



Express.Js

Gold - Chapter 5 Topic 1

**Selamat datang di Chapter 5 Topik 1 online
course Fullstack Web dari Binar Academy!**



Halo, jumpa lagi di chapter baru 🙌!

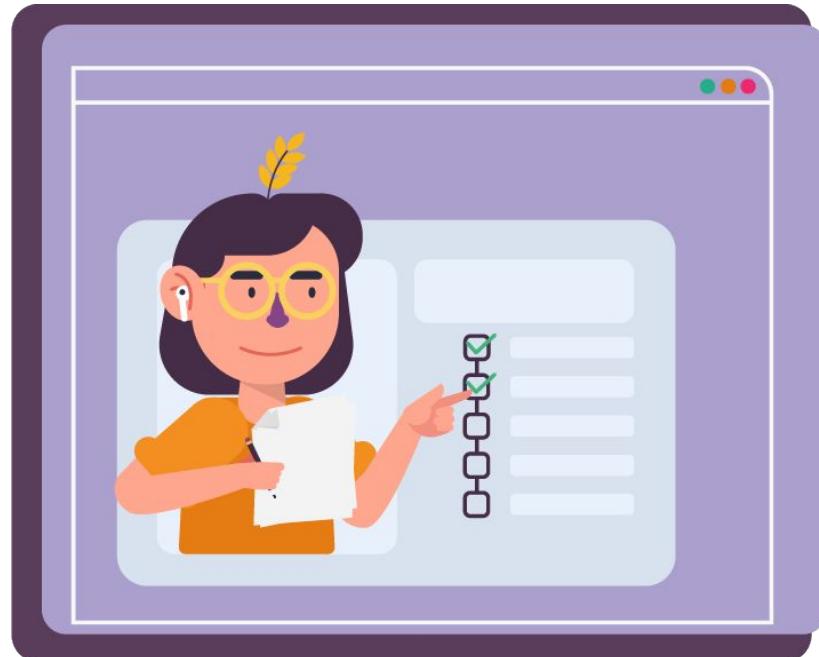
Di chapter sebelumnya kita udah belajar nih tentang OOP, DOM dan HTTP Server. Nah di chapter ini, kita bakal belajar tentang HTTP Server lebih mendalam. Yaitu tentang gimana cara membuat HTTP Server yang lebih mudah, dan cara menyimpan data. Pokoknya seru deh.

Nah untuk topic ini, kita bakal belajar gimana cara membuat HTTP Server yang lebih gampang dan menyenangkan, yaitu dengan menggunakan **Express.js**.



Detailnya, kita bakal bahas hal-hal berikut ini:

- Mengenal konsep Express.Js
- Memahami routing pada Express.Js
- Middleware pada Express.Js
- Penggunaan view engine untuk membuat HTML





Hmmm... Kenapa namanya Express ya?
Apakah dengan menggunakan
Express.Js, perfoma HTTP Server kita
akan jauh lebih cepat?

Untuk menjawab pertanyaan ini, kita
perlu tau dulu apa sih Express.Js itu?





Express.Js

Sebuah framework yang ditulis dan diperuntukan untuk aplikasi Node.Js, yang **berguna untuk membuat sebuah HTTP Server**. Framework ini cepat, unopinionated dan minimalis seperti yang diklaim oleh situs resminya Express.Js yaitu expressjs.com.

Express 4.17.3

Fast, unopinionated,
minimalist web framework for
Node.js

```
$ npm install express --save
```

Express 5.0 beta documentation is now available.

The beta API documentation is a work in progress. For information on what's in the release, see the Express [release history](#).

Web Applications

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

APIs

With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.

Performance

Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.

Frameworks

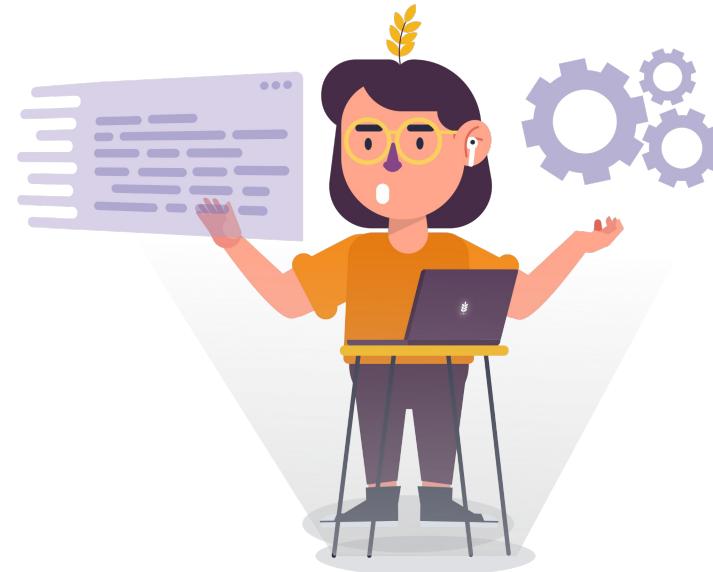
Many [popular frameworks](#) are based on Express.



