



NodeJS HTTP Server

Gold - Chapter 4 - Topic 4

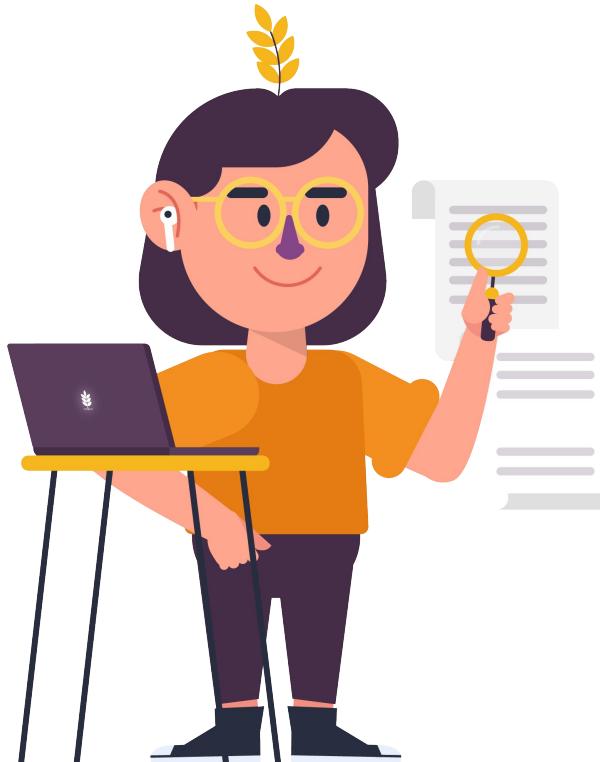
**Selamat datang di Chapter 4 Topic 4 online course
Full-stack Web dari Binar Academy!**





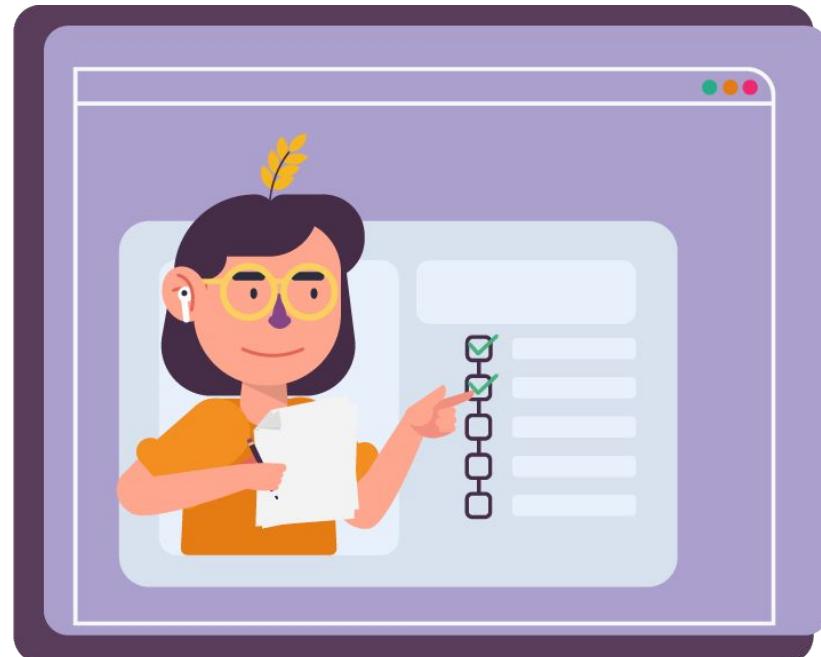
Di topik ini kita bakal belajar lebih detail lagi terkait **HTTP Server**. Nah HTTP Server ini penting buat backend maupun frontend, karena hal ini lah yang akan membuat website kita dapat diakses oleh user.

Tanpa basa basi lagi, kuy gaskeun ~



Detailnya, kita bakal bahas hal-hal berikut ini:

- Konsep dan fungsi HTTP server
- Melakukan serving file HTML dengan HTTP Server
- Membuat endpoint yang merespon dengan file JSON





Kamu tau gak sih, apa sebenarnya **HTTP Server** itu?

