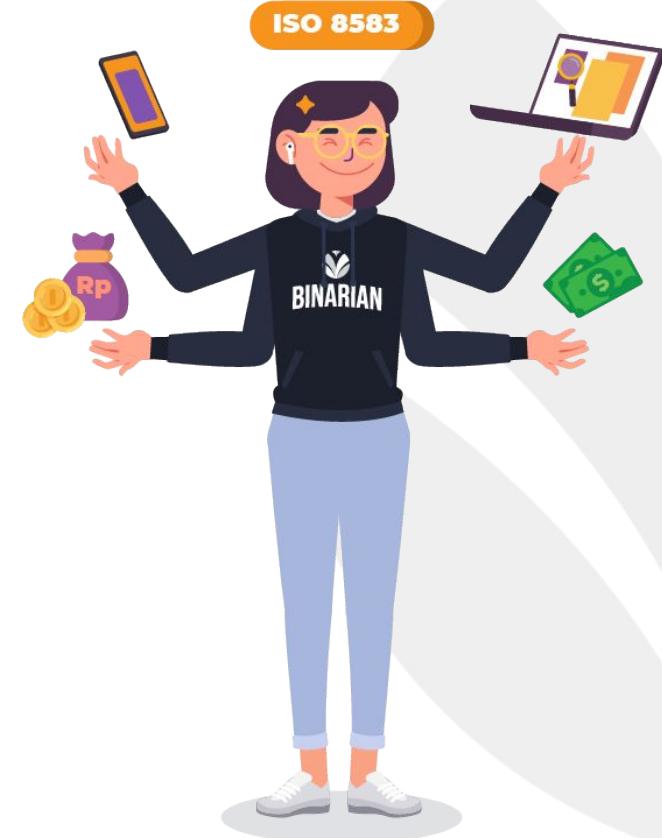
**E X TENSIBLE**
**M ARKUP**
**L ANGUAGE**



“Terus kalau ISO 8583 itu apa, dong?”

ISO 8583 ini udah banyak banget **dipakai untuk transaksi financial**, terutama transaksi di dunia perbankan.

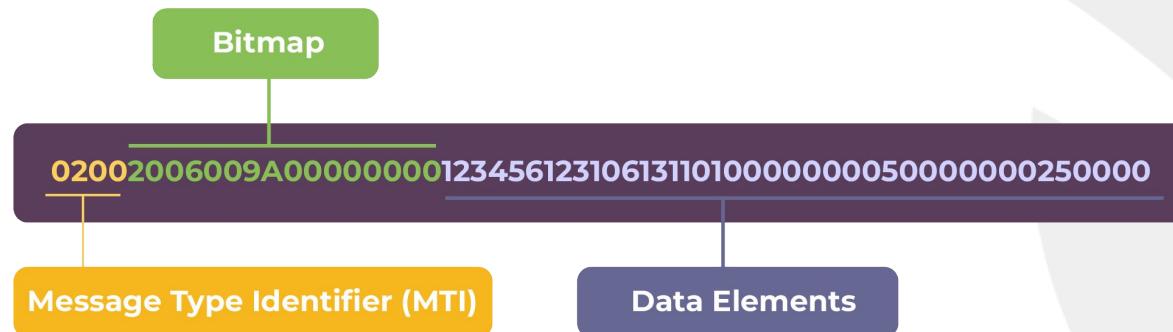
ISO 8583 ini nggak cuma bisa dipakai di bagian perbankan aja, tapi juga dipakai di bagian lain yang berhubungan dengan hal transaksi financial.



Ada catatan penting, nih. ISO 8583 bukanlah bagian dari REST API, melainkan ISO 8583 ini pakai protocols yang berbeda.

Meskipun begitu, kadang-kadang seorang Java developer suka mengintegrasikan aplikasinya ini pakai ISO8583.

Berikut adalah contoh data dengan ISO 8583!



Penjelasan lebih lengkap tentang ISO 8583 bisa kamu pelajari dalam referensi di bawah, yaaaa~

[ISO 8583](#)

Kalau udah paham sama konsepnya JSON, kurang lengkap dong kalau nggak paham juga sama **Struktur JSON**.

Pagi-pagi ke pasar minggu.

Nggak usah nunggu, yuk kita cari tahu!

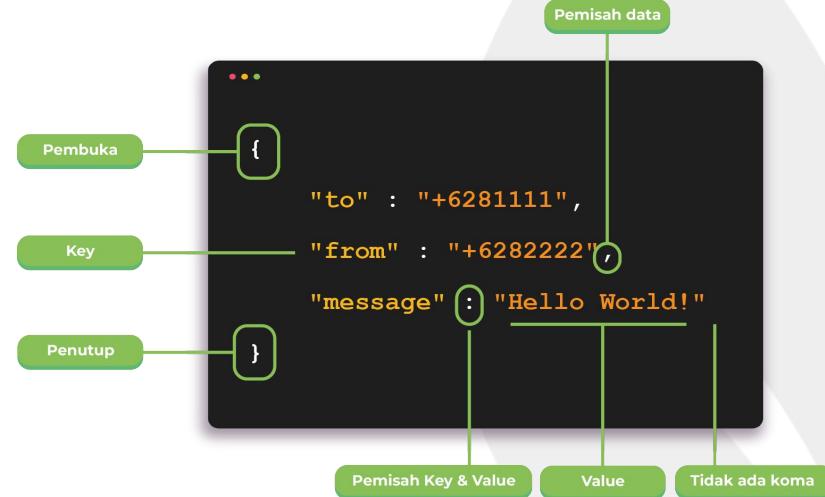


Struktur Dasar JSON

Coba perhatikan struktur JSON pada gambar di samping. Ini adalah struktur JSON yang paling sederhana.

JSON selalu dimulai dengan tanda kurung kurawal { dan ditutup dengan kurung }.

Di dalam kurung kurawal, berisi data yang formatnya key dan value. Kalau ada lebih dari satu data, maka dipisah dengan tanda koma dan di data terakhir nggak dikasih koma.



Oh iya, terus kalau buat value-nya..

Kamu bisa memberikan tipe data apa pun, lho.

Bahkan kamu juga bisa isi pakai array dan objek, sampai tipe data null, number, dan Boolean.

Keren banget, kan?

```
1 {  
2     "value1": null,  
3     "value2": null,  
4     "text1": null,  
5     "text2": "hello",  
6     "intValue": 0,  
7     "myList": [],  
8     "myEmptyList": null,  
9     "boolean1": null,  
10    "littleboolean": false  
11 }
```

Biar skill kamu makin ajib, sekarang kita bakal belajar sambil praktik, ya.

Kita bakal coba membuat REST dan RESTful API dengan `@Controller` annotation.

Who's excited?!



“Cara bikin Rest dan Restful API itu gimana, ya?”

Buat bikin REST API pakai Spring, kita bakal bikin class yang berbeda sama class service.

“Lho, kok di bedain sih?”

Karena class service seharusnya cuma berisi business logic, sedangkan class controller seharusnya cuma berisi cara menghandle suatu request.



Sebuah class controller berisi mapping terdiri dari:

- endpoint,
- HTTP method dari endpoint,
- gimana menghandle Request dan Response

Selain itu, class controller ditandai dengan annotation `@Controller` yang merupakan turunan dari annotation `@Component`



Buat REST API kita bisa pakai annotation @RestController yang lebih spesifik, lho~

@RestController ini nggak bisa dipakai untuk mengolah servlet, jadi cuma REST API aja yang diolah pakai annotation ini.

Terus, supaya controller ini terhubung sama class service, kita cuma perlu meng-inject class service sebagai dependency di class controller ini.



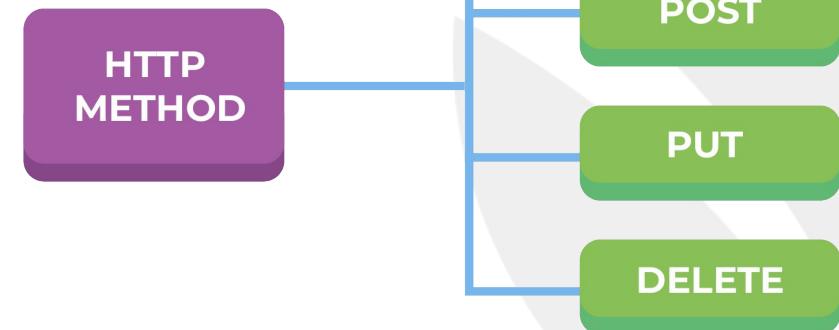
Nantinya endpoint-endpoint yang dibikin bakal berbentuk suatu method dengan access modifier public.

Gimana menurut kamu? Nggak ribet kan, sobat?



Buat menentukan HTTP method yang dipakai,
kita bakal pakai annotation berikut, gengs~

1. `@GetMapping`, buat GET
2. `@PostMapping`, buat POST
3. `@PutMapping`, buat PUT
4. `@DeleteMapping`, buat DELETE



Annotation yang dipakai di method ini nantinya bakal jadi endpoint.

Selain itu, parameter dari method tadi bakal jadi body, header, parameter atau path variable, gengs~

Parameter tersebut menggunakan annotation berikut:

1. `@RequestParam`, untuk membuat parameter pada endpoint.
2. `@RequestBody`, untuk membuat content body.
3. `@PathVariable`, kalau ada suatu URI path yang jadi variable.
4. `@Request Header`, untuk menambahkan header.



Return type dari method tersebut merupakan response dari endpoint, lho~

Kita bisa pakai object secara langsung atau mengembalikan object berupa ResponseEntity.

Kalau kita pakai Response Entity, kita bisa bebas buat bikin response. Selain itu, kita juga bisa mengatur HTTP status dan content jadi lebih gampang.