H hmm... Cepat, unopinionated dan minimalis maksudnya apa ya?

Maksudnya, performa Express.Js relatif **cepat dibanding framework lain di dalam Node.Js**, karena Express.Js ini memiliki fitur yang minimalis dan sederhana, dimana hanya menyediakan cara routing sederhana.

Sedangkan, maksud unopinionated adalah ketika kita menggunakan Express.Js kita bebas mau membuatnya dengan cara apapun, **tidak ada kaidah baku dalam membuat sebuah aplikasi Express.Js**.





Oh iya, untuk membuktikan pernyataan-pernyataan diatas, kamu bisa explore lagi melalaui link berikut nih.

[Fastify.io / Benchmark](#)

Disitu kamu akan dijelasan perbandingan Express.Js dengan framework lainnya yang ditulis pakai Node.Js.





```
● ● ●

const http = require("http");
const PORT = process.env.PORT || 8000;
const server = http.createServer(function onRequest(req, res) {
  switch(req.url) {
    case "/":
      if (req.method === "GET") {
        res.writeHead(200);
        res.end('Hello, World!');
      }
      break;
    case "/api/v1/login":
      if (req.method === "POST") {
        res.writeHead(201);
        res.end('Masooook!');
      }
      break;
    default:
      res.writeHead(404)
      res.end('Mau kemana bos?')
  }
});

server.listen(PORT)
```

Selain cepat dari sisi performa, Express.Js sebagai framework juga bisa **mempercepat proses pembuatan HTTP Server** itu sendiri.

Coba liat contoh kode disamping yang ditulis tanpa Express.Js hehehe.

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



```
● ● ●  
  
const express = require("express");
const app = express();
const { PORT = 8000 } = process.env;  
  
app.get("/", (req, res) => {
    res.send("Hello, World!")
})  
  
app.post("/api/v1/login", (req, res) => {
    res.status(201).send("Masooook!")
})  
  
app.use((req, res) => {
    res.status(404).send("Mau kemana bos?")
})  
  
app.listen(PORT)
```

Nah... bedanya kalau pake Express.Js kaya gambar disamping, kode pada aplikasi kalian akan jauh lebih sederhana dan mudah dipahami.

Kita udah ga perlu switch statement atau if statement buat nge-handle HTTP request.

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



Fun fact Express.Js~

Bayangin deh ketika kita mau bikin puding dengan bentuk segitiga. Walaupun sebenarnya kita bisa buat puding kotak lalu dipotong secara manual, tapi akan lebih mudah dan cepat kalau kita bikin puding dengan cetakan segitiga. Dimana cetakan-cetakan ini adalah Express.Js.

Sama halnya dengan kamu bikin HTTP Server, bisa aja kamu pake modul http aja, tapi bakalan lebih **mudah** dan **cepat** kalo kamu pake Express.Js.





Sebenarnya, ada banyak sekali framework yang bisa kita pakai untuk membuat HTTP server. Diantaranya

- [Fastify.io](#) (lagi naik daun nih)
- [Strapi](#) (Solusi lebih ekspres dari express hehehe)
- [Hapi](#)

Tapi, di sesi ini kita hanya belajar **Express.Js** aja. Buat kalian yang penasaran & kepo, boleh eksplor lagi dari link diatas ya!



Sudah tau kan sekarang, apa itu Express.Js dan kenapa namanya Express~

Sekarang saatnya kita latihan cara membuat HTTP server pake Express.Js. Eits... tapi sebelumnya, kita wajib untuk instalasi Express.Js dulu ya!

Markicuus ~





Instalasi Express.Js

1. Inisialisasi Project Node.Js

Express.Js merupakan sebuah NPM package, tentu saja kita perlu inisialisasi Project **Node.Js** terlebih dahulu.



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

```
# Pengguna Yarn  
yarn init -y
```



2. Instalasi Express.Js

Seperti yang sudah disebut di langkah pertama tadi, **Express.Js** ini merupakan sebuah NPM package, maka kita harus kita instalasi dulu ya! :)



```
# Pengguna NPM  
npm install --save express
```

```
# Pengguna Yarn  
yarn add express
```

3. Buat main file yang berisi implementasi Express.Js

Setelah kita instal Express.Js, sekarang saatnya kita mengimplementasikan Express.Js untuk membuat sebuah **HTTP Server**.