Contohnya, ketika kamu membuka `google.com`. Siapa yang melayani permintaan (request) yang kamu tulis dihalaman web?

Hmmm... siapa ya? Biar ga bingung, langsung kita kepoin yuk!





Apa sih HTTP Server itu?

adalah sebuah **aplikasi yang dapat kita tulis, yang bertujuan untuk melayani request dari internet dan memberi response yang sesuai dengan request tersebut.**

Sebagai contoh, Sabrina membuat request ke google.com untuk membuka halaman Home, maka dari itu server akan memberikan response halaman Home, baik itu berupa HTML, ataupun format lain.

Anggep aja, kayak kamu pesen nasi goreng, kamu request nasi goreng, kamu dapet nasi goreng.



Sama halnya kayak restoran, atau tempat makan yang lain, ketika kalian memesan sesuatu, pasti ada semacam template cara pesan atau SOP nya kan?

Ada yang ketika kamu pesen harus nungguin sampe jadi, ada yang kamu pesen bisa langsung kamu tinggal duduk, ada yang pesennya harus nunggu pesenan orang lain selesai, dan lain2.





Nah **prosedur ini lah yang kita sebut sebagai Protocol**. HTTP Server memiliki beberapa protocol dalam menerima request dan response, namun hal ini akan kita bahas pada chapter-chapter berikutnya.

Nah, sebelum kita belajar lebih dalam, kita pelajari dulu yuk tentang **request** dan **response**.



Request

Merupakan sebuah **permintaan yang dikirim client ke server**. Kalau analogi abang nasi goreng tadi, request nya itu adalah permintaan nasi goreng.

Setiap request itu memiliki 3 informasi yang wajib tercantum di dalamnya:

- **Request URL**
- **Request Method**
- **Request Header**
- **Request Body (Optional)**





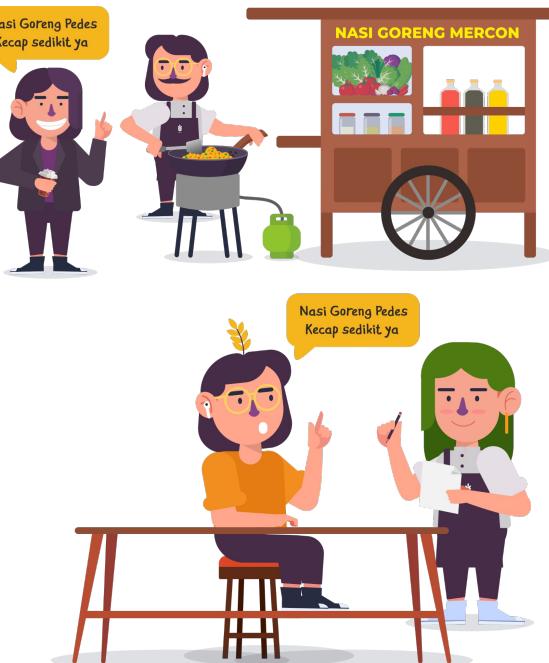
Request URL

Request tuh dianalogikan sebagai sebuah pesan. Request URL, menjelaskan siapa yang kamu kirimin pesan. Kalau case beli nasi goreng, berarti kamu pesennya itu ke siapa? Apa ke abang koki nya langsung, atau lewat mba yang bantu melayani?

Tiap orang memiliki request URL yang berbeda-beda, karena kita ngirimin pesan ke orang yang berbeda.

Dalam kasus website, request URL itu biasanya menandakan halaman mana yang ingin kita buka yang biasanya dengan nama domain yang sama, contoh:

- [Halaman Home Binar Academy](#)
- [Halaman Bootcamp Binar Academy](#)

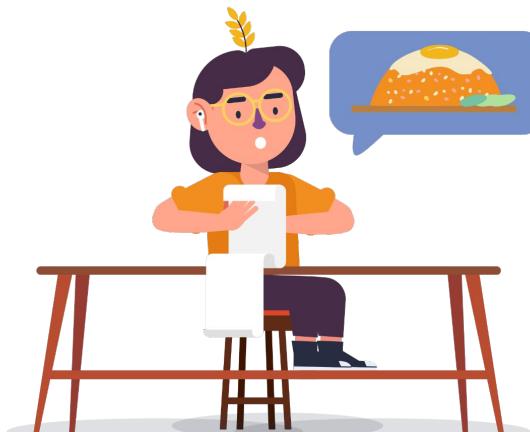




Request Method

Kalian pernah ga sih ngalamin **pesan nasi goreng, tapi cara pesennya beda**. Ada yang cukup ngomong ke abangnya, tapi ada yang harus nulis form pesanan, atau malah harus langsung pesen dan bayar di kasir.

