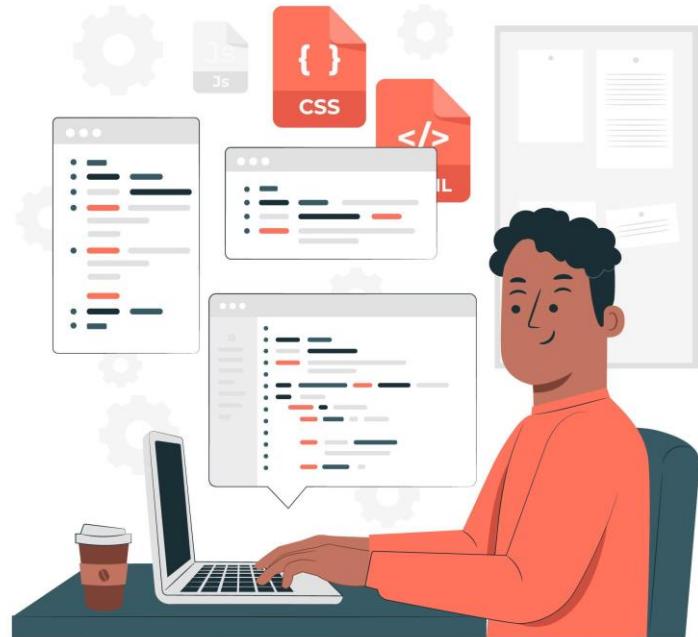


---

# Pengenalan Express.js, Web Service API & Middleware





KEMNAKER



# Outline

---

- Apa itu Express.js
- Instalasi Express.js
- Basic Routing Express.js
- Static File Express.js
- Web Service API
- Middleware di Express.js

---

# Apa itu Express.js

# Apa itu Express.js

---

- Express.js adalah framework web app untuk Node.js yang ditulis dengan bahasa pemrograman JavaScript. Framework open source ini dibuat oleh TJ Holowaychuk pada tahun 2010 lalu.
- Express.js adalah **framework back end**. Artinya, ia bertanggung jawab untuk mengatur fungsionalitas website, seperti pengelolaan routing dan session, permintaan HTTP, penanganan error, serta pertukaran data di server.

# Apa itu Express.js

---

Analogi:

Kalau Node.js itu seperti mesin mobil, Express.js adalah rangka dan dashboard yang membuat kita lebih mudah mengemudikan mobil itu.

Tanpa Express bisa jalan, tapi lebih ribet mengatur pintu, kemudi, dan lampu. Dengan Express, semuanya sudah siap di tempatnya.

# Tanpa Express vs Pakai Express

## Kekurangan tanpa Express:

- Routing manual → harus pakai if/else untuk setiap URL.
- Menangani *headers* dan *status code* secara manual.
- Tidak ada dukungan *middleware* bawaan.
- Parsing request body (JSON/form) ribet, harus bikin sendiri.

```
//tanpa express
// server-http.js
const http = require('http');
const port = 3000;

const server = http.createServer((req, res) => {
  if (req.url === '/' && req.method === 'GET') {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello World dari Node.js murni!');
  } else if (req.url === '/about' && req.method === 'GET') {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Ini halaman About');
  } else {
    res.writeHead(404, { 'Content-Type': 'text/plain' });
    res.end('Halaman tidak ditemukan');
  }
});

server.listen(port, () => {
  console.log(`Server berjalan di http://localhost:\${port}`);
});
```

# Tanpa Express vs Pakai Express

## Kelebihan dengan Express:

- Routing singkat (app.get, app.post, dst.).
- Middleware bawaan untuk parsing JSON & form.
- Lebih mudah menambah fitur autentikasi, logging, dll.
- Penanganan 404 & error lebih rapi.
- Komunitas besar dan banyak library siap pakai.

```
//pakai express
// server-express.js

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

// Middleware untuk parsing JSON
app.use(express.json());

// Route
app.get('/', (req, res) => {
  res.send('Hello World dari Express!');
});

app.get('/about', (req, res) => {
  res.send('Ini halaman About');
});

// 404 otomatis jika tidak ada route yang cocok
app.use((req, res) => {
  res.status(404).send('Halaman tidak ditemukan');
});

// Jalankan server
app.listen(port, () => {
  console.log(`Server berjalan di http://localhost:${port}`);
});
```

# Tanpa Express vs Pakai Express

---

## Kesimpulan:

- Kalau **Node.js** murni itu fleksibel tapi verbose (butuh banyak kode dasar).
- **Express.js** itu seperti “kit” siap pakai yang mempercepat pembuatan API dan website.

# Kelebihan dan Kekurangan Express.js

---

<b>Kelebihan</b>	<b>Kekurangan</b>
Bebas menentukan sendiri arsitektur dan struktur website yang akan dikembangkan	Diperlukan penelusuran kode lebih untuk membuat, mengelola, dan merawat arsitektur website yang sudah dibuat sebelumnya dengan express
Ukuran framework lebih kecil dan ringan, karena hanya berisi package inti	Terlalu banyak pilihan plugin dan library untuk digunakan, bisa membingungkan bagi sebagian orang
Kinerja website yang dihasilkan jadi lebih baik, mengingat ringannya framework	Pilihan metode pengembangan yang berbeda-beda, sehingga tidak ada standar baku