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

```
const express = require("express");
const app = express();

// Ambil port dari environment variable
// Dengan nilai default 8000
const PORT = process.env.PORT || 8000;

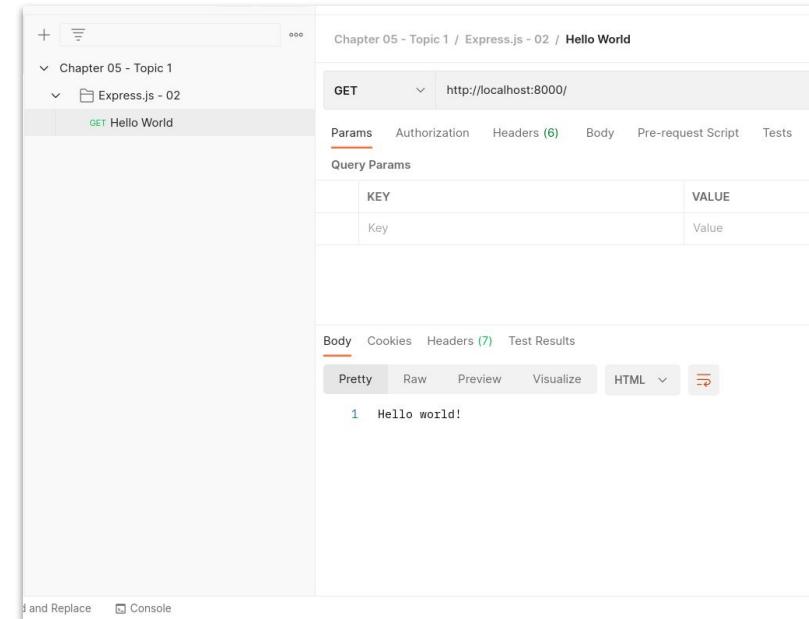
app.get("/", (req, res) => {
    res.send("Hello world!");
});

app.listen(PORT, () => {
    console.log(`Express nyala di http://localhost:${PORT}`);
});
```

4. Request dengan menggunakan Postman

Aplikasi yang kita buat cuma punya 1 endpoint, yaitu endpoint **Root**, yang nantinya akan merespon dengan **Hello world!**

Kalo belum download **Postman**, kalian bisa download lewat [link ini](#).



The screenshot shows the Postman application interface. On the left, there's a sidebar with a tree view: Chapter 05 - Topic 1, then Express.js - 02, and finally a green 'GET Hello World' node. The main area has tabs for GET, POST, PUT, etc., with 'GET' selected. The URL is set to 'http://localhost:8000/'. Below the URL, there are tabs for Params, Authorization, Headers (6), Body, Pre-request Script, and Tests. Under Headers, there are two entries: 'Content-Type: application/json' and 'Accept: */*'. The Body tab is selected, showing a JSON object with a single key 'message' and a value 'Hello world!'. At the bottom, there are tabs for Body, Cookies, Headers (7), and Test Results, with 'Body' selected. The response body shows the text '1 Hello world!'. At the very bottom, there are buttons for 'Find and Replace' and 'Console'.

Simpel kan bikin HTTP Server pake Express.Js? 😎

Nah, untuk melakukan routing di Express.Js itu juga ga kalah simpelnya, kita cukup panggil method yang ada di dalam object app.

Yuk kita pelajarin bareng-bareng!

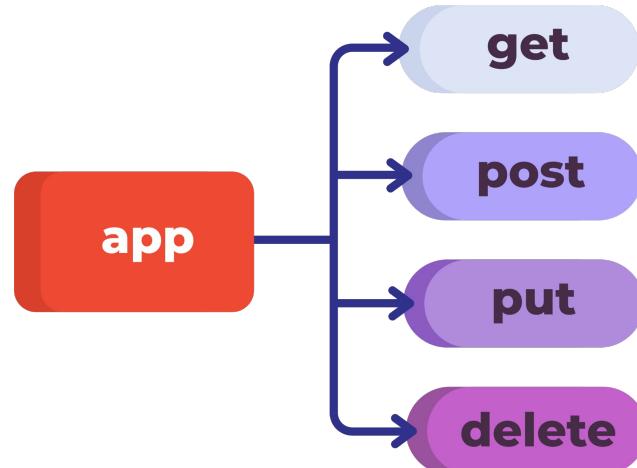


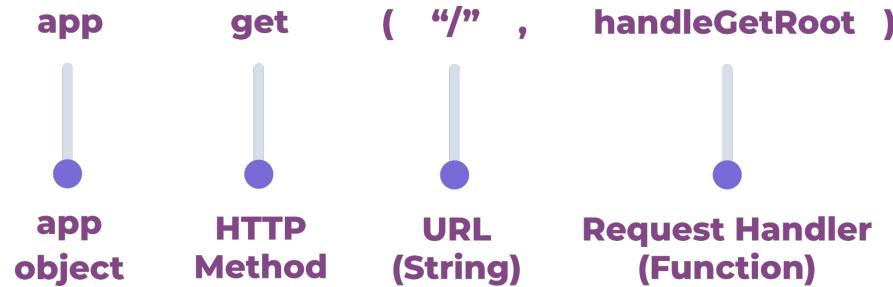


Routing

Untuk melakukan routing di Express.js, kita **cukup memanggil method dari app yang sudah kita inisialisasi di Express.js.**

Method-method tersebut memiliki nama yang sama dengan HTTP Request Method. Ketika kita memanggil **app.get**, artinya kita sedang membuat sebuah endpoint dengan method get, dengan URL sebagai parameter utama, dan request handler di parameter kedua.





Seperti gambar diatas, ketika app diikuti dengan HTTP method, itu berarti kita sedang mendefinisikan sebuah endpoint di dalam HTTP server kita.