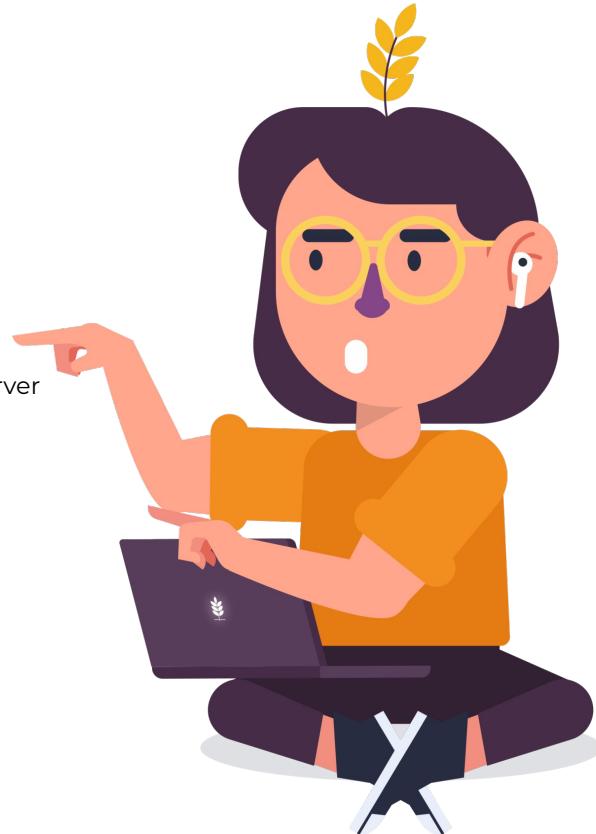
Walaupun pesennya item yang sama, yaitu nasi goreng. Tapi metode pesennya yang beda. Perbedaan cara request tersebut dibedakan menggunakan yang namanya method.





Nah, request method yang sering dipake itu ada 4 jenis, yaitu:

- **GET** : Request yang gak butuh nulis apa-apa, tinggal minta aja
- **PUT** : Request yang dipake untuk merubah sesuatu yang udah ada di server
- **POST** : Request yang dipake buat menambahkan sesuatu di server
- **DELETE**: Request yang dipake buat menghapus sesuatu di server.





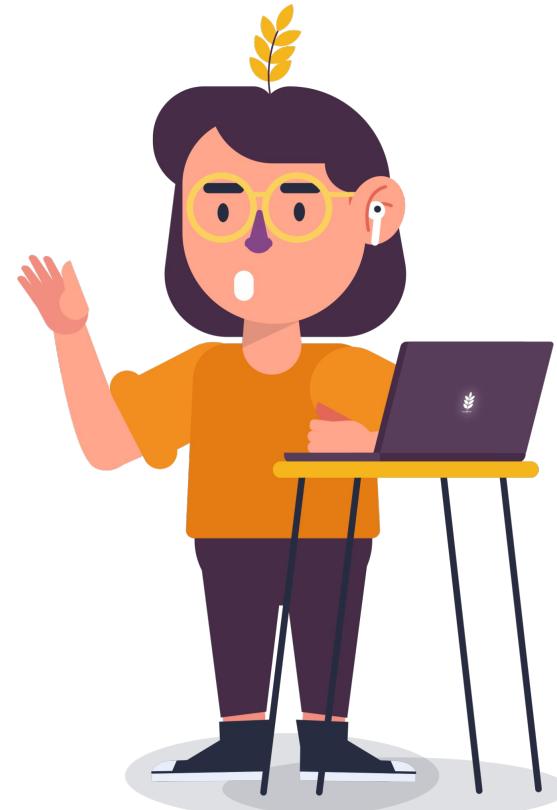
Request Header

Balik lagi ke analogi pesan nasi goreng, pasti ada konsumen yang pesennya kan?