Contoh pembuatan REST API buat aplikasi CRUD bisa kamu cek [di sini](#), ya!

```
● ● ●

@RestController
@RequestMapping("/user-management")
public class UserController {

    @Autowired
    private UserService userService;

    @PostMapping ("add-user")
    public ResponseEntity<Object> addUser(
        @RequestBody List<String> users,
        @RequestHeader(value="Authorization") String auth
    ){
        ResponseModel responseModel = new ResponseModel();
        try {
            userService.addUsers(users);
            responseModel.setResponseCode(ResponseConstant.CODE_SUCCESS);
            responseModel.setResponseMessage(ResponseConstant.MESSAGE_SUCCESS);
            return new ResponseEntity<>(responseModel, HttpStatus.OK);
        }catch (Exception e) {
            responseModel.setResponseCode(ResponseConstant.CODE_FAILED);
            responseModel.setResponseMessage(ResponseConstant.MESSAGE_FAILED);
            responseModel.setData(e.getMessage());
        }
        return new ResponseEntity<>(responseModel, HttpStatus.INTERNAL_SERVER_ERROR);
    }

    @DeleteMapping ("remove-user")
    public ResponseEntity<Object> deleteUser(
        @RequestParam String user,
        @RequestHeader(value="Authorization") String auth
    ){
        ResponseModel responseModel = new ResponseModel();
        try {
            userService.deleteUser(user);
            responseModel.setResponseCode(ResponseConstant.CODE_SUCCESS);
            responseModel.setResponseMessage(ResponseConstant.MESSAGE_SUCCESS);
            responseModel.setData(user);
            return new ResponseEntity<>(responseModel, HttpStatus.OK);
        }catch (Exception e) {
            responseModel.setResponseCode(ResponseConstant.CODE_FAILED);
            responseModel.setResponseMessage(ResponseConstant.MESSAGE_FAILED);
            responseModel.setData(e.getMessage());
        }
        return new ResponseEntity<>(responseModel, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Setelah tahu konsep dan cara bikinnya, sekarang kita bakal mulai melakukan **API Testing**.

Salah satu caranya yaitu kita perlu pakai **POSTMAN**.



Apa itu POSTMAN?

POSTMAN adalah sebuah **aplikasi yang berfungsi sebagai REST Client untuk uji coba REST API**.

POSTMAN biasa digunakan oleh developer API sebagai tools untuk menguji API yang sudah dibuat. Selain itu bisa juga menjadi tools untuk melakukan proses development API.



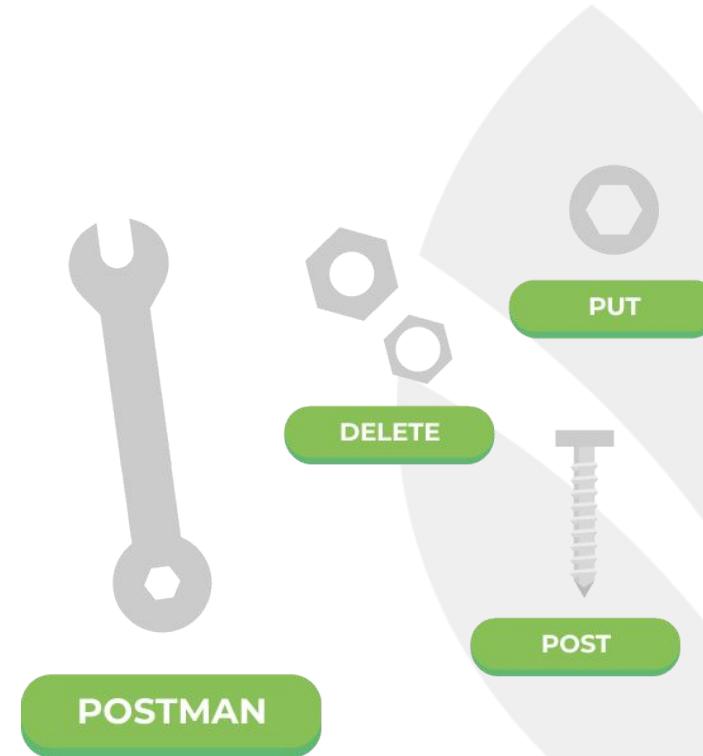
POSTMAN

“Eh tapi kenapa perlu pakai POSTMAN ya?”

Gini, kalau kamu mau mengakses RESTful API dengan metode GET, kamu bisa melakukannya langsung lewat browser aja.

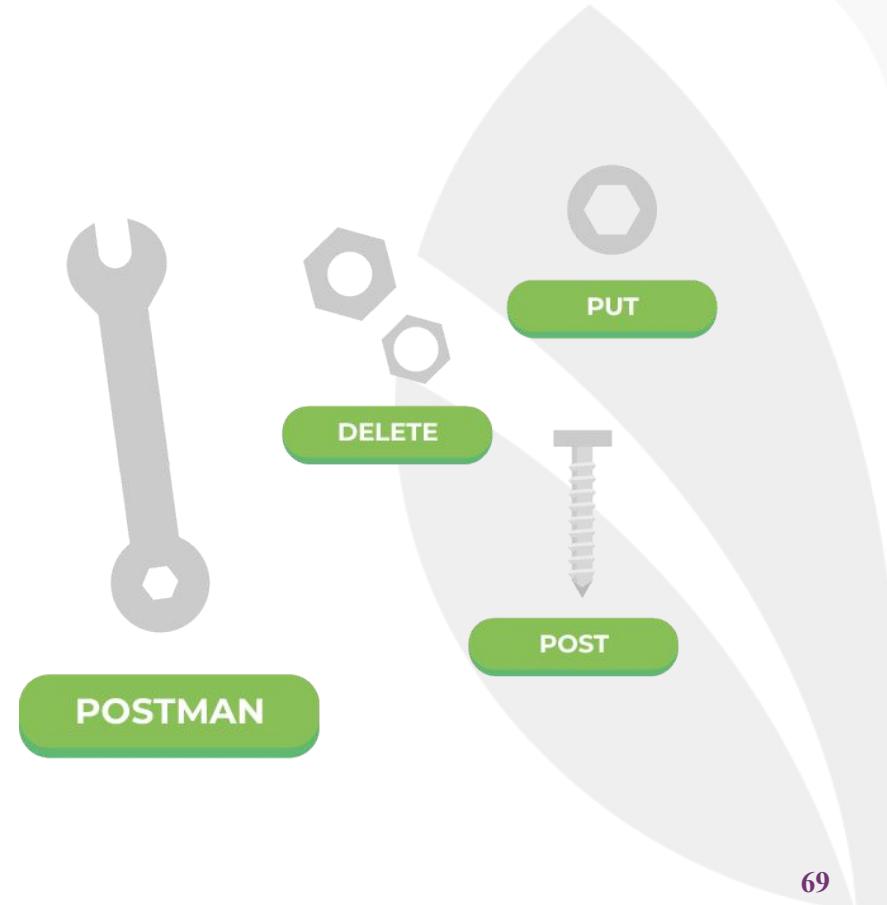
Tapi.. kalau kamu mau melakukan operasi dengan metode lain seperti POST, PUT, dan DELETE, kamu harus pakai sebuah tool REST Client.

Salah satu tools pada kategori REST Client ini adalah **POSTMAN**.

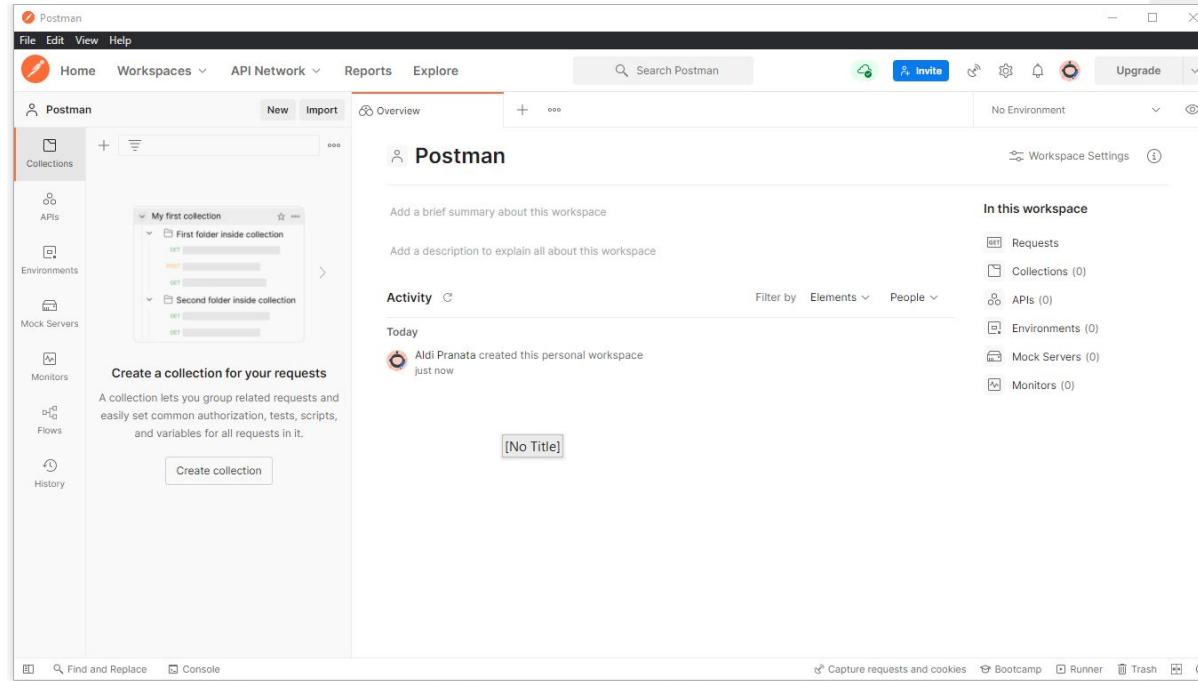


Dari situ kita jadi tahu deh kalau melalui POSTMAN, kamu bisa mengupdate dan menghapus data pada sebuah server melalui RESTful API.

Mantul banget kan, bestie?



Biar kamu kebayang, begini tampilan aplikasi POSTMAN~

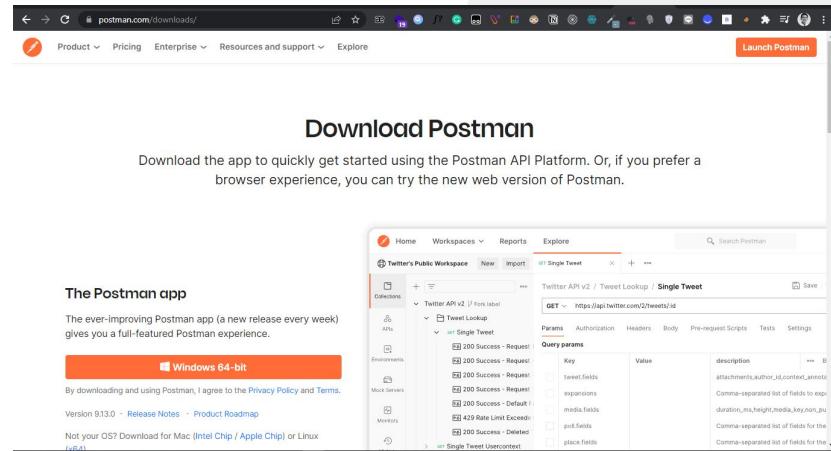


“Terus, gimana caranya nih supaya bisa mengunduh POSTMAN?”

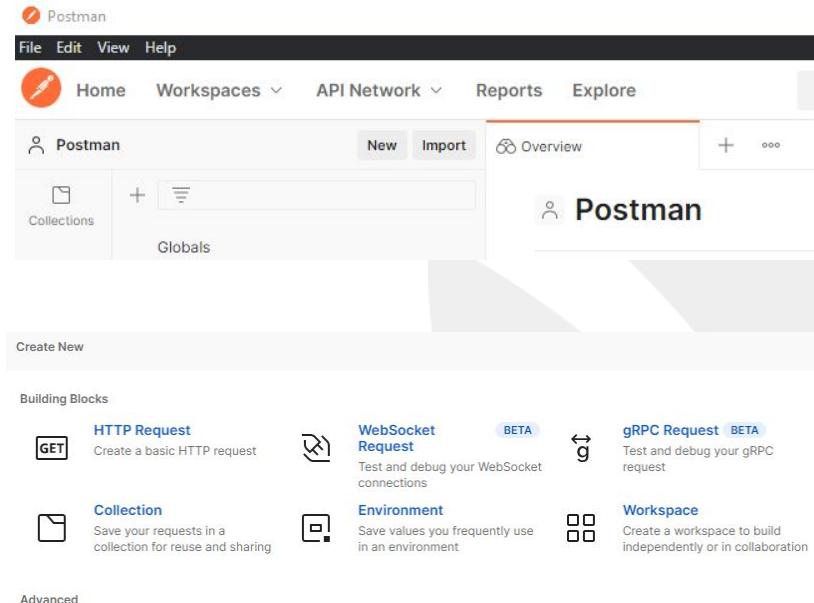
Gampang banget, gengs!

Untuk mengunduh aplikasi POSTMAN ini, kamu bisa langsung download di situs web resmi POSTMAN [di sini](#) ya!

Cara instalasinya juga gampang banget kok. Setelah filenya ter-download, kamu tinggal klik tombol next, next, next aja deh~



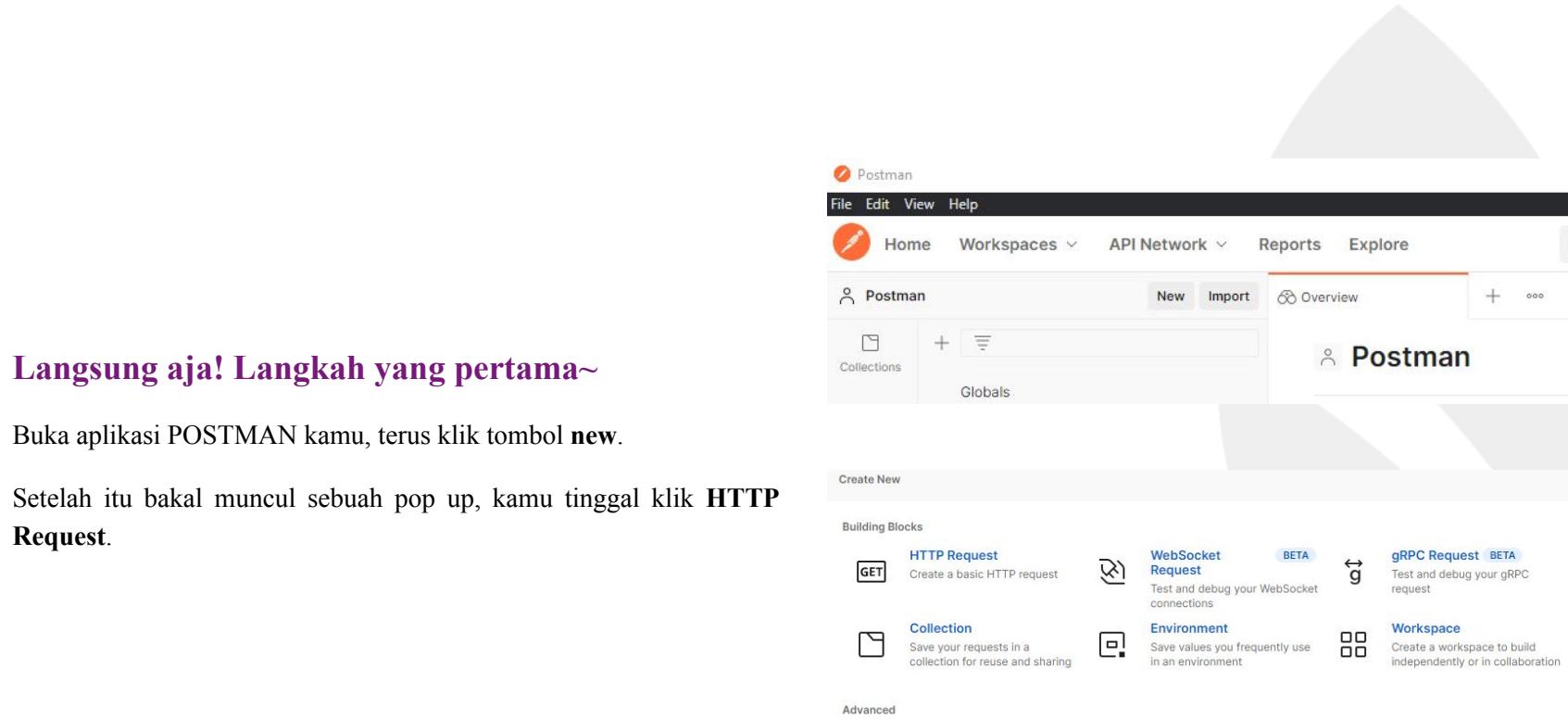
The screenshot shows the Postman website at postman.com/downloads/. The main heading is "Download Postman". Below it, a sub-headline says "Download the app to quickly get started using the Postman API Platform. Or, if you prefer a browser experience, you can try the new web version of Postman." On the right, there's a large screenshot of the Postman application interface. The interface shows a sidebar with "Twitter API v2 / Tweet Lookup" selected under "Collections". In the main pane, there's a "Single Tweet" request with a "GET" method and the URL <https://api.twitter.com/2/tweets/1d>. The "Params" tab is active, showing query parameters like "id", "expansions", "media.fields", "place.fields", and "tweet.fields". The "Headers" tab shows "Content-Type: application/json". The "Body" tab contains JSON data for a tweet lookup. At the bottom left of the screenshot, there's a large orange button labeled "Windows 64-bit".



“Kalau cara pakainya, gimana dong?”

Yaudah, yuk langsung aja kita sambil belajar eksekusi.

Kita bakal coba bikin sebuah request pakai method GET buat menampilkan data Gempa BMKG.



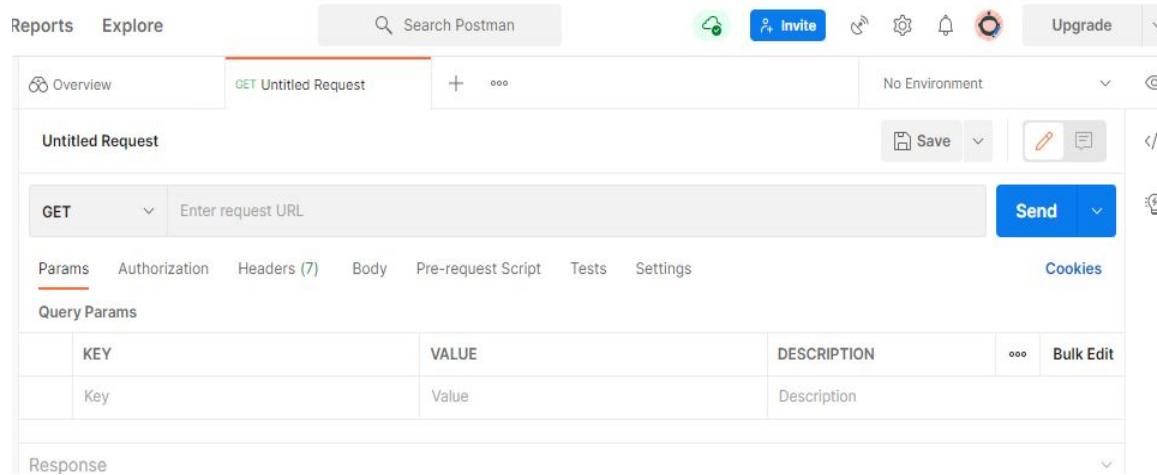
Langsung aja! Langkah yang pertama~

Buka aplikasi POSTMAN kamu, terus klik tombol **new**.

Setelah itu bakal muncul sebuah pop up, kamu tinggal klik **HTTP Request**.

Langkah keduaaa~

Selanjutnya bakal muncul sebuah tab baru pada POSTMAN. Di tampilan inilah kamu bakal melakukan request data ke server~



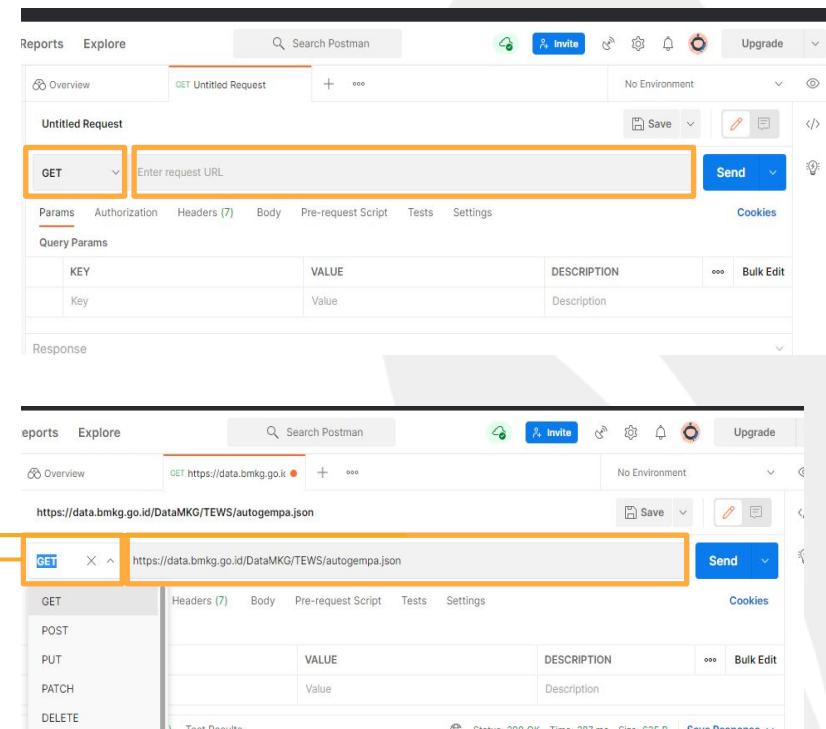
The screenshot shows the Postman application interface. At the top, there are tabs for 'Reports' and 'Explore', a search bar, and various icons for account management and upgrades. Below the header, the main workspace is titled 'Untitled Request'. It displays a 'GET' method and an empty 'Enter request URL' field. To the right of the URL field are 'Save' and 'Send' buttons. Below the method and URL, there are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Params' tab is currently selected, showing a table for 'Query Params'. The table has columns for 'KEY', 'VALUE', 'DESCRIPTION', and 'Bulk Edit'. A single row is present with the key 'Key' and the value 'Value'. At the bottom of the workspace, there is a section labeled 'Response'.

Lanjut ke langkah yang ketiga, guys!

Yaitu mengisi URL dan memilih HTTP method pada POSTMAN. Begini caranya:

1. Pilih method GET, karena kamu mau mendapatkan data dari BMKG
2. Isi ENDPOINT dan PATH pada bagian kotak ‘Enter Request URL’. Masukan ENDPOINT dan PATH nya yaitu:

<https://data.bmkg.go.id/DataMKG/TEWS/autogempa.json>



The screenshot shows two instances of the Postman application interface. Both instances have the following configuration:

- Method:** GET (highlighted with an orange arrow pointing from the list above)
- URL:** https://data.bmkg.go.id/DataMKG/TEWS/autogempa.json (highlighted with an orange box)
- Headers:** (7) (highlighted with an orange arrow pointing from the list above)
- Body:** (empty)
- Pre-request Script:** (empty)
- Tests:** (empty)
- Settings:** (empty)

Langkah keempat, alias langkah terakhir!

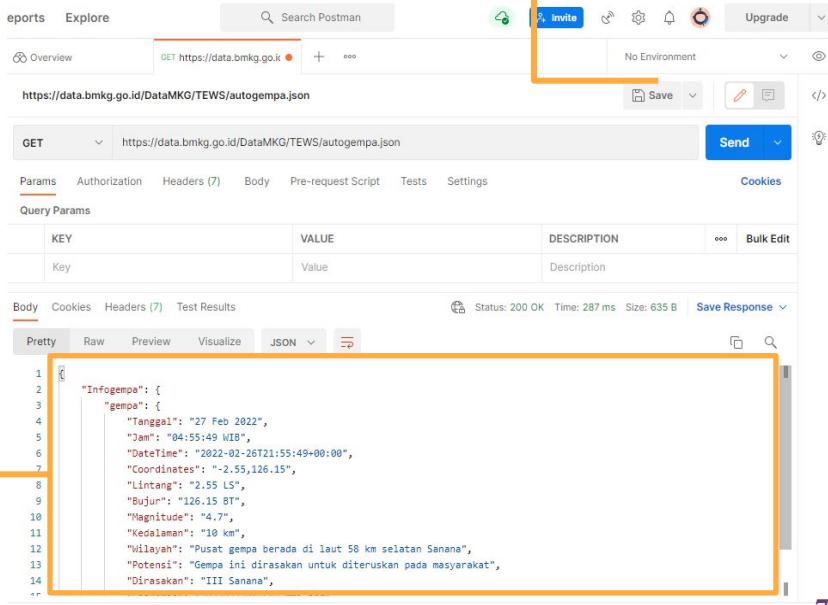
Setelah itu, kamu klik tombol “Send”. Nantinya POSTMAN bakal mengirimkan request untuk membaca info gempa terbaru ke server.

Setelah proses request selesai, server bakal mengirimkan data yang udah di-request tadi, yaitu data Info gempa terbaru.

Pada bagian bawah POSTMAN adalah data yang diterima dan ditampilkan dari server milik BMKG.

Response data dari POSTMAN

Pilih method GET, karena kita ingin mendapatkan data dari BMKG



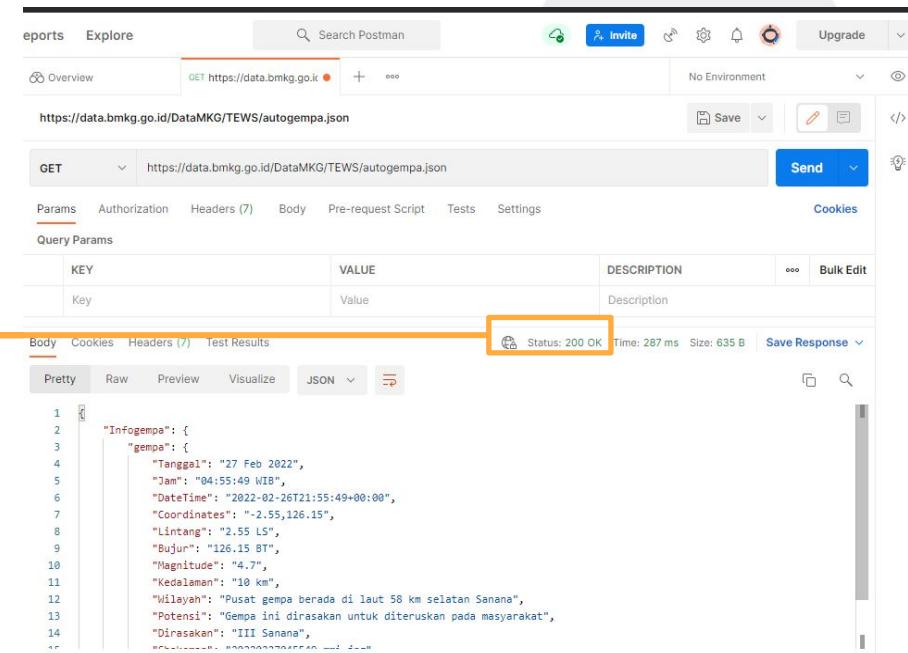
The screenshot shows the Postman interface with a GET request to `https://data.bmkg.go.id/DataMKG/TEWS/autogempa.json`. The response status is 200 OK, and the response body is a JSON object:

```
1  {
2   "Infogempa": [
3     "gempa": [
4       {
5         "Tanggal": "27 Feb 2022",
6         "Jam": "04:55:49 WIB",
7         "Datetime": "2022-02-26T21:55:49+00:00",
8         "Coordinates": "-2.55,126.15",
9         "Lintang": "2.55 LS",
10        "Bujur": "126.15 BT",
11        "Magnitude": "4.7",
12        "Kedalaman": "10 km",
13        "Wilayah": "Pusat gempa berada di laut 58 km selatan Sanana",
14        "Potensi": "Gempa ini dirasakan untuk diteruskan pada masyarakat",
15        "Dirasakan": "III Sanana"
16      }
17    ]
18  ]
```

HTTP Status Code

Kalau kamu perhatikan, selain menampilkan Response data dari server, POSTMAN juga menampilkan kode response status, lho.

Jadi apa sih kode response status ini?



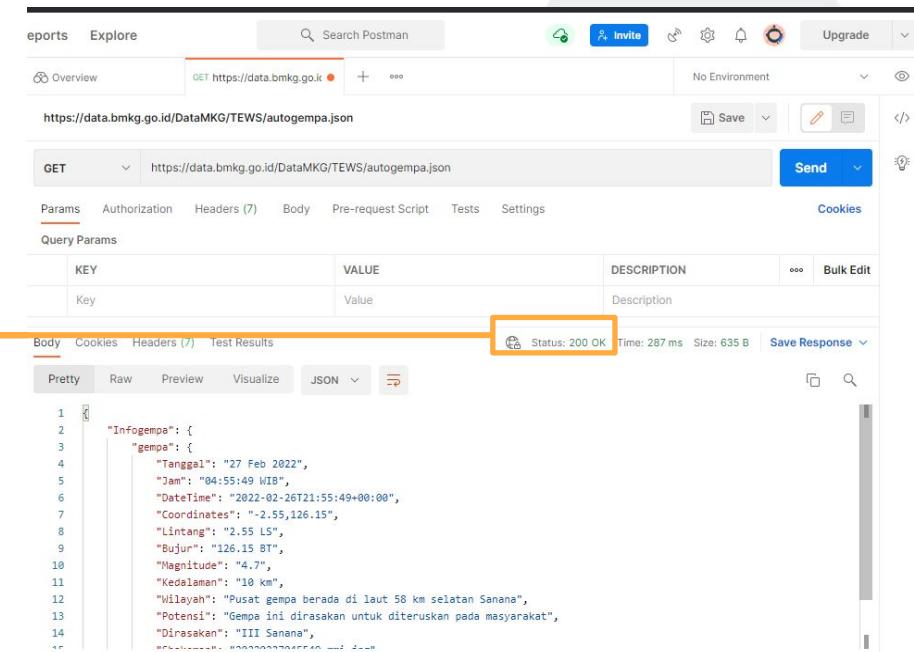
The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Reports', 'Explore', a search bar ('Search Postman'), and various icons. Below the bar, a tab labeled 'Overview' is selected, showing the URL 'https://data.bmkg.go.id/DataMKG/TEWS/autogempa.json'. The main area has a 'GET' method selected and the same URL. Below the method, there are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. Under 'Body', there's a 'Query Params' table with columns 'KEY', 'VALUE', and 'DESCRIPTION'. The 'Value' column contains 'Key' and 'Value'. At the bottom of the interface, there's a status bar with 'Status: 200 OK', 'Time: 287 ms', 'Size: 635 B', and a 'Save Response' button. The 'Body' tab is currently active, displaying the JSON response:

```
1 "Infogempa": {  
2     "gempa": {  
3         "Tanggal": "27 Feb 2022",  
4         "Jam": "04:55:49 WIB",  
5         "DateTime": "2022-02-26T21:55:49+00:00",  
6         "Coordinates": "-2.55,126.15",  
7         "Lintang": "2.55 Lt",  
8         "Bujur": "126.15 Bt",  
9         "Magnitude": "4.7",  
10        "Kedalaman": "10 km",  
11        "Wileayah": "Pusat gempa berada di laut 58 km selatan Sanana",  
12        "Potensi": "Gempa ini dirasakan untuk diteruskan pada masyarakat",  
13        "Dirasakan": "III Sanana",  
14        "Rasakan": "Rasakan gempa di Sanana"}
```

Kode responses status ini adalah salah satu **indikator** yang menandakan kalau request kamu **berhasil** atau ketika dalam proses request ke server ada kendala **error**.

Singkatnya, Kode HTTP Status adalah response server atas request data yang kamu lakukan.

Kode response status ini berbentuk kode tiga digit dan biasanya kamu nggak sadar kalau proses pertukaran data berjalan dengan lancar.



The screenshot shows the Postman application interface. A search bar at the top right contains the text "Search Postman". Below it, a navigation bar includes "Reports", "Explore", and a search field. A tab labeled "Overview" is active, showing a red error icon and the URL "GET https://data.bmkg.go.id". To the right, there are buttons for "invite", "refresh", "gear", "bell", and "upgrade". A large grey triangle graphic is visible on the right side of the screen.

The main workspace displays a GET request to "https://data.bmkg.go.id/DataMKG/TEWS/autogempa.json". The "Params" tab is selected. In the "Query Params" table, there is one row with "Key" and "Value" columns. The "Body" tab is active, showing the JSON response:

```
1 "Infogempa": {  
2     "gempa": {  
3         "Tanggal": "27 Feb 2022",  
4         "Jam": "04:55:49 WIB",  
5         "DateTime": "2022-02-26T21:55:49+00:00",  
6         "Coordinates": "-2.55,126.15",  
7         "Lintang": "2.55 S",  
8         "Bujur": "126.15 E",  
9         "Magnitude": "4.7",  
10        "Kedalaman": "10 km",  
11        "Wileayah": "Pusat gempa berada di laut 58 km selatan Sanana",  
12        "Potensi": "Gempa ini dirasakan untuk diteruskan pada masyarakat",  
13        "Dirasakan": "III Sanana",  
14        "Rasakan": "Rasakan gempa di Sanana" } }
```

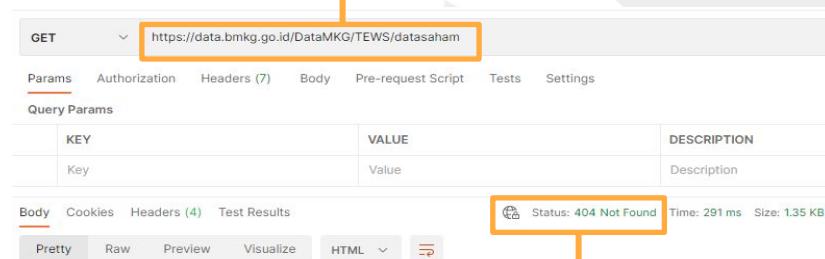
Akses path /DataMKG/TEWS/datasaham pada server BMKG, yang mana BMKG tidak menyediakan data info saham

Dalam HTTP Status Code, ada tiga macam kode yang umum dipakai~

Simak baik-baik yaa, supaya kamu nggak salah mengartikan~

1. 2xx – Kode Berhasil

Kalau http status code kamu diawali sama angka 2, berarti request kamu terbilang sukses tanpa ada kendala error.



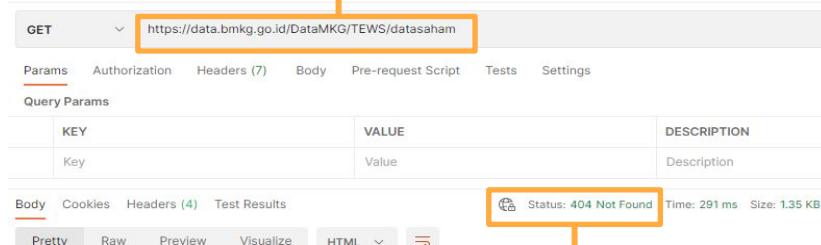
Request akan gagal, dan server akan memberikan response error berawalan dengan angka 4, yang artinya kesalahan dari sisi kamu yang meminta info data yang tidak ada

2. 4xx – Kode Error karena kesalahan ketika request

Error kode yang berawalan angka 4 ini maksudnya adalah error kategori yang disebabkan oleh request yang kamu buat.

Misalnya, kamu mau mengakses data informasi saham di server BMKG. Pas kamu mengakses data tersebut, BMKG bakal memberi respon Error 4xx karena BMKG nggak pernah punya data informasi saham.

Akses path /DataMKG/TEWS/datasaham pada server BMKG, yang mana BMKG tidak menyediakan data info saham

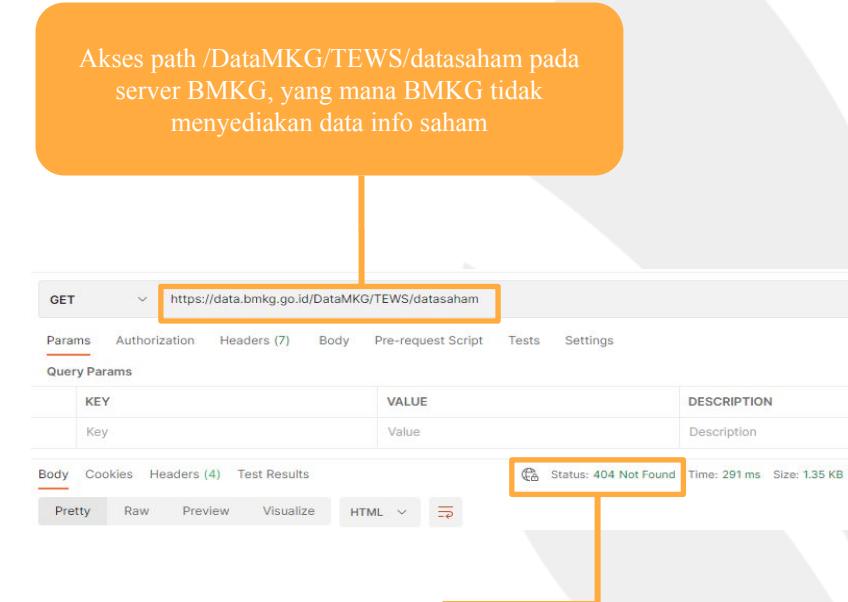


Request akan gagal, dan server akan memberikan response error berawalan dengan angka 4, yang artinya kesalahan dari sisi kamu yang meminta info data yang tidak ada

3. 5xx – Kode Error karena server lagi error

Error kode yang berawalan dengan angka 5 adalah error yang disebabkan ada kesalahan dari sisi server.

Misalnya servernya lagi mati, sob.



Akses path /DataMKG/TEWS/datasaham pada server BMKG, yang mana BMKG tidak menyediakan data info saham

Request akan gagal, dan server akan memberikan response error berawalan dengan angka 4, yang artinya kesalahan dari sisi kamu yang meminta info data yang tidak ada

Lanjut ya~ Berikut adalah cara bikin Request POST, PUT, DELETE dengan POSTMAN!

Sekarang kamu bakal coba pakai server lain buat melakukan proses Pembuatan Data (POST), mengupdate Data (PUT) dan menghapus data (DELETE).

Kamu bakal pakai server yang punya Restful API buat bikin to do list dan pakai ENDPOINT server code.aldipee.com buat melakukan semua operasi method request (GET, POST, PUT, dan DELETE).



Eittss tapi..

Kamu kan udah tahu nih, bakal pake server/endpoint code.aldipee.com buat bikin data, hapus data, update data, dan baca data, tapi ada satu catatan.

Masalahnya kamu udah tahu belum sama Path apa yang bakal kamu pakai buat mengakses data-data pada server/endpoint code.aldipee.com?

Dan di mana ya kamu bisa dapetin informasi Path yang tersedia di server code.aldipee.com tersebut?

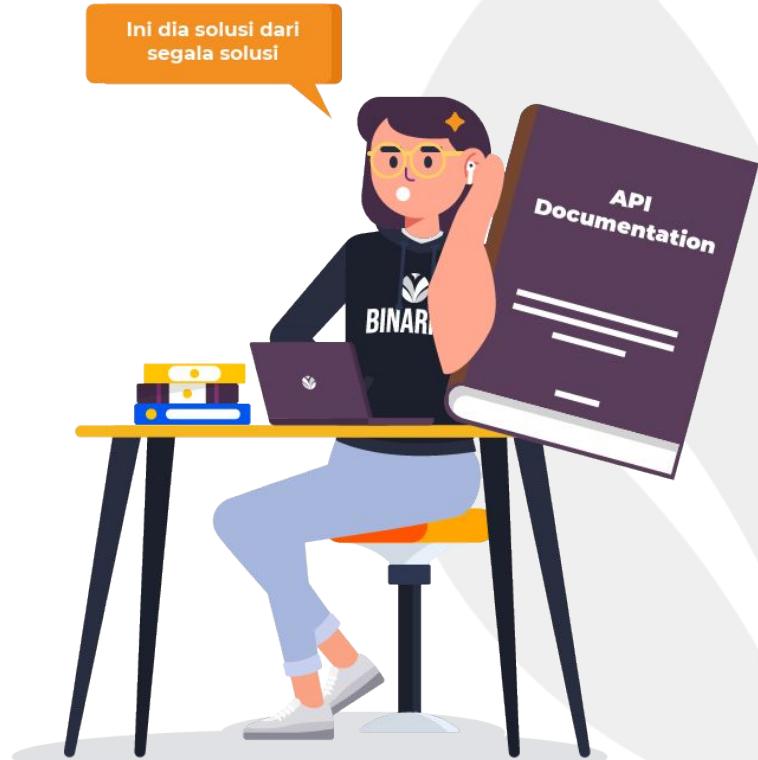


Jawabannya adalah..

API Documentation/Dokumentasi API!

Jadi setiap RESTful API itu pasti ada yang namanya dokumen dokumentasi.

Nah, di dalam dokumen itulah nanti kamu bisa mendapatkan banyak informasi yang kamu butuhkan, berhubungan dengan Path yang akan kamu pakai.



**Namanya juga dokumentasi, ya pasti
fungsinya buat memberikan informasi, dong~**

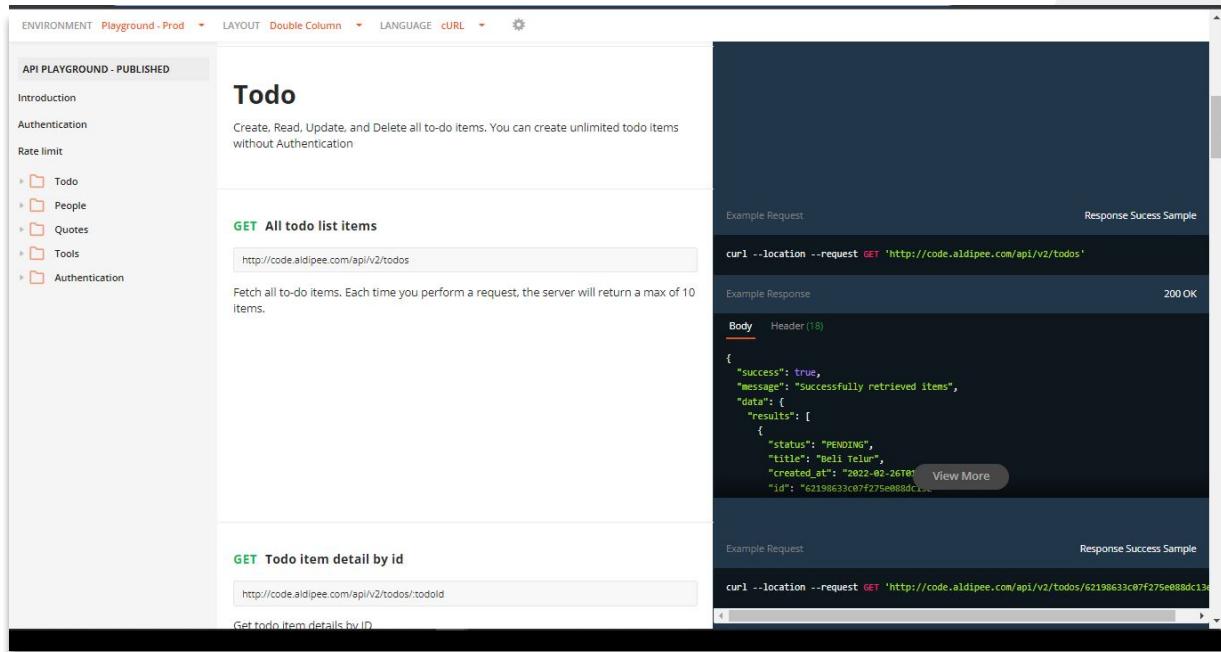
Jangan dianggurin.

Sama kayak kalau kamu mau bikin kue brownies, kamu harus lihat dokumentasi resepnya dulu buat tahu bahan-bahan apa aja yang kamu butuhkan.

RESTful API juga begitu. Kamu **harus lihat dokumentasinya dulu** buat dapat **PATH** yang mau kamu gunakan buat mengakses data tertentu.



Buat server code.aldipee.com, kamu bisa lihat dokumen dokumentasinya di [link ini](#), gengs~



The screenshot shows the API Playground interface for a Todo API. On the left sidebar, under the 'Todo' category, there are links for 'Todo', 'People', 'Quotes', 'Tools', and 'Authentication'. The main content area displays two API endpoints:

- GET All todo list items**:
Description: Create, Read, Update, and Delete all to-do items. You can create unlimited todo items without Authentication.
Example Request: curl --location --request GET 'http://code.aldipee.com/api/v2/todos'
Response Success Sample:

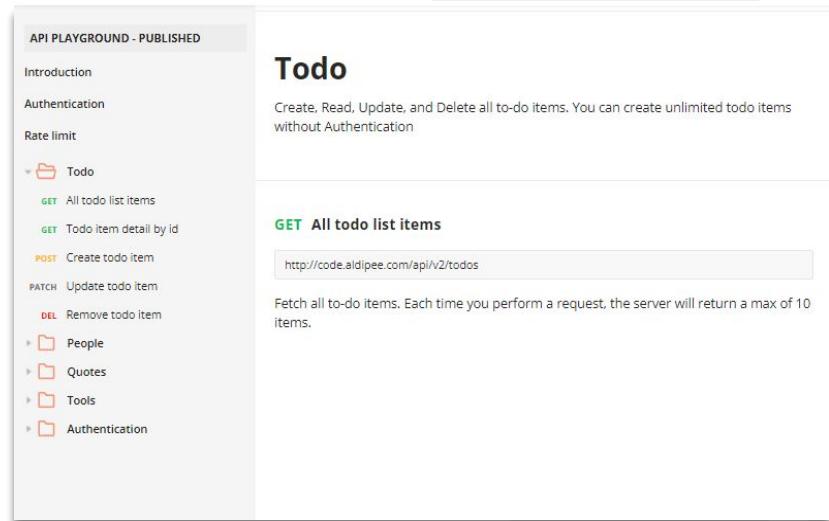
```
curl --location --request GET 'http://code.aldipee.com/api/v2/todos'
[{"id": "62198633c07f275e088dc13", "title": "beli telur", "status": "PENDING", "created_at": "2022-02-26T01:23:45Z"}]
```
- GET Todo item detail by id**:
Description: Get todo item details by ID.
Example Request: curl --location --request GET 'http://code.aldipee.com/api/v2/todos/62198633c07f275e088dc13'
Response Success Sample:

```
curl --location --request GET 'http://code.aldipee.com/api/v2/todos/62198633c07f275e088dc13'
{"id": "62198633c07f275e088dc13", "title": "beli telur", "status": "PENDING", "created_at": "2022-02-26T01:23:45Z"}
```

Perhatikan gambar di samping deh..

Karena kamu mau bikin To-do app, berarti **kamu perlu buka tab di sebelah kiri pada document dokumentasi**, yang berhubungan sama data to-do app, ya.

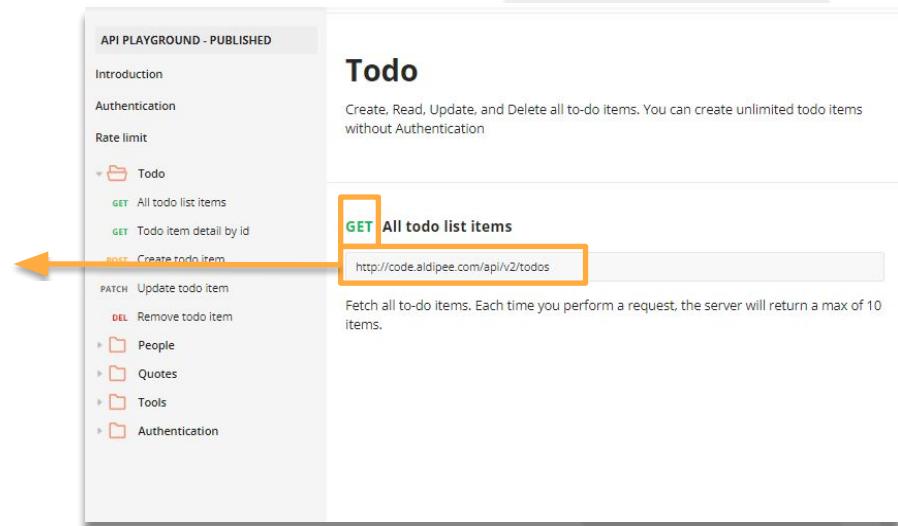
Di folder '**Todo**' tersebut, ada beberapa operasi yang bisa kamu lakukan beserta method HTTP-nya.



The screenshot shows the API PLAYGROUND - PUBLISHED interface. On the left, there's a sidebar with navigation links: Introduction, Authentication, Rate limit, Todo (selected), People, Quotes, Tools, and Authentication. Under the Todo section, there are four operations listed: GET All todo list items, GET Todo item detail by id, POST Create todo item, and PATCH Update todo item. Below these, there are three more sections: DEL Remove todo item, People, Quotes, and Tools. On the right, the main content area is titled 'Todo' and describes it as a place to 'Create, Read, Update, and Delete all to-do items. You can create unlimited todo items without Authentication'. It includes a 'GET All todo list items' section with a URL input field containing 'http://code.alpinee.com/api/v2/todos' and a note about fetching up to 10 items at a time.

Kira-kira apa yang bisa kamu lihat dari gambar disamping ini?

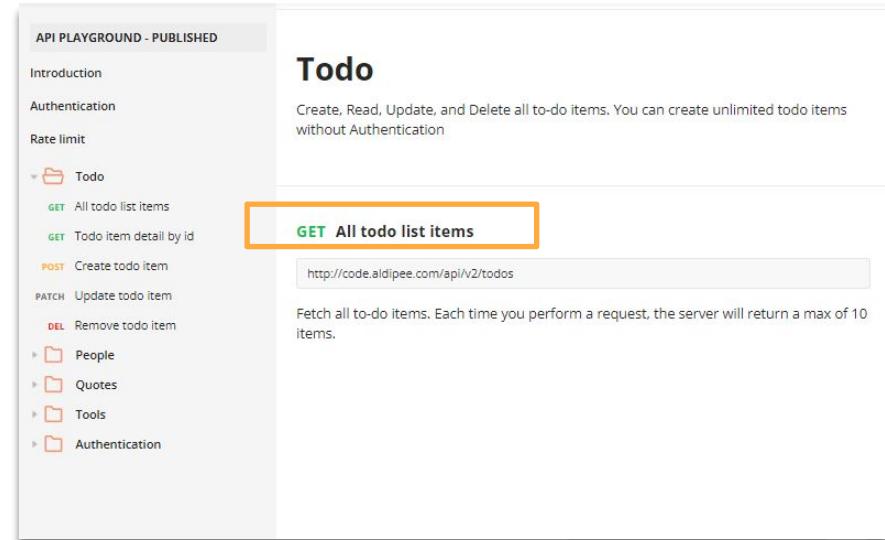
Dari dokumentasi ini kamu bisa tahu jika kamu mau mengakses semua data to-do list, maka kamu harus pakai path **/api/v2/todos** dan pake method **GET**.



“Selanjutnya gimana?”

Buat akses semua data to-do list, pada dokumentasi bakal di kasih tahu kalau kamu harus pakai method **GET**.

Sekarang kita coba akses yuk datanya melalui **POSTMAN**~



The screenshot shows a web-based API documentation tool. On the left, there's a sidebar with navigation links: 'API PLAYGROUND - PUBLISHED', 'Introduction', 'Authentication', and 'Rate limit'. Below these are sections for 'Todo', 'People', 'Quotes', 'Tools', and 'Authentication', each with a small folder icon. Under the 'Todo' section, there are several API endpoints listed with their methods and descriptions:

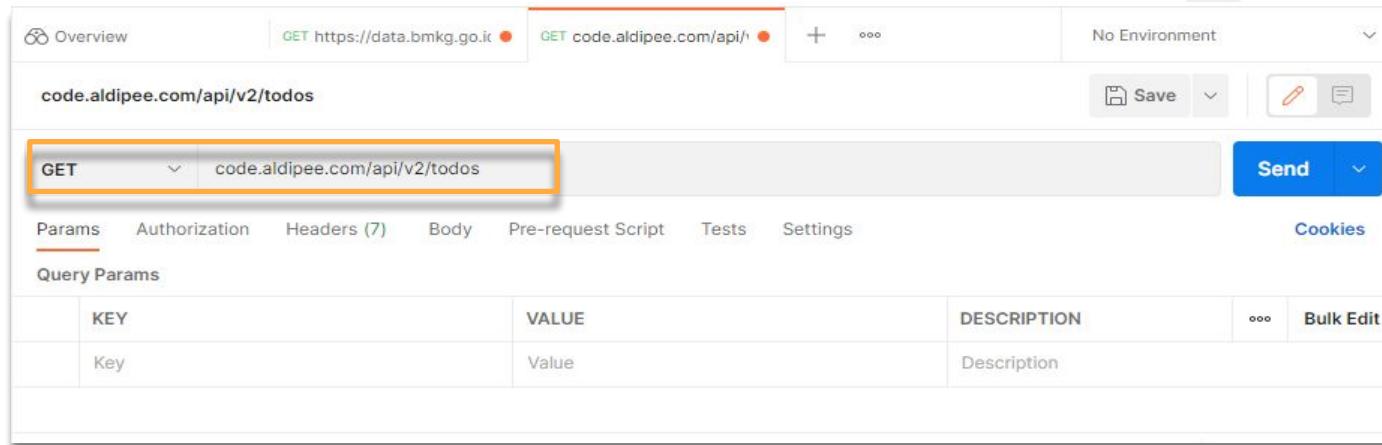
- Todo**
 - GET All todo list items**: Fetch all to-do items. Each time you perform a request, the server will return a max of 10 items.
 - GET Todo item detail by id**
 - POST Create todo item**
 - PATCH Update todo item**
 - DEL Remove todo item**
- People**
- Quotes**
- Tools**
- Authentication**

A specific endpoint, **GET All todo list items**, is highlighted with an orange border. Below it, the URL `http://code.aldipee.com/api/v2/todos` is shown in a text input field. The main content area has a heading 'Todo' and a sub-instruction: 'Create, Read, Update, and Delete all to-do items. You can create unlimited todo items without Authentication'.

Berikut adalah langkah mengakses data melalui POSTMAN!

Masukan ENDPOINT-nya yaitu code.aldipee.com, lalu masukan juga Path yang kamu dapatkan dari dokumen dokumentasi tadi, yaitu /api/v2/todos.

Selanjutnya, pilih method sesuai dengan yang ada dokumentasi tadi yaitu GET, lalu tekan tombol ‘Send’.

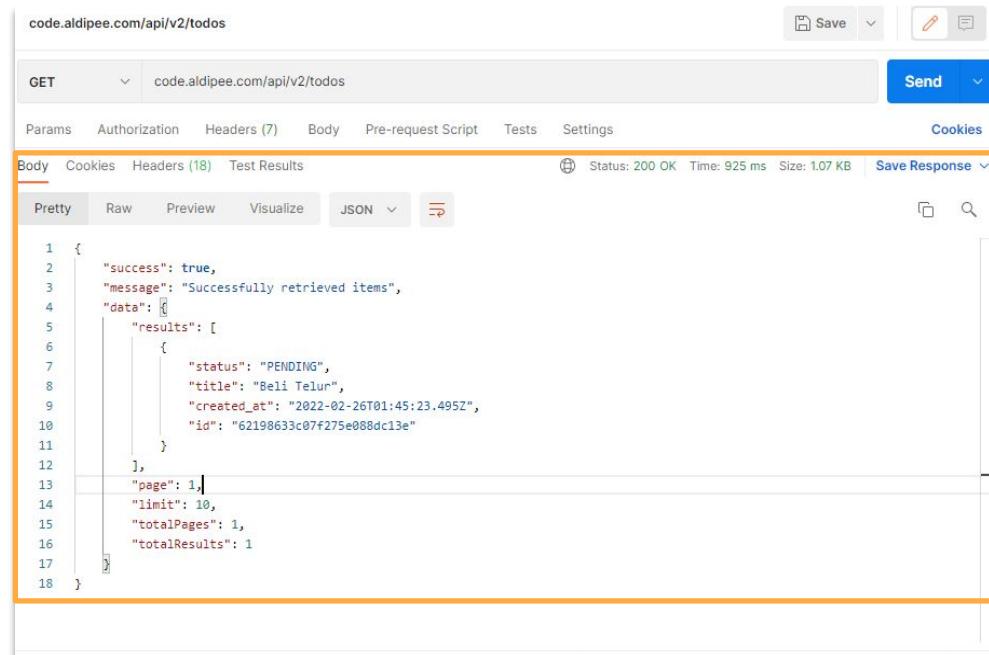


The screenshot shows the POSTMAN application interface. At the top, there are tabs for Overview, GET https://data.bmkg.go.id (selected), and GET code.aldipee.com/api/. Below these are buttons for Save, Edit, and Delete. The main area shows a GET request to code.aldipee.com/api/v2/todos. The 'Params' tab is selected, showing a table for Query Params:

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Request yang udah dibuat barusan berhasil nih, bestie.

Selanjutnya, dibawah ini ada POSTMAN yang udah memunculkan data JSON.



The screenshot shows a POSTMAN interface with the following details:

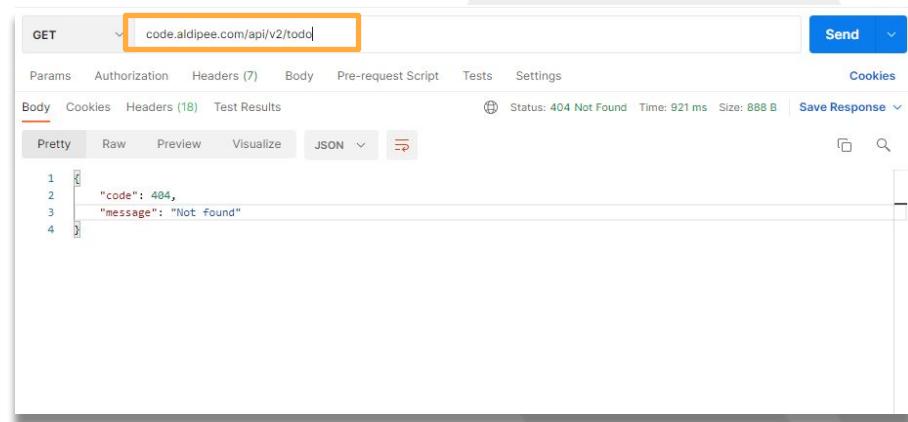
- URL: code.aldipee.com/api/v2/todos
- Method: GET
- Headers: Authorization, Headers (7), Body, Pre-request Script, Tests, Settings, Cookies
- Body tab selected
- Response status: 200 OK, Time: 925 ms, Size: 1.07 KB
- Pretty JSON output:

```
1  {
2      "success": true,
3      "message": "Successfully retrieved items",
4      "data": [
5          {
6              "results": [
7                  {
8                      "status": "PENDING",
9                      "title": "Beli Telur",
10                     "created_at": "2022-02-26T01:45:23.495Z",
11                     "id": "62198633c07f275e088dc13e"
12                 ],
13                 "page": 1,
14                 "limit": 10,
15                 "totalPages": 1,
16                 "totalResults": 1
17             }
18 }
```

Jangan salah masukin Path lho, gengs~

Misalnya di dokumentasi dijelaskan pakai `/api/v2/todos` tapi kamu malah pakai `/api/v2/todo` yang nggak ada di dokumentasi.

Kira-kira, apa yang terjadi ya kalau misalnya kamu salah masukin Path yang nggak sesuai di dokumentasi?

A screenshot of the Postman API client interface. The URL field contains `code.aldipee.com/api/v2/todo`. The response shows a 404 Not Found error with the message: "code": 404, "message": "Not found".

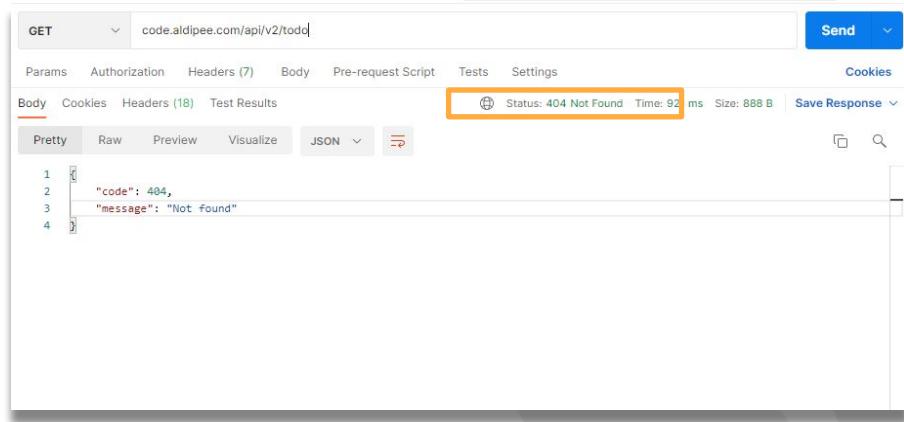
```
1 [object Object]
2 {
3     "code": 404,
4     "message": "Not found"
5 }
```

Kalau kamu salah masukin Path, maka ini nih yang bakal terjadi..

Kamu nggak bakal berhasil mendapatkan data yang kamu mau.

Sebaliknya, kamu malah bakal dapat respon status 404.

Berhubung ada respon dengan awalan angka 4, ini artinya ada error yang disebabkan oleh kesalahan kamu dalam membuat request.

A screenshot of the Postman application interface. The top bar shows a GET request to "code.aldipee.com/api/v2/todo". The "Body" tab is selected, showing a JSON response with code 404 and message "Not found". The status bar at the bottom indicates "Status: 404 Not Found Time: 92 ms Size: 888 B".

```
1  {
2   "code": 404,
3   "message": "Not found"
4 }
```

Lanjuttt!

Kita bakal bahas gimana cara membuat data pada sebuah server melalui POSTMAN.

Siap-siap ya karena kamu akan mulai melakukan request dengan menggunakan method POST.

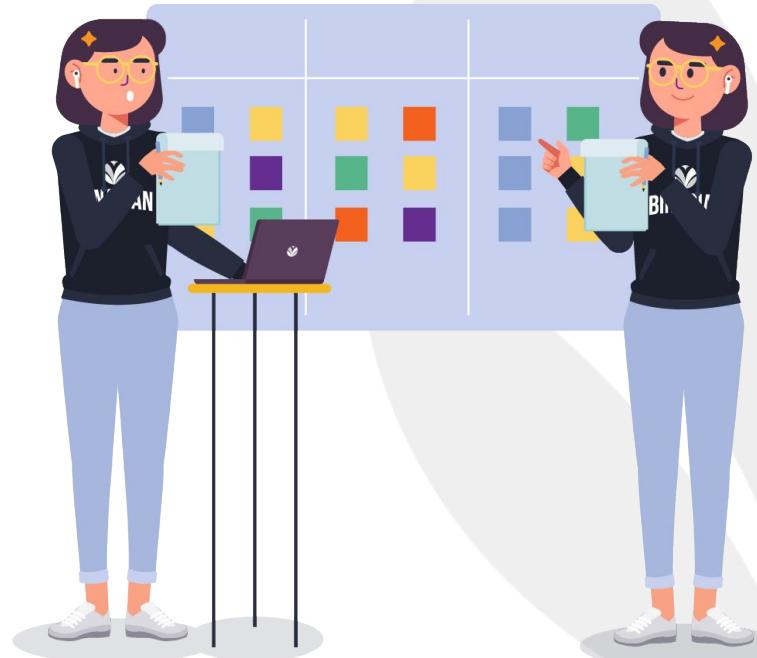
Are you ready?



Hmm.. kita mulai dari melihat document dari dokumentasinya dulu, ya!

Yepp. Sama kayak sebelumnya, kamu harus lihat dulu di document dokumentasi yang udah diberikan.

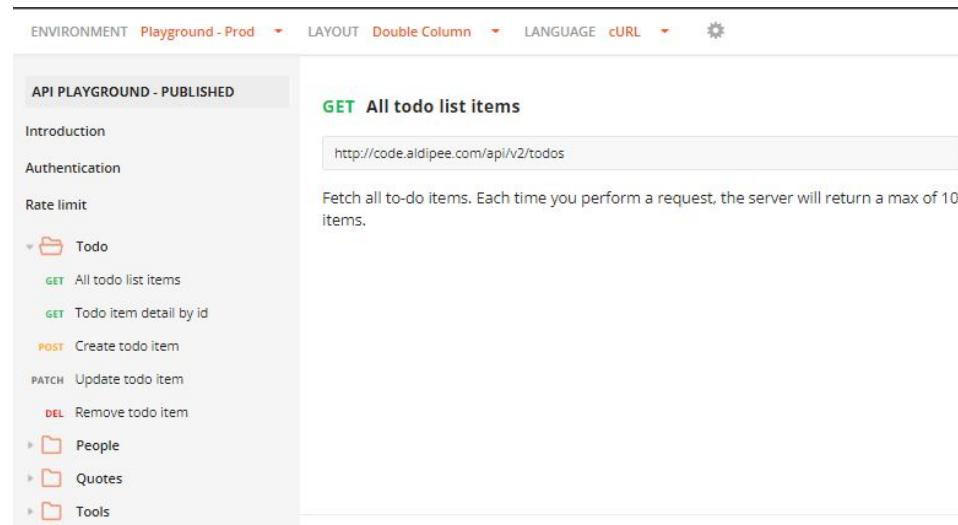
Tujuannya, supaya kamu dapat informasi berupa path dan method yang bisa dipakai untuk membuat data to-do baru pada server.



Perhatikan gambar di samping ya, lalu ikuti langkah-langkah-nya!

1. Pada bagian navigasi menu sebelah kiri, ada nama operasi yang kita mau yaitu “Create todo item”.

Klik menu tersebut ya, bestie.



The screenshot shows the API Playground interface with the following details:

- ENVIRONMENT**: Playground - Prod
- LAYOUT**: Double Column
- LANGUAGE**: CURL

API PLAYGROUND - PUBLISHED

Todo

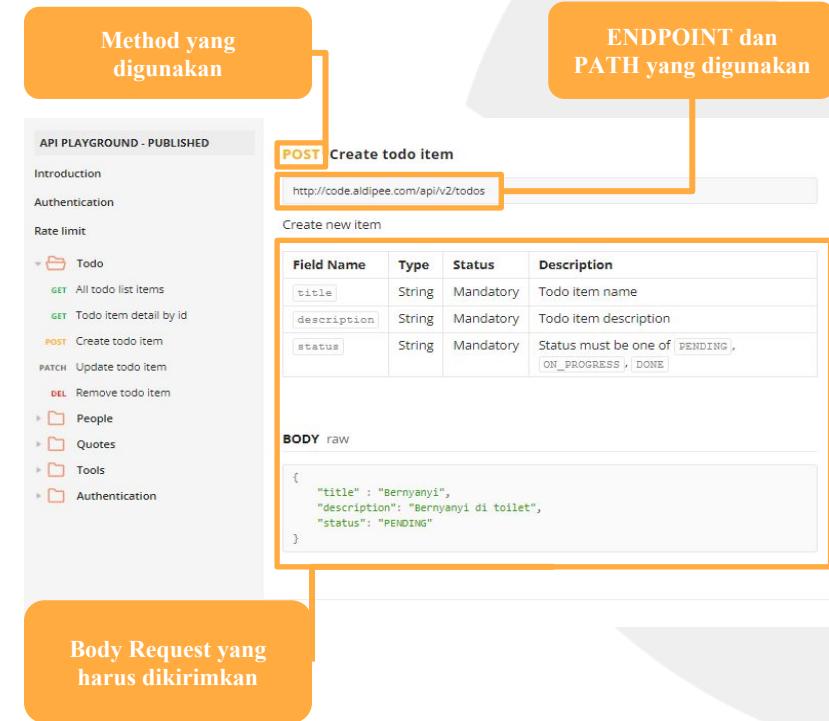
- GET All todo list items**:
HTTP URL: <http://code.aldipee.com/api/v2/todos>
Description: Fetch all to-do items. Each time you perform a request, the server will return a max of 10 items.
- POST Create todo item**
- PATCH Update todo item**
- DEL Remove todo item**

- Setelah itu bakal ada informasi berupa Path dan Method yang kamu butuhkan untuk membuat data baru pada server.

Selain itu, ada juga yang namanya body request dalam method POST. Apa itu body request?

Jadi, karena kamu mau bikin data baru pada server, maka secara otomatis kamu juga bakal **mengirimkan body request yang berisi data-data** apa yang mau kamu bikin di server.

Yuk, kita coba di POSTMAN!



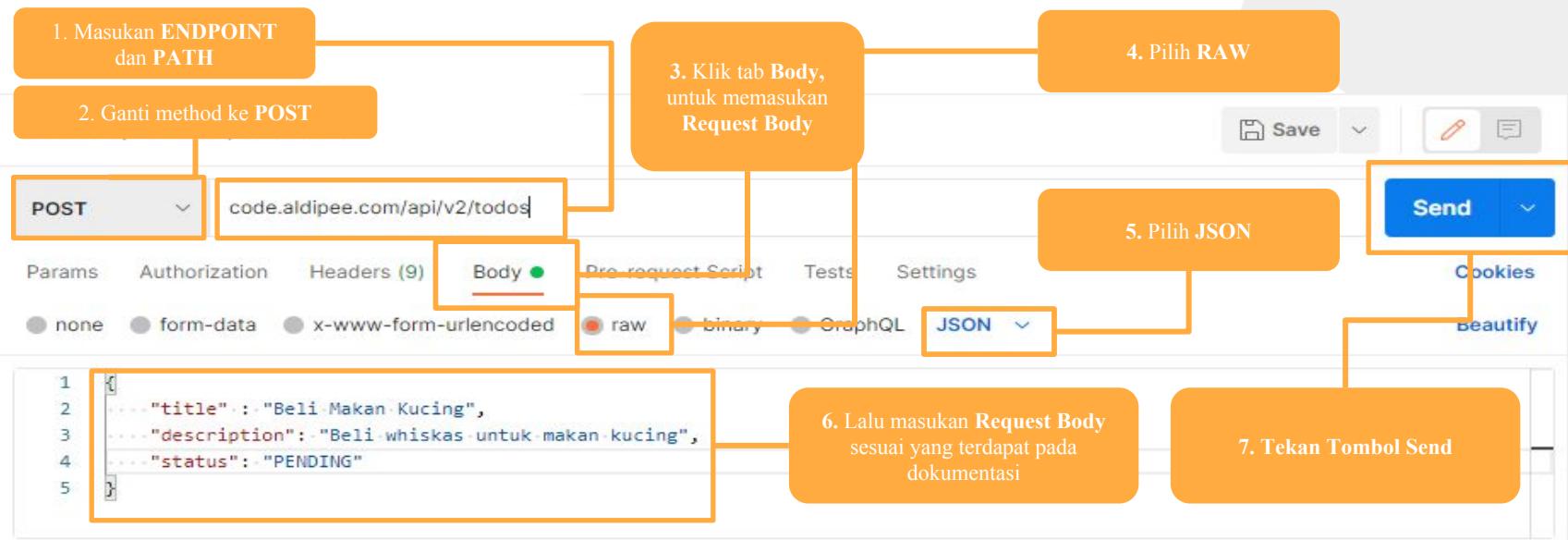
The screenshot shows the API Playground interface for a Todo API. On the left, there's a sidebar with categories like Todo, People, Quotes, Tools, and Authentication. Under Todo, there are several methods: GET All todo list items, GET Todo Item detail by id, POST Create todo item (which is highlighted in orange), PATCH Update todo item, and DEL Remove todo item. The main area shows a POST request to 'Create todo item' with the URL 'http://code.aldipee.com/api/v2/todos'. Below this, there's a 'Create new item' section with a table for field names, types, statuses, and descriptions. The 'title' field is mandatory and must be a string. The 'description' field is also mandatory and must be a string. The 'status' field is mandatory and must be one of 'PENDING', 'ON_PROGRESS', or 'DONE'. At the bottom, there's a 'BODY raw' section with a JSON object:

```
{  
    "title": "Bernyanyi",  
    "description": "Bernyanyi di toilet",  
    "status": "PENDING"  
}
```

Three callout boxes point to specific parts of the interface:

- A yellow box labeled 'Method yang digunakan' points to the 'POST' button in the request section.
- A yellow box labeled 'ENDPOINT dan PATH yang digunakan' points to the URL 'http://code.aldipee.com/api/v2/todos'.
- A yellow box labeled 'Body Request yang harus dikirimkan' points to the 'BODY raw' JSON object.

Berikut adalah langkah-langkah bikin body request!



Setelah kamu berhasil mengirimkan Request POST ke server, maka server bakal memberikan Respon berupa data yang kamu kirim tadi, gengs.

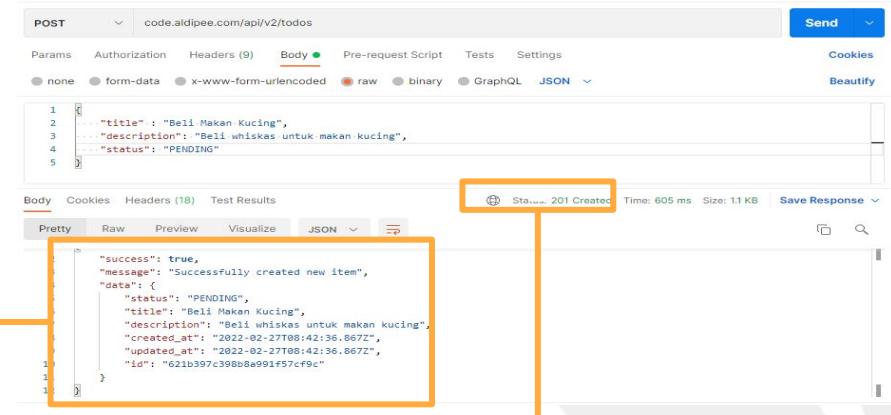


- Selanjutnya, ada message “Successfully created new Item” dan Kode Status HTTP yang diawali dengan angka 2, yang menandakan request pembuat data baru telah berhasil diterima oleh server.

Sekarang data **dengan title “Beli Makan Kucing”** telah berhasil disimpan.

Terdapat informasi response dari server yang menandakan request berhasil

HTTP Status Code diawali dengan angka 2, yang menandakan Request Berhasil



The screenshot shows a POST request to `code.alidpee.com/api/v2/todos`. The request body contains:

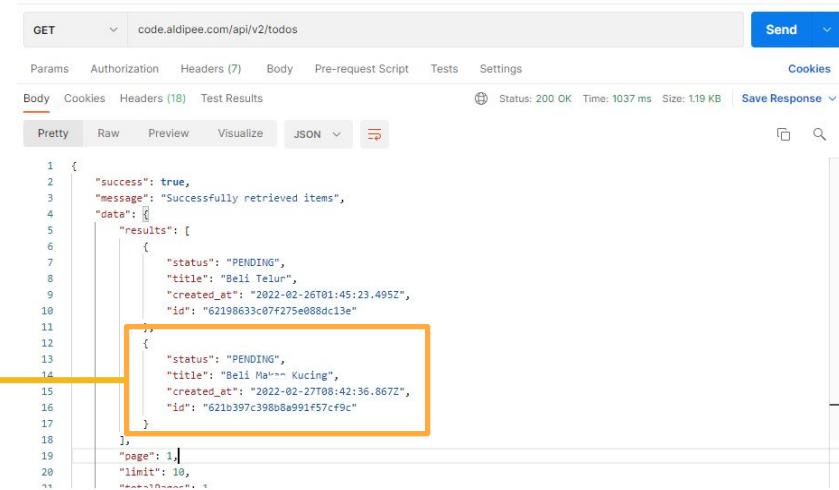
```
1 {  
2   "title": "Beli Makan Kucing",  
3   "description": "Beli whiskas untuk makan kucing",  
4   "status": "PENDING"  
5 }
```

The response status is **201 Created**, and the response body is:

```
1 {  
2   "success": true,  
3   "message": "Successfully created new item",  
4   "data": {  
5     "status": "PENDING",  
6     "title": "Beli Makan Kucing",  
7     "description": "Beli whiskas untuk makan kucing",  
8     "created_at": "2022-02-27T08:42:36.867Z",  
9     "updated_at": "2022-02-27T08:42:36.867Z",  
10    "id": "621b397c398b8a991f57cf9c"  
11  }  
12 }
```

- Untuk mengkonfirmasi data kamu udah tersimpan di server, kamu bisa request untuk membaca semua data to-do item pada server, kayak yang udah kita lakukan sebelumnya.

Nah sekarang data yang telah kamu tambahkan pada server pada method POST sebelumnya udah muncul pas dibaca.



A screenshot of the Postman application interface. The top bar shows a GET request to `code.aldipee.com/api/v2/todos`. The status bar indicates `Status: 200 OK`, `Time: 1037 ms`, and `Size: 1.19 KB`. The main area shows a JSON response with two items in the results array, each representing a to-do item with fields like status, title, created_at, and id. The entire JSON response is displayed in a code block below:

```
1 {
2     "success": true,
3     "message": "Successfully retrieved items",
4     "data": [
5         "results": [
6             {
7                 "status": "PENDING",
8                 "title": "Beli Telur",
9                 "created_at": "2022-02-26T01:45:23.495Z",
10                "id": "62198633c07f275e088dc13e"
11            },
12            {
13                "status": "PENDING",
14                "title": "Beli Makanan Kucing",
15                "created_at": "2022-02-27T08:42:36.867Z",
16                "id": "621b397c39808a991f57cf9c"
17            }
18        ],
19        "page": 1,
20        "limit": 10,
21        "total": 2
22    ]
}
```

Selesai sudah materi kita pada Spring Web part 1 ini. Sebelum lanjut ke kuis, ada latihan khusus buat kamu.

Buatlah REST dan RESTful API dengan @Controller annotation. Kemudian, lakukan testing pada API yang sudah kamu buat.

Jangan lupa latihan yang sudah kamu kerjakan, **upload ke GIT ya!** Di pertemuan selanjutnya, kita akan latihan kembali menggunakan pekerjaan yang sudah kamu kerjakan di topic 1.



Waahh.. rampung nih topik pertama. Kita jadi tahu kalau feature Postman cukup powerful dalam membantu melakukan API test~

Selain Postman, apa lagi ya API test tool yang cukup populer di kalangan Back End Java Engineer? Boleh banget lho kamu mau eksplor API test tool selain si Postman ini.



Nah, selesai sudah pembahasan kita di Chapter 5 Topic 1 ini.

Selanjutnya, kita bakal bahas tentang Spring Web Part 2.

Penasaran kayak gimana? Yuk, langsung ke topik selanjutnya~