Ketika ada request yang masuk ke endpoint yang kita buat, maka Express.js akan mendelegasikan request tersebut ke fungsi **handleGetRoot**, yang mana dapat kita buat seperti gambar ini

```
function handleGetRoot(req, res)
{
    console.log("Everything");
    res.send("Hello world!");
}
```



Terus untuk membuat banyak endpoint gimana caranya ya?

Kalau kalian sadar, di beberapa contoh sebelumnya, kita sudah bikin banyak endpoint di HTTP Server kita.

Pada dasarnya, untuk membuat endpoint di Express.js itu, kita hanya perlu spam **app.[method]** berkali-kali hehe.



```
app.get("/", handleGetRoot);
app.post("/api/v1/login", handleLogin);
app.post("/api/v1/books", handleCreateBooks);
app.get("/api/v1/books", handleListBooks);
app.get("/api/v1/books/:id", handleGetBook);
app.put("/api/v1/books/:id", handleUpdateBook);
app.delete("/api/v1/books/:id", handleDeleteBook);
```

Gimana, sekarang udah bisa memakai routing di Express.Js kan? Pastinya udah dong!

Nah, Express.Js juga bisa membantu kita dalam mengolah data dari **request** dan **response**. Yuk kita kepoin!





Request Object

Nah, request yang masuk ke dalam sebuah HTTP Server yang ditulis menggunakan Express.Js akan **memiliki beberapa properti tambahan yang dapat membantu untuk mengolah request yang masuk**. Yaitu :

- req.params
- req.query
- req.body





1. Path Parameter

Di dalam HTTP Request, terdapat sebuah parameter yang dikirim melalui URL secara spesifik adalah path dari URL-nya atau biasa kita sebut sebagai endpoint.

Nah, untuk menghandle ini, kita hanya perlu menggunakan sebuah notasi **titik dua** diikuti dengan nama parameter, sebagai contoh **:id**, yang mana id ini akan dapat kita akses menggunakan **req.params.id** di dalam request handler.

GET /api/v1/books/1

Dari contoh **diatas angka 1 inilah yang kita sebut sebagai path parameter.**

```
app.get("/api/v1/books/:id", (req, res) => {
  console.log("Book.id", req.params.id);
  res.send(
    `Kamu sedang mencari buku dengan ID: ${req.params.id}`
  );
})
```



2. Query Parameter

Query parameter adalah sebuah parameter yang dikirim melalui URL, dimana parameter tersebut adalah sebuah pasangan key dan value, yang mana query parameter ini ditulis setelah tanda tanya di dalam sebuah URL.

Sebagai contoh: <https://www.google.com/search?q=Kucing>

Dimana **q=Kucing** adalah sebuah query parameter.

Untuk mengakses query parameter di dalam Express.Js, kita hanya perlu memanggil **req.query** yang mana berbentuk sebuah object yang atribut-atributnya akan sesuai dengan query parameter yang dikirim.

```
// GET /api/v1/books?author=Fikri
app.get("/api/v1/books", (req, res) => {
  console.log(req.query.author);
  res
    .status(200)
    .send(`Kamu sedang mencari buku
yang ditulis oleh ${req.query.author}
`));
});
```



3. Request Body

Request body adalah sebuah data yang kirim melalui request body di dalam sebuah HTTP request yang ditulis dengan format tertentu.

Request body ini di dalam standar industri hanya akan digunakan pada request dengan method **POST**, **PUT**, dan **PATCH**.

Nah, karena request body ini ditulis dalam format tertentu, kita bisa menentukan format apa yang akan kita terima di dalam HTTP Server kita.

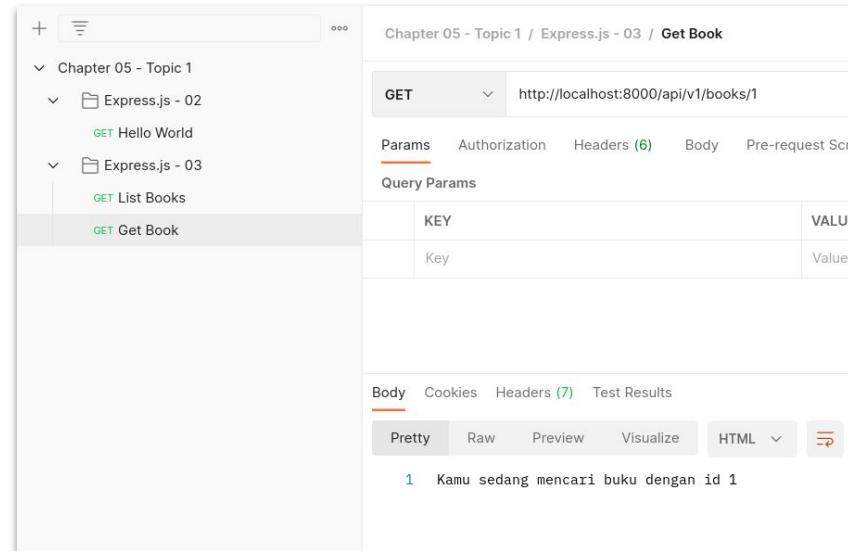
Sebagai contoh, kita akan menggunakan **application/x-www-urlencoded**.



```
app.use(express.urlencoded( ))  
  
const BOOK_MSG = `  
Terima kasih sudah  
menambahkan buku  
di dalam database kami.  
`  
  
app.post("/api/v1/books", (req, res) => {  
  console.log(req.body)  
  res.status(201).send(BOOK_MSG)  
});
```

Get Book

Lakukan request ke endpoint Get Book, dengan path parameter.