Nah **informasi tentang pengirim request itu, biasa disebut sebagai header**. Kalau kamu masih inget surat-surat jadul, request header ini kaya kop surat. Biasanya berisi informasi terkait dari pesan tersebut, contoh:

- Pesan tersebut ditulis dengan format apa
- Pesan tersebut ditulis menggunakan bahasa apa
- Pesan tersebut dikirim melalui apa
- Dan sebagainya.

Nah request header ini akan berperan penting nantinya ketika kita akan mempelajari tentang protocol HTTP seperti RESTful API.





Request Body

Request body merupakan **isi pesan yang dibaca oleh penerima pesan**. Kalo dari analogi pesan nasi goreng tadi, maka isi pesannya adalah “pesanan nasi goreng” yang terima oleh abang-abangnya

Kalo kalian lagi main-main ke website, request body itu biasanya berisi apa yang kalian masukkan di dalam sebuah input di website, seperti **form** dan sebagainya.





Response

Merupakan **bentuk informasi yang dikirimkan dari server ke client atas request yang dikirimkan**. Kalau pesanan nasi goreng tadi, berarti response ini adalah nasi goreng yang dikasih abang nya ke konsumen.

Setiap response itu memiliki atribut yang sama dengan request, hanya ada 1 atribut tambahan, yaitu response status.

Response status ini berguna untuk memberi tanda berhasil atau tidaknya sebuah request. Kan bisa aja, kalian pesen nasi goreng telor, ternyata telor-nya udah abis.

Nah untuk daftar response status yang ada, bisa kalian lihat pada link berikut:

[HTTP Status Code](#)

HTTP Status Codes



**Contoh response status code**

Code	Message	Description
200	OK	Request berhasil dan tidak ada perubahan di dalam server.
201	Created	Penambahan data baru di dalam server berhasil.
401	Unauthorized	Request ditolak karena hak akses tidak valid
403	Forbidden	Request ditolak karena hak akses tidak mencukupi
500	Internal Server Error	Request gagal karena kesalahan server



Cukup basa-basinya, sekarang kita langsung latihan untuk membuat **HTTP Server** menggunakan Node.Js



HTTP Server Menggunakan Node.JS

Step 1: Buka IDE dan Buat Project Baru

Silahkan buat project baru di dalam IDE kalian, dan pastikan kalian membuka project yang benar yak.





Step 2 : Buka Terminal di dalam Project-mu

Setelah kalian membuka terminal, inisialisasi project-mu agar menjadi sebuah project node.js, dengan mengetik perintah di samping.



```
# Pengguna npm  
npm init -y
```

```
# Atau pengguna Yarn  
yarn init -y
```



Step 3 : Buat file bernama index.js

File tersebut akan berisi kode yang berguna untuk membuat aplikasimu menjadi sebuah HTTP Server. Kalian bisa lihat kode disamping.

[HTTP Server](#)

```
const http = require('http');
const { PORT = 8000 } = process.env;

function onRequest(req, res) {
  res.writeHead(200);
  res.end("Halo dari server!");
}

const server = http.createServer(onRequest);

server.listen(PORT, '0.0.0.0', () => {
  console.log("Server sudah berjalan, silahkan buka http://0.0.0.0:%d", PORT);
})
```



Step 4 : Jalankan file index.js

Setelah kalian menjalankan file index.js, kalian dapat mengakses server tersebut melalui <http://0.0.0.0:8000>. Ketika kalian membuka link tersebut, server akan memberi response :

Halo dari server!