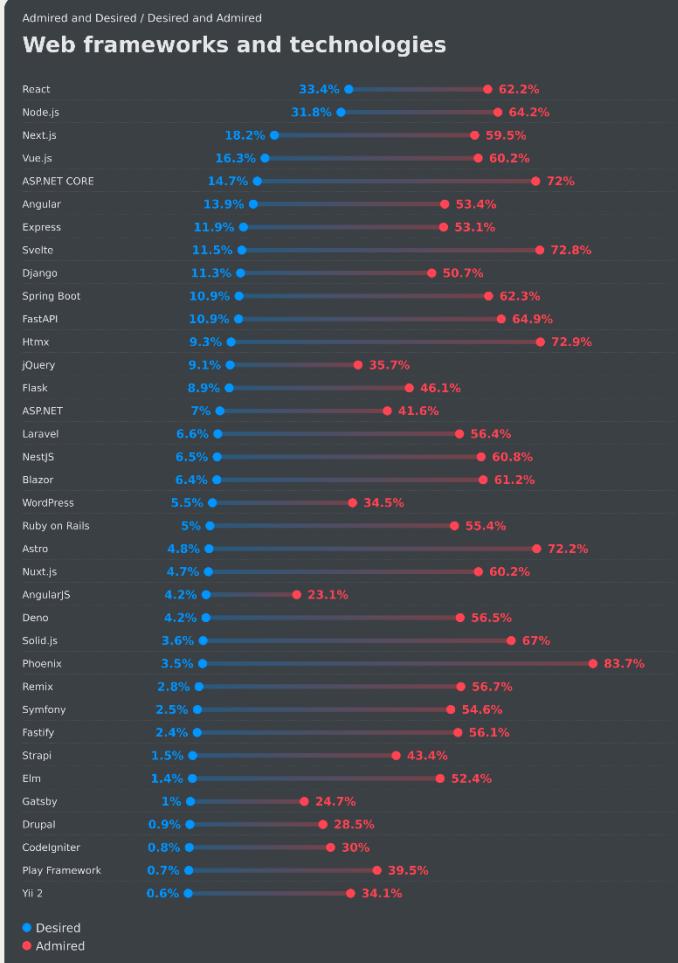
# Hasil Survey Stackoverflow 2024

- **React (33.4% desired / 62.2% admired):**  
Framework frontend paling populer, banyak diminati dan memuaskan developer.
- **Node.js (31.8% desired / 64.2% admired):**  
Platform backend JavaScript paling dominan, dipercaya untuk aplikasi skala besar.
- **Express (11.9% desired / 53.1% admired):**  
Framework backend ringan, masih relevan meski muncul banyak pesaing modern.

\*Desired (Diinginkan)

\*Admired (Dikagumi)

<https://survey.stackoverflow.co/2024/technology#2-web-frameworks-and-technologies>



# Contoh Alur

JSON berperan sebagai **format data di jalur komunikasi** antara *client* dan *server*, bukan format penyimpanan di database.



# Komponen Js dan Analogi singkatnya

---

- **Client**

Bisa browser, aplikasi mobile, atau Postman yang mengirim request.

- **Middleware**

express.json() membaca body request (jika ada data JSON) dan mengubahnya jadi objek JavaScript.

- **Routing**

Express memeriksa metode HTTP + path untuk memutuskan handler mana yang dijalankan.

- **Controller**

Fungsi yang mengolah data (di sini kita simpan di array users).

- **Response**

Server mengirim balik data (JSON) atau pesan status ke client.

Bayangkan Express.js itu seperti **restoran**:

- **Middleware** = pelayan yang mengecek pesanan, memvalidasi, menyiapkan meja.
- **Routing** = menentukan dapur mana yang akan masak pesanan.
- **Controller** = koki yang memasak sesuai pesanan.
- **Response** = makanan yang diantar ke meja.

---

# Installasi Express.js

# Step Installasi

---

- <https://expressjs.com/en/starter/installing.html>
- <https://expressjs.com/en/starter/hello-world.html>

# Step (1)

---

buatlah folder baru lalu masuk ke directory folder tersebut

```
$ mkdir myapp  
$ cd myapp
```

setelah itu jalankan perintah

```
$ npm init
```

lalu enter semua dialog sampai berhasil membuat file *package.json*

lalu buatlah file *.gitignore* yang berisi

```
node_modules  
.env
```

\*mkdir = make directory  
\*cd = change directory

# Step (1) Penjelasan

---

```
$ npm init
```

Untuk membuat file package.json

File package.json jangan sampai hilang/terhapus karena berisi konfigurasi dasar

Setiap perubahan konfigurasi tercatat otomatis di package.json

# Step (1) Penjelasan

---

Mengapa perlu .gitignore ?

File .gitignore digunakan untuk memberi tahu Git **file atau folder mana yang tidak perlu dilacak (track)** atau disimpan di repository.

**Fungsinya:**

- Menghindari commit file yang bersifat pribadi atau sementara (contoh: file konfigurasi lokal, cache, log).
- Mencegah file besar atau hasil build (misalnya node\_modules) ikut tersimpan di GitHub, sehingga repository tetap ringan.

Biasanya Isi .gitignore :

- node\_modules/ → folder dependency tidak di-commit.
- .env → file environment (misal password, API key) tidak di-upload.
- \*.log → semua file log diabaikan.
- dist/ → folder hasil build tidak ikut di Git.

# Step (1) Penjelasan

---

node\_modules tidak dimasukkan ke repository karena beberapa alasan penting:

- Ukuran besar
- Bisa dibuat ulang kapan saja
- Menghindari Konflik Versi di Git
- Praktik Standar di Semua Project Node.js karena hampir semua developer memasukkan node\_modules ke .gitignore karena dianggap temporary build files, bukan bagian dari source code asli.