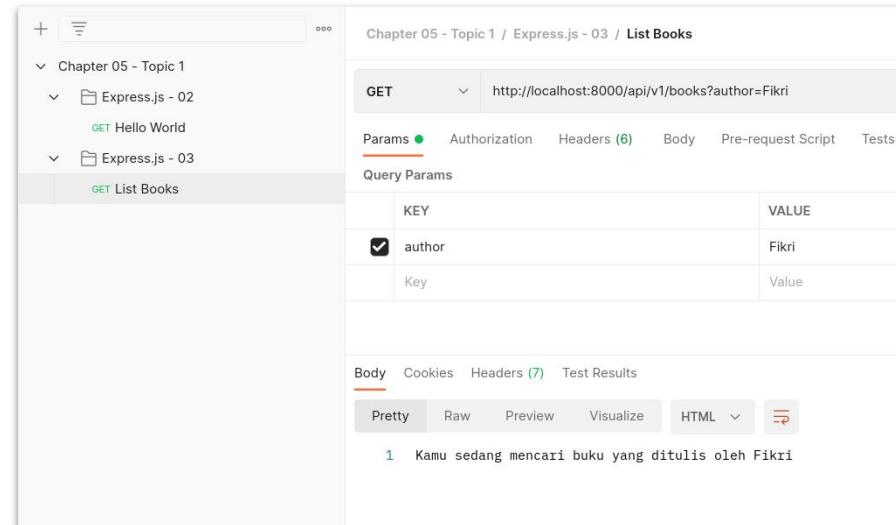


The screenshot shows the Postman application interface. On the left, there's a tree view of API endpoints under 'Chapter 05 - Topic 1'. Under 'Express.js - 02', there's a 'GET Hello World' endpoint. Under 'Express.js - 03', there are 'GET List Books' and 'GET Get Book' endpoints, with 'GET Get Book' currently selected. To the right, the main panel shows a 'GET' request to 'http://localhost:8000/api/v1/books/1'. The 'Params' tab is active, showing a single parameter 'Key' with an empty value. Below the request, the 'Body' tab is active, displaying the response: '1 Kamu sedang mencari buku dengan id 1'. Other tabs like 'Cookies', 'Headers', and 'Test Results' are also visible.

List Books

Lakukan request ke endpoint List Books, dengan query parameter author dengan nilai nama penulis.

Yuk, kita coba test di Postman. Source code dapat kamu download [disini](#).



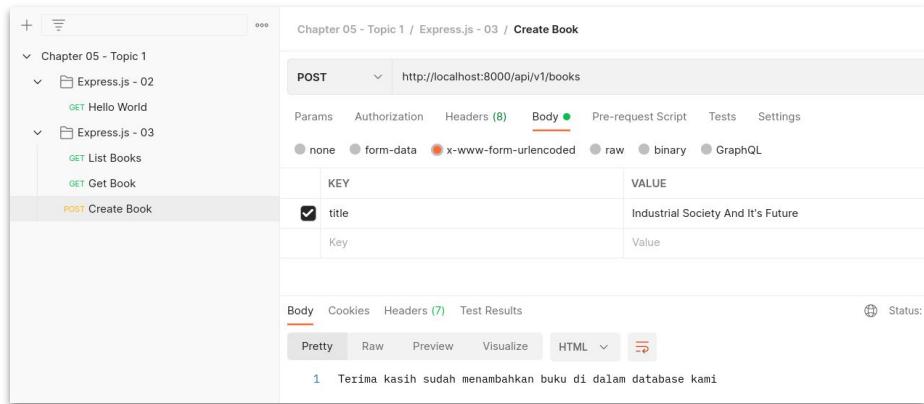
The screenshot shows the Postman application interface. On the left, there's a tree view of projects and collections: Chapter 05 - Topic 1, Express.js - 02 (with a GET Hello World endpoint), and Express.js - 03 (with a GET List Books endpoint, which is selected). The main right panel shows a GET request to `http://localhost:8000/api/v1/books?author=Fikri`. Under the "Params" tab, there is a single entry: `author` with value `Fikri`. Below the request, the "Body" tab is selected, showing the response body: `1 Kamu sedang mencari buku yang ditulis oleh Fikri`.