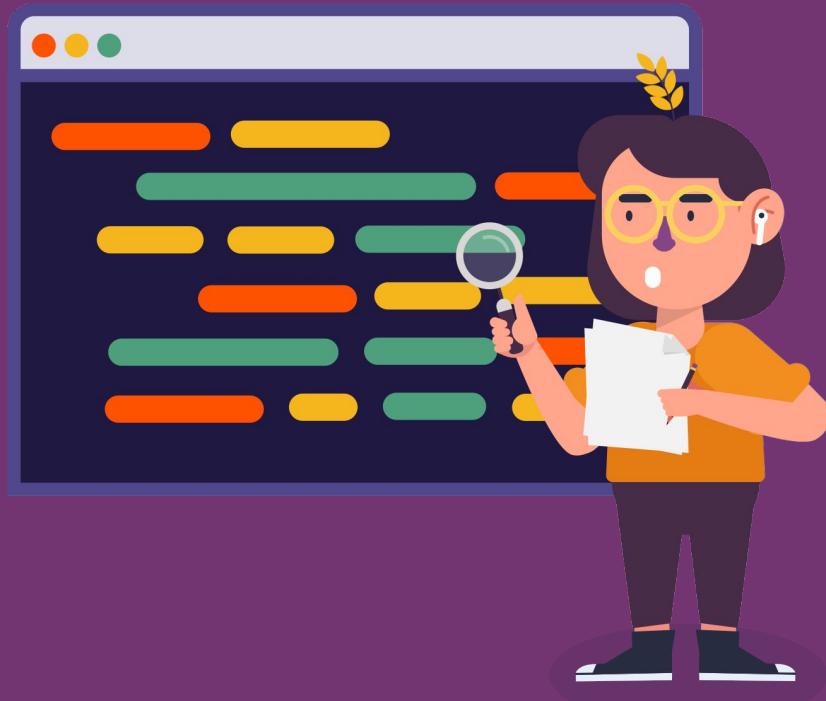
Seperti gambar disamping ~



Sekarang kalian sudah tau, gimana cara membuat server dan menghandle sebuah request.

Nah pada umumnya, sebuah website yang kita akses akan **merespon dengan file HTML**. Kira-kira gimana ya cara bikinnya?

Yuk kita cari tahu~





```
const http = require('http');
const { PORT = 8000 } = process.env;

const fs = require('fs');
const path = require('path');
const PUBLIC_DIRECTORY = path.join(__dirname, 'public');

function onRequest(req, res) {
  const htmlFile = path.join(PUBLIC_DIRECTORY, 'index.html');
  const html = fs.readFileSync(htmlFile, 'utf-8')
  res.setHeader('Content-Type', 'text/html')
  res.writeHead(200)
  res.end(html)
}

const server = http.createServer(onRequest);

server.listen(PORT, '0.0.0.0', () => {
  console.log("Server sudah berjalan, silahkan buka http://0.0.0.0:%d", PORT);
})
```

Serve HTML File

Untuk memberikan HTML sebagai response, kita **perlu menggunakan fs library**. Library tersebut akan kita gunakan untuk membaca file html di server kita, yang kemudian kita jadikan response.

Untuk source code-nya sama dengan yang HTTP server yang sebelumnya sudah dibahas. Hanya ada sedikit perbedaan pada bagian **onRequest** nya.

[HTTP Server dengan HTML](#)



Step 1: Buat file HTML pada direktori public

Karena kita ingin memberikan file HTML sebagai response, maka kita harus menyediakan terlebih dahulu file HTML nya.

Kalian bisa membuat file dengan menggunakan perintah berikut:

```
mkdir public
```

```
touch index.html
```

Silahkan buat file HTML sesukamu, dengan catatan, harus pake CSS internal, atau CSS yang di-host menggunakan CDN.

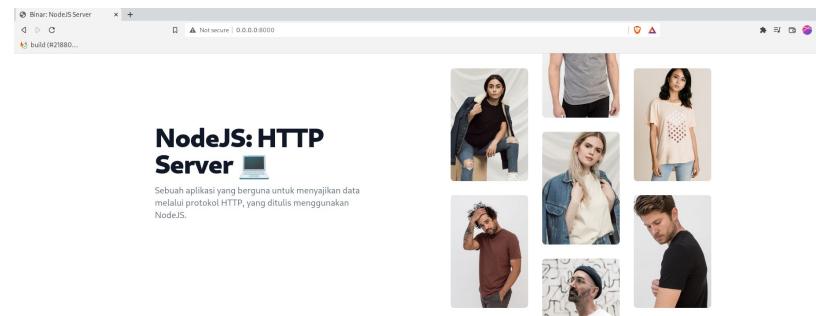


```
<h1>Hello World</h1>
```



Step 2 : Jalankan file index.js

Setelah kalian menjalankan file index.js, kalian dapat mengakses server tersebut melalui <http://0.0.0.0:8000>. Dan ketika kalian membuka link tersebut, server akan memberi response seperti gambar disamping.