# Step (2)

---

lalu jalankan perintah install express

**npm install express**

**npm install nodemon**

lalu di scripts tambahkan

```
"scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1",  
    "start": "node index.js",  
    "dev": "nodemon index.js"  
},
```

# Step (2) Penjelasan

---

**Nodemon** adalah tool di Node.js yang memantau (watch) perubahan file di project kita dan secara otomatis me-restart server (**hot reload**) ketika ada perubahan kode.

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "start": "node index.js",  
  "dev": "nodemon index.js"  
},
```

Kalau kita tidak pakai nodemon, setiap kali mengedit file JavaScript, kita harus:  
CTRL + C # stop server  
node server.js # start server lagi

**Dengan nodemon**, cukup jalankan sekali: npx nodemon server.js

Lalu saat kamu ubah file (misalnya server.js), server akan otomatis restart dan langsung memuat kode baru. → HOT RELOAD

# Step (3)

buat file [index.js](#) lalu isi dengan kode dibawah ini:

---

```
const express = require('express');
const app = express();
const port = 3000;

// middleware untuk parsing JSON
app.use(express.json());

// Route sederhana
app.get('/', (req, res) => {
| res.send('Hello World dari Express!');
});

// Menjalankan server
app.listen(port, () => {
| console.log(`Server berjalan di http://localhost:\${port}`);
});
```

# Running

---

## 1 npm start

**Fungsi default** untuk menjalankan script "start" yang ada di package.json.

Kalau di package.json tertulis:

```
"scripts": { "start": "node server.js" }
```

maka npm start akan mengeksekusi:

```
node server.js
```

 Biasanya dipakai untuk **menjalankan server di mode produksi** (production).

# Running

---

## 2 npm run dev

Menjalankan script "dev" di package.json.

Contoh:

```
"scripts": { "dev": "nodemon server.js" }
```

maka npm run dev akan mengeksekusi:

nodemon server.js

📌 Biasanya dipakai untuk **mode development** karena nodemon akan otomatis restart server saat file diubah.

# Running

---

## 3 npx nodemon server.js

Langsung menjalankan nodemon **tanpa perlu menulis script di package.json.**

npx artinya menjalankan package yang ada di project atau yang di-install sementara.

Efeknya sama seperti "dev": "nodemon server.js", tapi dijalankan langsung lewat terminal.

---

# Static File Express.js

# Apa itu static file ?

---

Di **Express.js**, *static file* adalah file yang disajikan langsung ke browser **tanpa diubah atau diproses oleh server**.

Biasanya ini adalah asset front-end seperti:

- **HTML statis** (index.html, about.html)
- **CSS** (style.css)
- **JavaScript front-end** (app.js)
- **Gambar** (logo.png, banner.jpg)
- **Font, video, dll.**

# Cara Kerja

---

Express menyediakan **middleware bawaan** **express.static()** untuk melayani **file statis**.

Artinya, kalau sebuah file ada di folder yang kita tentukan sebagai *static*, Express akan langsung mengirimkan file itu ke browser tanpa routing tambahan.

# Konfigurasi Dasar Static File di Express

---

cara menggunakan Static File di express.js adalah seperti dibawah ini:

```
project/
|
└── public/
    ├── index.html
    ├── style.css
    └── script.js
|
└── server.js
```

```
const express = require('express');
const app = express();
const port = 3000;

// 1. Tentukan folder public sebagai
tempat file statis
app.use(express.static('public'));
```

ini berarti file yang ada dalam folder public itu **bisa langsung diakses tapi bersifat statis**, yang maksudnya file tersebut dikirim ke client tanpa diproses di server

Kalau kita jalankan:

- <http://localhost:3000/index.html> → langsung menampilkan index.html
- <http://localhost:3000/style.css> → langsung mengirim file CSS
- <http://localhost:3000/script.js> → langsung mengirim JS

# Kenapa pakai static file?

---

- **Efisien** – file dikirim apa adanya, tidak diproses tiap request.
- **Memisahkan back-end & front-end** – back-end hanya fokus pada API atau logika server.
- **Wajib untuk website** – CSS, JS, gambar, dan font perlu disajikan sebagai file statis.

---

# API

# API

---

**API** adalah singkatan dari **Application Programming Interface**, yaitu sebuah **jembatan** yang memungkinkan **dua aplikasi atau sistem** saling berkomunikasi dan bertukar data.

**Jenis API yang Umum :**

**1. Web API**

API yang diakses melalui internet, biasanya lewat HTTP (contoh: REST API, GraphQL).

**2. Library API**

API yang disediakan oleh bahasa pemrograman atau pustaka (contoh: DOM API di JavaScript).

**3. Hardware API**

API yang menghubungkan software dengan perangkat keras (contoh: API kamera di ponsel).

# Contoh web api sederhana :

---

Misalnya kita ingin ambil data cuaca dari API publik:

```
fetch('https://api.weatherapi.com/v1/current.json?key=API_KEY&q=Jakarta')
  .then(res => res.json())
  .then(data => {
    console.log(`Suhu di Jakarta: ${data.current.temp_c}°C`);
  });

```

Keterangan:

- Kita **mengirim request** ke API cuaca.
- API **mengirim response** dalam format JSON.
- Aplikasi kita **menampilkan data** yang diterima.

---

# Web Service API

# Apa itu Web Service API?

---

Web Service API adalah sebuah **web** yang menerima **request** dari client dan menghasilkan **response**, biasa berupa JSON/XML.

# Web Service

---

## **Web Service**

- **Jenis API** yang khusus berjalan di **web** (internet atau intranet).
- Menggunakan **protokol jaringan** seperti:
  - **HTTP/HTTPS** (paling umum).
  - SOAP (XML-based, format lama).
  - gRPC, dll.
- Biasanya digunakan antar server atau server-client.

**Analogi:** “*Web Service itu pelayan yang hanya bekerja di restoran online.*”

# Apa itu REST API?

---

REST API (Representational State Transfer):

- REST API adalah **konsep arsitektur yang mendefinisikan aturan** untuk membangun layanan web.
- Berfokus pada sumber daya (resources) dan tindakan (actions) yang diambil terhadap sumber daya tersebut.
- Menggunakan metode HTTP standar (GET, POST, PUT, DELETE) untuk berinteraksi dengan sumber daya.
- Tidak memiliki aturan khusus terkait dengan format data yang digunakan bisa menggunakan XML, JSON, atau format lainnya.

# REST API

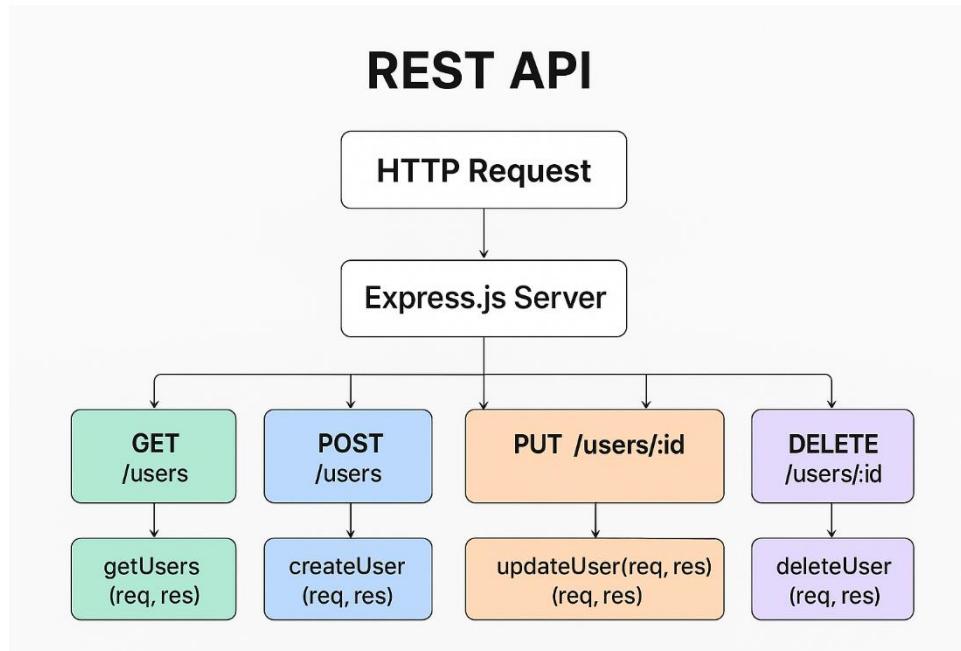
## (Representational State Transfer API)

---

- **Salah satu Gaya arsitektur** untuk membuat Web API.
- Menggunakan **HTTP method**:
  - GET → ambil data
  - POST → tambah data
  - PUT → update data. Memperbarui **seluruh** data resource berdasarkan ID.
  - PATCH → Memperbarui **sebagian** data resource
  - DELETE → hapus data berdasarkan id.
- Format data biasanya **JSON** (kadang XML).
- Sifatnya **stateless** (server tidak menyimpan status sesi client).

# Menggunakan HTTP method

---



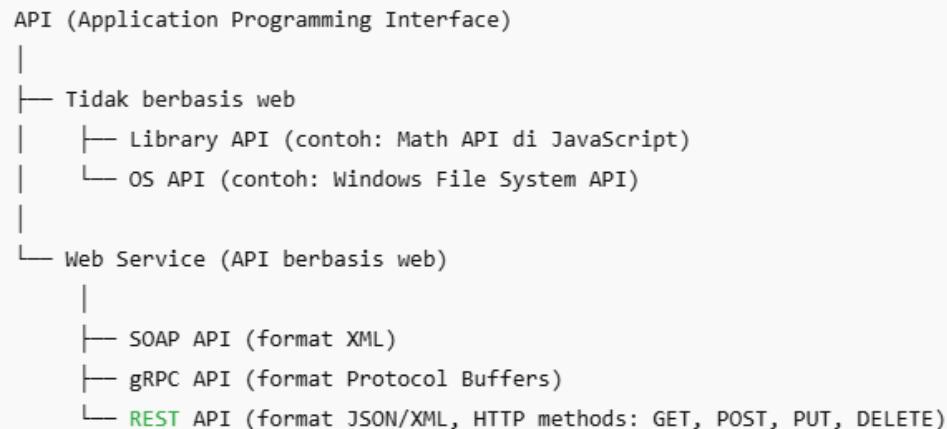
# Menggunakan HTTP method

---

Method	Tujuan	Idempotent (aman dijalankan berulang kali)	Mengubah Data?	Data di Body?
<b>GET</b>	Ambil data	✓	✗	✗
<b>POST</b>	Tambah data baru	✗	✓	✓
<b>PUT</b>	Update seluruh data	✓	✓	✓
<b>PATCH</b>	Update sebagian data	⚠️ Tergantung	✓	✓
<b>DELETE</b>	Hapus data	✓	✓	✗ (biasanya)

# Istilah-istilah yang perlu diingat

---



## Penjelasan Diagram

**1. API** → konsep paling luas. Semua interface yang memungkinkan program lain berkomunikasi.

**2. Web Service** → API yang berjalan di internet atau intranet (pakai protokol jaringan).

**3. REST API** → salah satu *gaya* membuat Web Service, menggunakan aturan REST dan biasanya format JSON.

# Apa itu JSON?

---

**JSON (JavaScript Object Notation)** adalah text format yang digunakan untuk menyimpan data, bentuk umumnya seperti **javascript object**.

Contoh format JSON:

```
{"nama": "John", "usia": 30}
```

---

# Basic Routing Express.js

# Basic Routing Express

---

cara menggunakan Routing pada express.js adalah seperti dibawah ini:

```
app.METHOD(PATH, HANDLER)
```

- app merupakan instance dari express
- METHOD diisi dengan HTTP Method  
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>
- PATH diisi dengan path dari route yang dibuat
- HANDLER diisi function yang harus memiliki dua parameter request dan response

# Basic Routing Express

---

**cara menggunakan Routing pada express.js adalah seperti dibawah ini:**

```
app.METHOD(path, [middleware1, middleware2, ...], handler)
```

METHOD = HTTP method (get, post, put, delete, dll),

path = string/pattern,

dan handler = fungsi (req, res, next) => { ... }. Express

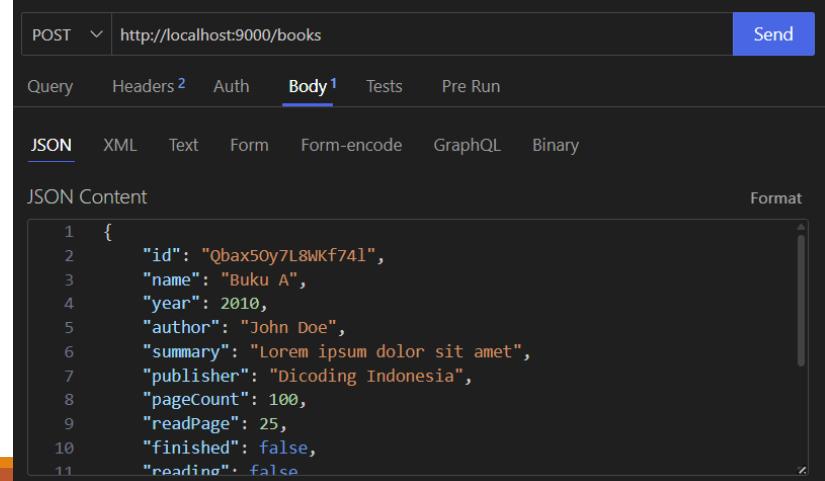
- 
- **req (Request)** → wadah semua informasi yang dikirim *client*. Objek ini merepresentasikan **permintaan** dari client (browser, Postman, aplikasi mobile, dll) ke server.
  - **res (Response)** → alat untuk membalas ke *client*. Objek ini digunakan server untuk **mengirimkan jawaban** kembali ke client.

# req (Request Object).. Yang sering dipakai

---

## **req.params, req.query, req.body —**

- req.params → nilai dari bagian path yang dinamis (/users/:id → req.params.id).
- req.query → data dari query string setelah ? (mis. /search?q=abc → req.query.q).
- req.body → data request body (POST/PUT) — **butuh** middleware parser seperti express.json() atau express.urlencoded(...).
- Contoh di pengujian Thunder Client:



# req (Request Object)

Objek ini merepresentasikan **permintaan dari client** (browser, Postman, aplikasi mobile, dll) ke server.  
Sumber data utama dalam req:

---

Properti	Kegunaan	Contoh
req.method	HTTP method yang dipakai	"GET", "POST", "PUT", "DELETE"
req.url	URL lengkap dari request	"/users/5?active=true"
req.path	Path tanpa query string	"/users/5"
req.params	<b>Route parameter</b> dari path	req.params.id jika rute /users/:id
req.query	<b>Query string</b> setelah ?	/search?q=tes → req.query.q = "tes"
req.body	Data body request (POST/PUT)	{ "name": "Cika" }
req.headers	Header HTTP dari client	req.headers['content-type']
req.ip	Alamat IP client	"127.0.0.1"

Catatan: req.body hanya tersedia jika sudah pakai middleware seperti express.json() atau express.urlencoded().

# res (Response Object)

Objek ini digunakan server untuk **mengirimkan jawaban** kembali ke client.

---

Method	Fungsi	Contoh
res.send(data)	Kirim teks, HTML, atau objek	res.send('Halo!')
res.json(obj)	Kirim JSON	res.json({ sukses: true })
res.status(code)	Set HTTP status code	res.status(404).send('Not Found')
res.redirect(url)	Redirect ke URL lain	res.redirect('/login')
res.render(view, data)	Render template view (jika pakai template engine)	res.render('index', { title: 'Home' })
res.set(header, value)	Set header HTTP	res.set('Content-Type', 'text/plain')
res.end()	Akhiri response tanpa data	res.end()

# Alur sederhananya

---

- 1.Client mengirim request ke server (GET/POST/...).
- 2.Express membuat objek req berisi semua informasi tentang request tersebut.
- 3.Express membuat objek res yang bisa kamu pakai untuk mengirim response.
- 4.Handler menerima (req, res) → ambil data dari req, kirim balasan lewat res.

# Contoh

---

Jika Client memanggil

```
POST /greet/Cika?age=25  
Body: { "hobby": "ngoding" }
```

Maka respon

```
{  
  "message": "Halo Cika, umur 25, hobi ngoding"  
}
```

```
Client (Browser / App)  
-----  
Kirim HTTP Request  
(GET, POST, PUT, DELETE)
```

```
Express.js Server  
(app.get, app.post, dll)
```

```
Express membuat objek:  
req → berisi info request  
res → untuk kirim balasan
```