Create Book

Lakukan request ke endpoint Create Book, dengan request body yang isinya bebas dan cek terminal, nanti akan ada output object.

```
{ title: "Industrial Society And It's Future" }
```



The screenshot shows the Postman application interface. On the left, there is a sidebar with project navigation. The main area displays a POST request to the URL `http://localhost:8000/api/v1/books`. The 'Body' tab is selected, showing the following JSON payload:

```
{ "title": "Industrial Society And It's Future" }
```

Below the request, the response is shown in the 'HTML' tab, displaying the message: `1 Terima kasih sudah menambahkan buku di dalam database kami`.

Nah, selain mempermudah routing dan memproses data, dengan menggunakan Express.Js, kita bisa mengurangi LOC (Line of Code) dengan menggunakan **middleware** untuk algoritma yang sama, daripada kita nulis berkali-kali, ya gak?

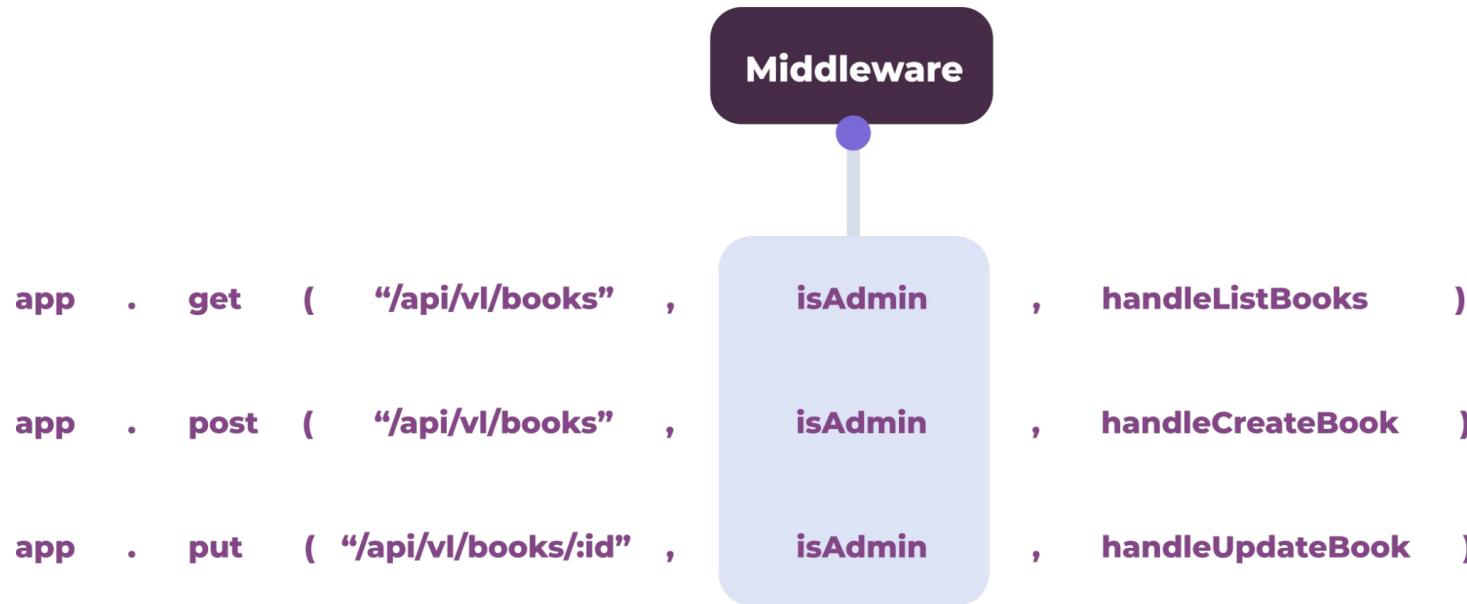


Middleware

Coba deh bayangin pas kalian naik kereta 🚂

Ketika kamu naik kereta, layanan yang kamu minta dari PT. KAI kan perjalanan menggunakan kereta, tapi untuk naik kereta kamu perlu punya tiket, dan dari PT. KAI akan mengecek tiket kamu ketika kamu mau naik kereta. Pengecekan inilah yang bisa kita sebut **middleware**.





Seperti yang disebutkan tadi, middleware dapat membantu kita dalam memproses request dengan urusan yang sama tanpa perlu menulisnya berulang-ulang. Contoh diatas menunjukkan bahwa pada endpoint **Create Book, List Books, dan Update Book** terdapat middleware yang digunakan untuk mengecek apakah client yang melakukan request ke endpoint tersebut adalah admin.



Nah, kalau kalian lihat pada gambar di slide sebelumnya, kan ada beberapa fungsi yang kita taruh pada sebuah endpoint, yaitu **isAdmin** dan fungsi selanjutnya.

Sebuah request tidak akan dilanjutkan ke fungsi selanjutnya apabila fungsi sebelumnya tidak memanggil fungsi **next** di dalamnya.