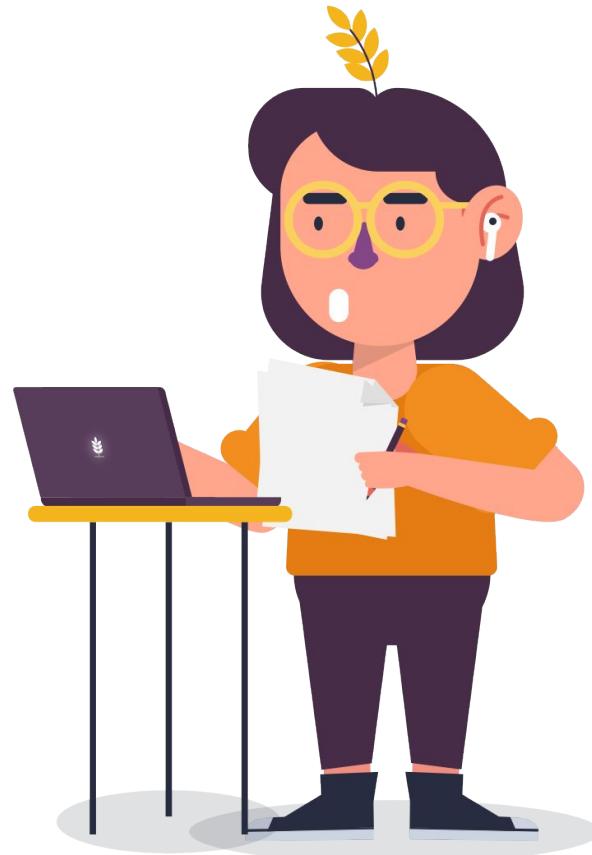




Serve HTML File

Mungkin dari latihan diatas kalian bertanya, “*Ini sih, cuma bisa dipake buat 1 halaman doang, website kan biasanya ada banyak halamannya, gimana tuh caranya?*”

Nah, solusi untuk itu adalah *routing*. Yuk kita simak gimana caranya~





```
● ● ●

function getHTML(htmlFileName) {
  const htmlFilePath = path.join(PUBLIC_DIRECTORY, htmlFileName);
  return fs.readFileSync(htmlFilePath, 'utf-8')
}

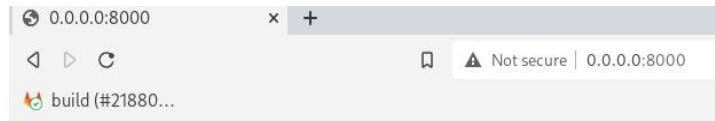
function onRequest(req, res) {
  switch(req.url) {
    case "/":
      res.writeHead(200)
      res.end(getHTML("index.html"))
      return;
    case "/about":
      res.writeHead(200)
      res.end(getHTML("about.html"))
      return;
    default:
      res.writeHead(404)
      res.end(getHTML("404.html"))
      return;
  }
}
```

Routing

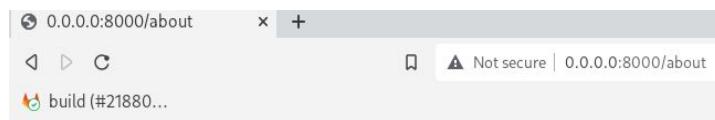
Untuk melakukan routing caranya cukup sederhana. Kalian masih inget kan pembahasan tentang conditional statement?

Kita bisa pake **if statement**, atau **switch statement**, untuk melakukan routing.