```
Route Handler:  
app.METHOD(path, (req, res) => { ... }) |
```

```
Route Handler:  
app.METHOD(path, (req, res) => { ... }) |
```

```
Ambil data dari req    Kirim balasan via res  
(req.params,            (res.send, res.json,  
req.query,            res.status, res.end)  
req.body, dst.)
```

```
Response terkirim ke    |  
client & koneksi selesai|
```

# Route Modular

---

Untuk memisahkan rute jadi file terpisah. Bisa dipisahkan agar lebih modular dan rapi, karena logic routes dipisahkan dari file utama server.js.

Contoh penerapan :

```
// routes/users.js
const express = require('express');
const router = express.Router();

router.get('/', (req, res) => res.send('List users'));
router.get('/:id', (req, res) => res.send(`User ${req.params.id}`));

module.exports = router;

// index.js
const express = require('express');
const usersRouter = require('./routes/users');
const app = express();

app.use('/users', usersRouter); // mount router
```

***express.Router()*** adalah “mini-app” yang bisa memiliki middleware dan route sendiri — cocok untuk modularisasi (API versi, area fitur, dsb).

Sumber : <https://expressjs.com/en/guide/routing.html>

---

# Demo membuat REST API

# Contoh Data Dummy untuk Demo

---

```
let movies = [
    {id: 1, title: "Spider-Man", year: 2002},
    {id: 2, title: "John Wick", year: 2014},
    {id: 3, title: "The Avengers", year: 2012},
    {id: 4, title: "Logan", year: 2017},
]
```

# Contoh Kode Basic Routing untuk Movie

---

```
project-folder/
|
+-- server.js
|
+-- routes/
    +-- movies.js
```

# Contoh Kode Basic Routing untuk Server.js

---

```
const express = require('express'); // Import express

const app = express();           // Inisialisasi app

const port = 3000;              // Port server

const moviesRouter = require('./routes/movies'); // Import router movies

app.use(express.json());        // Middleware untuk parsing JSON

app.use('/api/movies', moviesRouter); // Mount router di prefix /api/movies

// Jalankan server

app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
});
```

# Contoh Kode Basic Routing untuk Movie.js

---

```
const express = require('express');

const router = express.Router() // buat router instance

// Data dummy
let movies = [
  { id: 1, title: 'Inception', year: 2010 },
  { id: 2, title: 'Interstellar', year: 2014 }
];
```

# Contoh Kode Basic Routing untuk Movie.js...(lanjutan)

---

```
// GET /api/movies
router.get('/', (req, res) => {
  res.json(movies);
});

// GET /api/movies/:id
router.get('/:id', (req, res) => {
  const movie = movies.find(m => m.id === parseInt(req.params.id));
  if (!movie) return res.status(404).json({ message: 'Movie not found' });
  res.json(movie);
});
```

# Contoh Kode Basic Routing untuk Movie.js...(lanjutan)

---

```
// POST /api/movies
router.post('/', (req, res) => {
  const newMovie = {
    id: movies.length + 1,
    title: req.body.title,
    year: req.body.year
  };
  movies.push(newMovie);
  res.status(201).json(newMovie);
});
```

# Contoh Kode Basic Routing untuk Movie.js ...(lanjutan)

---

```
// PUT /api/movies/:id
router.put('/:id', (req, res) => {
  const movie = movies.find(m => m.id === parseInt(req.params.id));
  if (!movie) return res.status(404).json({ message: 'Movie not found' });

  movie.title = req.body.title || movie.title;
  movie.year = req.body.year || movie.year;

  res.json(movie);
});
```

# Contoh Kode Basic Routing untuk Movie.js ...(lanjutan)

---

```
// DELETE /api/movies/:id
router.delete('/:id', (req, res) => {
  movies = movies.filter(m => m.id !== parseInt(req.params.id));
  res.status(204).end();
});

module.exports = router; // export router
```

# Testing

---

## 3. Testing di Thunder Client

Setelah server jalan (`node server.js`), buka Thunder Client (di VSCode).

- GET semua movies

- Method: GET

- URL: `http://localhost:3000/api/movies`

- Response:

```
json
```

```
[  
  { "id": 1, "title": "Inception", "year": 2010 },  
  { "id": 2, "title": "Interstellar", "year": 2014 }  
]
```

- GET movie by ID

- Method: GET

- URL: `http://localhost:3000/api/movies/1`

- Response:

```
json
```

```
{ "id": 1, "title": "Inception", "year": 2010 }
```

# Testing

---

- **POST tambah movie**

- Method: POST
- URL: `http://localhost:3000/api/movies`
- Body → JSON:

```
json
```

```
{ "title": "The Dark Knight", "year": 2008 }
```

- Response:

```
json
```

```
{ "id": 3, "title": "The Dark Knight", "year": 2008 }
```

- **PUT update movie**

- Method: PUT
- URL: `http://localhost:3000/api/movies/2`
- Body → JSON:

```
json
```

```
{ "title": "Interstellar Updated", "year": 2014 }
```

- Response:

```
json
```

```
{ "id": 2, "title": "Interstellar Updated", "year": 2014 }
```

- **DELETE movie**

- Method: DELETE
- URL: `http://localhost:3000/api/movies/1`
- Response: (204 No Content) → artinya berhasil dihapus.

---

# Middleware & alur eksekusi

More info : <https://expressjs.com/en/guide/using-middleware.html>

# Middleware

---

- Middleware adalah **satu satunya mekanisme keamanan website**.
- Middleware berjalan sebelum fungsi utama di proses → berjalan **di antara** proses penerimaan request dari client dan pengiriman response oleh server.
- Middleware bertugas “menyaring”, “mengolah”, atau “menambahkan” sesuatu pada request/response sebelum dikirim ke handler utama atau sebelum diakhiri.

Contoh :

misalnya terdapat routing untuk mengarahkan ke dalam fungsi, fungsi tersebut digunakan untuk melihat data. Agar fungsi tersebut tidak dapat diakses orang maka harus melewati middleware, middleware berperan untuk menghentikan atau meneruskan proses.

# Middleware

---

- Aplikasi Express pada dasarnya adalah rangkaian dari pemanggilan fungsi-fungsi *middleware*.
- Middleware bisa dipasang di level aplikasi (`app.use()`), router (`router.use()`) atau per-route (parameter di `app.get(path, mw, handler)`).
- **Function Middleware** adalah fungsi yang memiliki akses terhadap objek **request (req)**, objek **response (res)**, dan fungsi *middleware* berikutnya dalam siklus *request-response* aplikasi. Fungsi *middleware* berikutnya biasanya ditulis dengan nama variabel **next**.
- Jika *middleware* yang sedang berjalan tidak mengakhiri siklus *request-response*, maka ia **harus** memanggil `next()` agar alur dapat diteruskan ke *middleware/handler* berikutnya. Jika tidak, maka *request* akan tertahan dan tidak akan terselesaikan.
- Untuk melewaskan error gunakan `next(err)` — Express akan mem-forward ke error-handling middleware.

# Middleware

---

- **Function khusus** yang punya akses ke:
  - req → objek request
  - res → objek response
  - next() → fungsi untuk melanjutkan ke middleware berikutnya
- Bisa digunakan untuk **semua route** atau **route tertentu**.
- Jadi, fungsi *middleware* dapat melakukan beberapa tugas, antara lain:
  - Menjalankan kode apa pun.
  - Melakukan perubahan pada objek **request** dan **response**.
  - Mengakhiri siklus *request-response*.
  - Bisa **menghentikan** proses dan langsung mengirim response
  - Memanggil fungsi *middleware* berikutnya di dalam stack.

# Contoh Penggunaan Middleware yang Umum

---

- **Built-in Middleware**
  - express.json() → parsing JSON body
  - express.urlencoded() → parsing form data
  - express.static() → melayani file statis
- **Custom Middleware**
  - Logging request
  - Authentikasi / otorisasi
  - Validasi data
- **Third-Party Middleware**
  - morgan → logging
  - cors → mengaktifkan CORS
  - helmet → keamanan HTTP header
  - ....dst

# Membuat Middleware

```
const express = require('express')
const app = express()
const port = 8080

app.use(express.json()) // middleware untuk parsing JSON

const logMiddleware = (req, res, next) =>{
  console.log("Ini dari middleware Log....")
  next()
}

const getText = (req,res)=>{
  res.status(200).json({text: "this is text"})
}

app.get('/text', logMiddleware, getText)

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

// middleware untuk parsing JSON  
app.use(express.json());

# Membuat Middleware

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

// Middleware logging
const logger = (req, res, next) => {
  console.log(`${req.method} ${req.url} at ${new Date().toISOString()}`);
  next(); // lanjut ke middleware berikutnya atau handler route
};

app.use(logger); // pasang middleware ke semua request

// Route utama
app.get('/', (req, res) => {
  res.send('Halo dari Express!');
});

app.listen(3000, () => {
  console.log('Server berjalan di http://localhost:3000');
});
```

//middleware log  
Untuk mencatat request berupa method dan URL.

## Alurnya:

- 1.Client kirim request.
- 2.Middleware logger dijalankan → mencatat method dan URL.
- 3.next() dipanggil → lanjut ke handler route /.
- 4.Server mengirim response.

# Middleware Basic Auth

```
// simple-basic-auth-server.js
// Example Express.js app with a custom middleware implementing Basic Auth (no JWT)

const express = require('express');
const app = express();
const PORT = 3000;

// Built-in middleware to parse JSON bodies
app.use(express.json());

// Custom middleware: Basic Auth checker
function basicAuth(req, res, next) {
  const authHeader = req.headers['authorization'];
  if (!authHeader) {
    res.setHeader('WWW-Authenticate', 'Basic');
    return res.status(401).send('Authentication required');
  }
  // Decode Base64 part of the header "Basic base64encoded"
  const base64Credentials = authHeader.split(' ')[1];
  const credentials = Buffer.from(base64Credentials, 'base64').toString('ascii');
  const [username, password] = credentials.split(':');

  // Very simple hardcoded check
  if (username === 'admin' && password === 'password') {
    req.user = { username }; // attach user info to request
    return next(); // allow access
  }
  return res.status(403).send('Forbidden: Wrong credentials');
}

// Public route
app.get('/', (req, res) => {
  res.send('Welcome to the public API');
});

// Protected route (requires Basic Auth)
app.get('/secret', basicAuth, (req, res) => {
  res.send(`Hello ${req.user.username}, this is top secret!`);
});

app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

# Middleware Basic Auth

---

Berikut panduan uji di Thunder Client:

## 1. Public route

- GET `http://localhost:3000/` → Tidak perlu Authorization → Respon: "Welcome to the public API".

## 2. Protected route

- GET `http://localhost:3000/secret`
- Di tab *Auth* Thunder Client pilih **Basic Auth**, isi:
  - Username: `admin`
  - Password: `password`
- Respon: "Hello admin, this is top secret!".
- Jika salah user/pass → `403 Forbidden`.
- Jika tidak kirim auth → `401 Authentication required`.