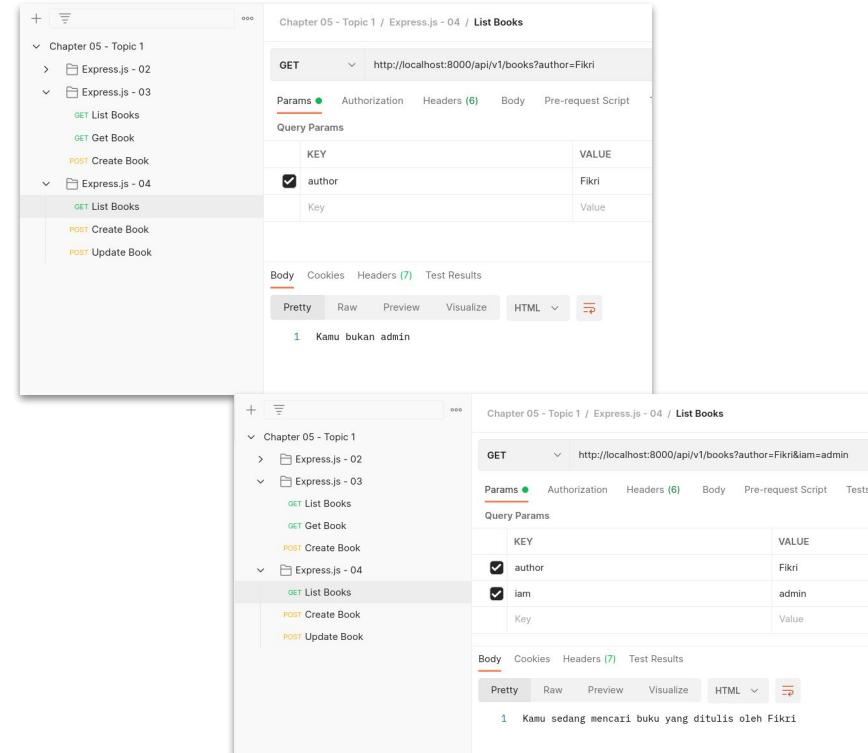
Maka dari itu, middleware biasanya digunakan untuk memvalidasi sebuah request karena kita tidak ingin melakukan sebuah logika bisnis apabila request tersebut tidak valid.

```
function isAdmin(req, res, next) {  
  if (req.query.iam === "admin") {  
    next();  
    return  
  }  
  
  res.status(401).send("Kamu bukan admin");  
}
```



Yuk, kita coba di Postman. Source code ini dapat kamu download [disini](#).

Kalau kalian perhatikan dari gambar disamping, ketika kita tidak mengirimkan query parameter **iam** yang bernilai **admin**, maka kita mendapat response **Kamu bukan admin**.



The image displays two side-by-side screenshots of the Postman application interface, illustrating the use of middleware in an Express.js application.

Screenshot 1 (Left): Shows a request to `http://localhost:8000/api/v1/books?author=Fikri`. The "Query Params" section shows a single entry: `author` with value `Fikri`. The response body is: `1 Kamu bukan admin`.

KEY	VALUE
author	Fikri

Screenshot 2 (Right): Shows a request to `http://localhost:8000/api/v1/books?author=Fikri&iam=admin`. The "Query Params" section shows two entries: `author` with value `Fikri` and `iam` with value `admin`. The response body is: `1 Kamu sedang mencari buku yang ditulis oleh Fikri`.

KEY	VALUE
author	Fikri
iam	admin

„ Terus gimana caranya pake express buat nyajuin HTML ke user kita, kan yang bakalan dipake user kita kan UI yang cantek bukan Postman atau terminal?

Nah, untuk menyajikan HTML ke user kita biar bisa aplikasi kita bisa dipake di browser, kita bisa pake yang namanya **View Engine**.

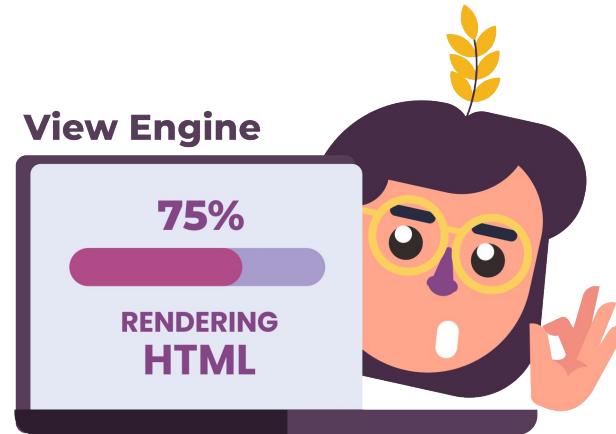


View Engine

View engine atau template engine adalah **sebuah tools yang digunakan untuk merender sebuah HTML berdasarkan data yang diberikan.**

Dengan view engine, kita bisa membuat HTML menjadi dinamis ketika disajikan ke user, namun kita tidak memiliki akses ke DOM, artinya HTML yang kita terima tidak dapat dimanipulasi secara realtime melalui View Engine, melainkan harus menggunakan Javascript yang ada di browser.

View engine ini dipakai Express.js untuk membuat response yang berupa HTML.



Macam-macam View Engine

Sama halnya dengan Express.Js, view engine ini ada buanyak yang bisa dijadikan opsi. Yang membedakan satu dengan yang lain adalah cara mereka mendefinisikan HTML. Nah di bootcamp ini kita hanya akan belajar **EJS** saja. Kenapa? Karena good default buat Express.Js.