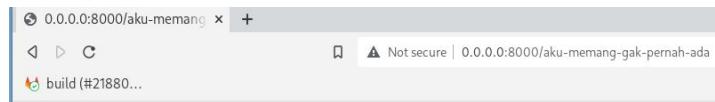
[HTTP Server dengan HTML dan Routing](#)



Aku adalah halaman Home



Aku adalah halaman About



Halaman tidak ditemukan

Jalankan server, dan coba buka halaman-halaman berikut:

- Home : <http://0.0.0.0:8000>
- About : <http://0.0.0.0:8000/about>
- 404 : <http://0.0.0.0:8000/aku-memang-qak-pernah-ada>

Nah, contoh-contoh tadi adalah contoh kode yang digunakan untuk arsitektur monolith (frontend dan backend jadi satu).

Pada kenyataannya, nanti kita akan menghadapi juga arsitektur website yang frontend dan backend nya terpisah.

Arsitektur ini biasanya menggunakan JSON untuk mentrasmisikan data antara backend dan frontend.

Nah sekarang kita coba cari tau yuk, cara membuat **http server** yang merespon dengan **JSON** ~





JSON

JSON atau Javascript Object Notation adalah **sebuah file atau encoding yang memiliki sintaks seperti Javascript object, namun tiap atribut ditandai dengan tanda kutip**. Contoh file JSON dapat dilihat pada contoh disamping.

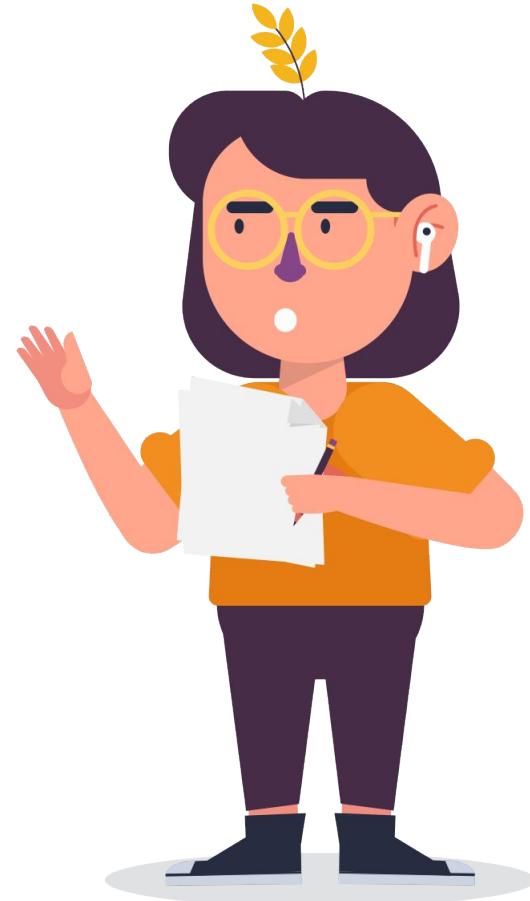
Nah file JSON ini biasa digunakan **untuk transmisi data antara frontend dan backend**. Kelebihan menggunakan JSON dibanding format lain adalah, JSON lebih mudah diurai dan dibentuk baik dari server, maupun dari client. Maka dari itu JSON sangat cocok sebagai format untuk transmisi data antara frontend dan backend.



```
{  
  "id": "0aa3e62a-9a27-45a2-9e7a-0d4f1d7f02dd",  
  "username": "sabrina",  
  "email": "sabrina@binar.co.id",  
  "encrypted_password": "ldfgkj78%&appdK0039*"  
}
```



Di dalam proses request-response, ketika kita mengirim request dan response dalam format JSON, ada header bernama **Content-Type** yang nilainya adalah **application/json**.





```
function toJSON(value) {
  return JSON.stringify(value);
}

function onRequest(req, res) {
  const responseJSON = toJSON({
    id: "0aa3e62a-9a27-45a2-9e7a-0d4f1d7f02dd",
    username: "sabrina",
    email: "sabrina@binar.co.id",
    encrypted_password: "ldfgkj78%^&appdK0039*"
  })

  res.setHeader("Content-Type", "application/json")
  res.writeHead(200)
  res.end(responseJSON);
}
```

Membuat Response Menggunakan JSON

Untuk merespon dengan JSON pada sebuah request, caranya mudah, kita hanya perlu melakukan **setHeader** untuk **Content-Type** dan beri nilai **application/json**.

Lalu untuk response body-nya kita hanya perlu memanggil **JSON.stringify** saja agar semua jenis data di Javascript dikonversi ke **string JSON**.