# Kode Status / HTTP response status codes

---

- Kode status respons HTTP menunjukkan apakah permintaan HTTP tertentu telah berhasil diselesaikan.
- Kode status respons adalah kode integer tiga digit yang menjelaskan hasil permintaan dan responnya, termasuk apakah permintaan berhasil dan konten apa yang disertakan (jika ada).
- Digit pertama kode status menentukan kelas respons. Dua digit terakhir tidak memiliki peran kategorisasi apa pun.
- Kode status yang tercantum didefinisikan oleh [RFC 9110](#).

# Kode Status / HTTP response status codes

---

Ada lima nilai untuk digit pertama:

- 1xx (Informational)
- 2xx (Successful)
- 3xx (Redirection)
- 4xx (Client Error)
- 5xx (Server Error)

# Kode Status / HTTP response status codes

---

Respons dikelompokkan dalam lima kelas:

1. [Informational responses](#) (100 – 199) : The request was received, continuing process
2. [Successful responses](#) (200 – 299) : The request was successfully received, understood, and accepted
3. [Redirection messages / Pesan Pengalihan](#) (300 – 399) : Further action needs to be taken in order to complete the request
4. [Client error responses](#) (400 – 499) : The request contains bad syntax or cannot be fulfilled
5. [Server error responses](#) (500 – 599) : The server failed to fulfill an apparently valid request

# Menangani error & 404

---

- Error-handling middleware: signature khusus (err, req, res, next) — hanya dipanggil ketika ada error atau next(err):

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ error: 'Something broke' });
});
```

- **404:** tambahkan **middleware terakhir** yang menangani request yang tidak ditangani route manapun:

```
app.use((req, res) => {
  res.status(404).send('Not Found');
});
```

# Tugas Praktek

Ikutilah langkah-langkah pengerjaan di bawah ini:

## 1. Menambahkan Folder baru dan file baru

Gunakanlah repository tugas (**jangan buat repository baru lagi**). Lalu buatlah 1 folder baru dalam folder "tugas-pengenalan-express"

## 2. Kerjakan Soal di bawah ini

buatlah **project express baru** dan tambahkan file `.gitignore` didalamnya.

masukkan kode dibawah ini dalam file `.gitignore`:

```
node_modules
```

lalu kerjakan soal dibawah ini:

### soal 1

buatlah route GET dengan path **"/lingkaran-tabung"**

lalu pada handlernya buatlah function luas lingkaran dan keliling lingkaran dan volume tabung. jari-jari yang di gunakan adalah 7 (jari jari ini masukan dalam parameter url). lalu tinggi tabungnya adalah 10 (tinggi ini masukan dalam parameter url)

lalu tampilkan text seperti ini: **jariJari : 7, tinggi: 10, volume tabung : ...., luas alas tabung : ...., keliling alas tabung : ....**

untuk titik2nya harap diisi dengan hasil perhitungannya (**wajib menggunakan parameter url untuk menjawab soal ini**)

# Tugas Praktek

## soal 2

buatlah route GET dengan path "**/data-orang**"

lalu buatlah handler untuk route tersebut dan tambahkan kode dibawah ini didalam handler tersebut

```
let dataOrang =[  
    {id: 1, name: "John", umur: 30, pekerjaan: "Penulis", jenisKelamin: "L"},  
    {id: 4, name: "Benzema", umur: 34, pekerjaan: "Pemain Bola", jenisKelamin: "L"},  
    {id: 5, name: "Sarah", umur: 27, pekerjaan: "Model", jenisKelamin: "P"},  
    {id: 9, name: "Shohei Ohtani", umur: 28, pekerjaan: "Pemain Baseball", jenisKelamin: "L"},  
    {id: 11, name: "Maria Sharapova", umur: 35, pekerjaan: "Petenis", jenisKelamin: "P"}  
]
```

handler ini dapat menerima parameter url berupa umur dan gender

jika **url parameter umur** diisi maka tampilkan data dengan umur diatas atau sama dengan **url parameter umur** tersebut

jika **gender** diisi maka tampilkan gender yang dipilih

berikut untuk contoh text yang ditampilkan jika parameter umur diisi 30 dan gender diisi "L":

1. John - Pekerjaan: Penulis - Umur: 30 Tahun
2. Benzema - Pekerjaan: Pemain Bola- Umur: 34 Tahun

# Tugas Praktek

## soal 3

buatlah route GET dengan path "/data-orang/:id"

lalu buatlah handler untuk route tersebut dan tambahkan kode dibawah ini didalam handler tersebut

```
let dataOrang =[  
    {id: 1, name: "John", umur: 30, pekerjaan: "Penulis", jenisKelamin: "L"},  
    {id: 4, name: "Benzema", umur: 34, pekerjaan: "Pemain Bola", jenisKelamin: "L"},  
    {id: 5, name: "Sarah", umur: 27, pekerjaan: "Model", jenisKelamin: "P"},  
    {id: 9, name: "Shohei Ohtani", umur: 28, pekerjaan: "Pemain Baseball", jenisKelamin: "L"},  
    {id: 11, name: "Maria Sharapova", umur: 35, pekerjaan: "Petenis", jenisKelamin: "P"}  
]
```

jika setelah menggunakan parameter id data tidak ditemukan maka munculkan text:

**Maaf data tidak ditemukan**

jika data ditemukan maka munculk text :

**Pak John adalah seorang Penulis yang berusia 30 tahun**

perlu diperhatikan untuk kata Pak, John, Penulis dan angka 30 didapatkan dari variabel dataOrang diatas



KEMNAKER



# Kesimpulan

---

- Apa itu Express.js
- Instalasi Express.js
- Basic Routing Express.js
- Static File Express.js
- Web Service API
- Middleware di Express.js

---

Terima Kasih