Balik lagi, buat kalian yang suka kepo dan eksplor, nih coba liat daftar ini:

- [Pug](#) (Kalo kalian benci XML dan HTML)
- [Handlebars](#) (Kalo kalian suka kumis)
- [Dan masih banyak lagi.](#)





```
app.set('view engine', 'ejs')
```

Setup EJS

Untuk mensetup EJS di dalam Express.Js, caranya mudah, kita hanya perlu menginstal **ejs** package menggunakan **npm** maupun **yarn**.

```
# Pengguna npm  
npm install --save ejs  
  
# Pengguna yarn  
yarn add ejs
```



```
<h1>Hi, <%= name %></h1>
```

Buat Template

Setelah melakukan setup di **index.js**, buatlah directory bernama **views** di dalam **project root**.

Dan buatlah file bernama **index.ejs** di dalam folder tersebut. File inilah yang akan menjadi template dari HTML kita, yang akan kita render secara dinamis berdasarkan request.

Nah dari contoh disamping, yang akan kita render secara dinamis adalah **<%= name %>**. Tag tersebut akan di-replace oleh express ketika membuat response.



Rendering HTML

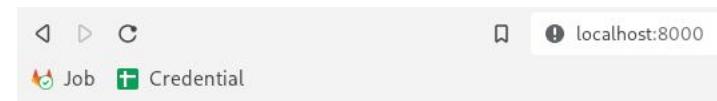
Untuk merender template yang sudah kita buat, kita bisa menggunakan **res.render**, yang mana ia berupa sebuah fungsi yang meminta dua parameter, yaitu **nama template**, dan **data** yang akan ditampilkan di dalam template.

Nama Template ini akan sama dengan nama file tanpa ekstensi di dalam folder **views**, jika di dalam folder views terdapat sub folder, maka nama view akan mengikuti relative path ke file tersebut tanpa ekstensi, sebagai contoh, **page/index**

Dan, untuk **data** sendiri, berupa sebuah object, yang mana tiap atributnya akan menjadi sebuah variabel di dalam **template**.



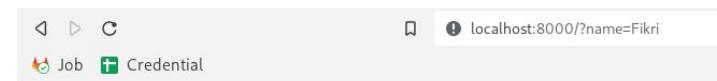
```
app.get("/", (req, res) => {
  res.render('index', {
    name: req.query.name || 'Guest'
  })
})
```



Hi, Guest

Yuk kita cobain aplikasi kita tadi yang pake view engine.

Source code-nya bisa kalian download [disini](#).

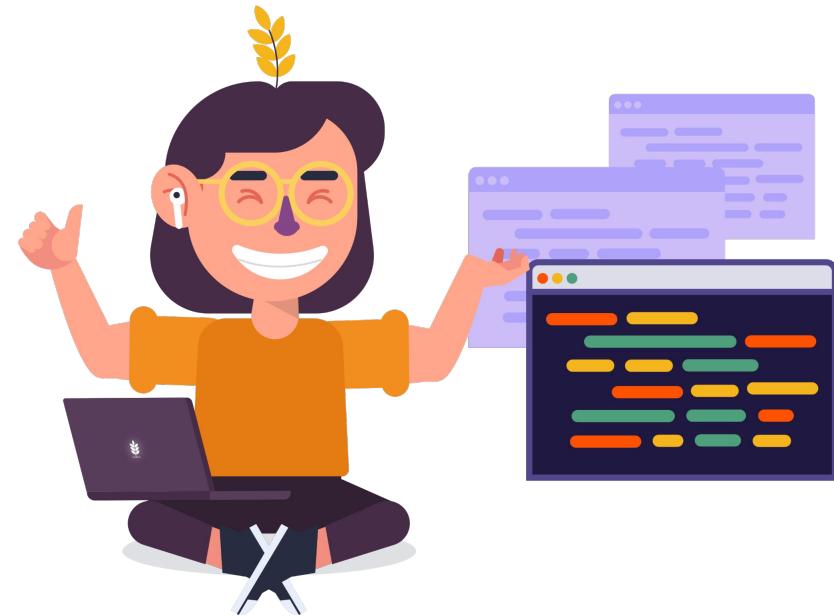


Hi, Fikri

Apa aja yang bisa kita lakukan dengan view engine?

- Membuat halaman-halaman di website kita
- Merender HTML secara dinamis berdasarkan kondisi
- Merender array of object ke dalam HTML jadi kita ga perlu tulis satu per satu
- Membuat form di dalam HTML dan sebagainya

Contoh advance dari penggunaan view engine ini dapat kalian lihat dari [repository ini](#).



- [Express.Js](#)
- [Express.Js - Digital Ocean](#)
- [Express.Js - Web Dev Simplified](#)
- [Express.Js - Net Ninjas](#)



Gimana, seru ga belajar express?

Udah mulai ngerasa ngoding ga seberat
yang dipikirkan kan? hehehe



Terima Kasih!



Next Topic

loading...