[HTTP Server dengan JSON](#)

Menerima JSON Request

Untuk menerima request body yang berupa JSON, kita perlu memastikan apakah **Content-Type** dari request tersebut berupa JSON, apabila iya, maka kita akan melakukan parsing dengan menggunakan **JSON.parse**.

Untuk detail kode dari hal ini dapat dilihat melalui link berikut:

[HTTP Server dengan JSON Request](#)



Saatnya kita Quiz!





1. Manakah pernyataan yang salah terkait HTTP Server!

- A. http server berguna untuk menampilkan halaman di dalam browser.
- B. http server berguna untuk menerima request dari client dan memberikan response sesuai dengan konteks.
- C. http server dapat menerima request dan memberikan response dengan format tertentu, seperti JSON.

1. Manakah pernyataan yang salah terkait HTTP Server!

- A. http server berguna untuk menampilkan halaman di dalam browser.
- B. http server berguna untuk menerima request dari client dan memberikan response sesuai dengan konteks.
- C. http server dapat menerima request dan memberikan response dengan format tertentu, seperti JSON.

http server tidak dapat dan tidak ada urusannya dengan menampilkan halaman di dalam browser, http server hanya memberikan template yang akan ditampilkan di dalam browser.



2. Manakah nilai yang menandakan sebuah request ditulis menggunakan format JSON?

- A. text/json
- B. application/json
- C. text/plain

- 2. Manakah nilai yang menandakan sebuah request ditulis menggunakan format JSON?**
- A. text/json
 - B. application/json**
 - C. text/plain

application/json adalah sebuah nilai dari Content-Type yang digunakan untuk menandakan bahwa sebuah request atau response ditulis dengan format JSON.



3. Atribut manakah yang akan kita gunakan dalam melakukan routing?

- A. req.path
- B. req.url
- C. req.query

3. Atribut manakah yang akan kita gunakan dalam melakukan routing?

- A. req.path
- B. **req.url**
- C. req.query

req.url memiliki nilai URL yang kita kirim kedalam sebuah server tanpa nama domain.



4. Manakah yang merupakan pernyataan benar / salah tentang statement dibawah ini, disertai alasannya

“Untuk mengirim request body kedalam sebuah request, kita perlu menggunakan metode POST, PUT, atau PATCH.”

- A. Benar, karena dalam standar industri, untuk mengirim payload atau request body, kita menggunakan POST, PUT, atau PATCH.
- B. Salah, karena kita bisa mengirim request body dalam semua request method.
- C. Benar, kita tidak dapat mengirim request body di dalam semua request method.

4. Manakah yang merupakan pernyataan benar / salah tentang statement dibawah ini, disertai alasannya

“Untuk mengirim request body kedalam sebuah request, kita perlu menggunakan metode POST, PUT, atau PATCH.”

- A. Benar, karena di dalam standar industri, untuk mengirim payload atau request body, kita menggunakan POST, PUT, atau PATCH.
- B. Salah, karena kita bisa mengirim request body dalam semua request method.
- C. Benar, kita tidak dapat mengirim request body di dalam semua request method.

A dan B sebenarnya sama-sama benar secara objektif, tapi di dalam standar industri, jawaban akan lebih condong ke A.



- 5. Apabila client yang belum login mengakses data yang hanya boleh diakses ketika login, kira-kira status code apa yang akan diterima client dari server?**
- A. 500
 - B. 403
 - C. 401

5. Apabila client yang belum login mengakses data yang hanya boleh diakses ketika login, kira-kira status code apa yang akan diterima client dari server?
- A. 500
 - B. 403
 - C. 401

401 atau Unauthorized, adalah sebuah status code yang diberikan ke client apabila client tidak memiliki hak akses (belum login)

Referensi bacaan ~

- [Slide Repository](#)
- [nodejs.dev](#)
- [Digital Ocean](#)



Gimana, udah tau kan cara bikin HTTP Server? Sekarang kamu bisa coba-coba bikin website kesayanganmu pake NodeJS, biar gaul!

Mungkin sampe sini dulu aja yak chapter kali ini hehehe, byeeee~



Terima Kasih!



Chapter ✓

completed