



# Backend Basic Node.js & Express.js

---

Introduction to Node.js and Express.js for  
building server-based web applications.

# What is Node.js?

- Node.js adalah runtime JavaScript yang berjalan di server.
- Memungkinkan pengembangan aplikasi backend dengan menggunakan JavaScript.
- Dapat menangani banyak permintaan sekaligus secara efisien (non-blocking I/O).



# What is Express.js?


Express.js adalah sebuah framework web minimalis yang berjalan di atas Node.js, dirancang untuk mempermudah pengembangan aplikasi web dan API.

Beberapa fitur utama Express.js:

1. **Framework minimalis:** Meskipun sederhana, Express bisa diperluas dengan modul tambahan sesuai kebutuhan.
2. **Penanganan Rute yang Mudah:** Express menyediakan cara mudah untuk mendefinisikan berbagai rute yang menangani URL tertentu pada aplikasi Anda.
3. **Dukungan Middleware:** Anda dapat menyisipkan beberapa middleware untuk mengelola permintaan sebelum memberikan respon, contohnya untuk autentikasi pengguna atau mengelola file.



# Project Initialization

1. Install Express Generator  
`npm install -g express-generator`
  2. Membuat project baru  
`express myapp`
  3. Install dependencies  
`npm install`
  4. Menjalankan aplikasi  
`npm start`
- 

# Express Project Structure

Express Generator secara otomatis membuat struktur direktori yang sudah terorganisir, meliputi:

1. **bin/**: Berisi file untuk memulai server.
2. **public/**: Menyimpan aset statis seperti file CSS, JavaScript, dan gambar.
3. **routes/**: Berisi definisi rute (misalnya index.js untuk halaman utama dan users.js untuk rute pengguna).
4. **views/**: Menyimpan template untuk merender halaman HTML (misalnya menggunakan Pug).
5. **app.js**: File utama aplikasi yang menginisialisasi Express, middleware, dan rute.

Dengan struktur ini, Anda dapat langsung memulai membangun fitur aplikasi tanpa harus menyiapkan setiap komponen dari nol.

# What is Middleware in Express.js?

Middleware adalah fungsi yang dijalankan pada setiap permintaan ke server sebelum rute menangani permintaan tersebut.

Middleware dapat digunakan untuk:

1. Mengautentikasi pengguna.
2. Mengubah atau memproses data dari permintaan.
3. Menangani kesalahan yang terjadi di server.

Express Generator menyediakan beberapa middleware bawaan, seperti:

1. `logger`: Untuk mencatat setiap permintaan yang datang.
2. `errorHandler`: Untuk menangani kesalahan.

# Auth Middleware Example

```
1  const jwt = require('jsonwebtoken');
2  const { JWT_SECRET } = process.env;
3
4  ✓ const AuthMiddleware = (req, res, next) => {
5      const token = req.headers.authorization?.split(' ')[1];
6      if (!token) {
7          return res.status(401).send('Access denied');
8      }
9
10     try {
11         const decoded = jwt.verify(token, JWT_SECRET);
12         req.user = decoded;
13         next();
14     } catch (err) {
15         res.status(400).send('Invalid token');
16     }
17 };
18
19 module.exports = AuthMiddleware;
```

```
1  const express = require('express');
2  const ProductController = require('../controllers/productController');
3  const AuthMiddleware = require('../middleware/authMiddleware');
4
5  const router = express.Router();
6
7  router.get('/', ProductController.getAllProducts);
8  router.get('/:id', ProductController.getProductById);
9  router.post('/', AuthMiddleware, ProductController.createProduct);
10 router.put('/:id', AuthMiddleware, ProductController.updateProduct);
11 router.delete('/:id', AuthMiddleware, ProductController.deleteProduct);
12
13 module.exports = router;
```

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

# Creating REST API with Express

REST API (Representational State Transfer) adalah standar dalam pengembangan web yang memungkinkan aplikasi untuk saling berkomunikasi melalui HTTP.

Dalam REST API, kita biasanya membuat endpoint yang merespons berbagai jenis permintaan HTTP seperti:

- GET** : Untuk mengambil data.
- POST** : Untuk mengirim data baru.
- PUT** : Untuk memperbarui data.
- DELETE** : Untuk menghapus data.



# Example of REST API

```
app.get('/api/users', (req, res) => {  
  res.json([  
    { id: 1, name: 'Bambang Dwi' },  
    { id: 2, name: 'Muhamad Kharis' }  
  ]));
```

```
app.post('/api/users', (req, res) => {  
  const newUser = req.body;  
  res.status(201).json(newUser);  
});
```

# Managing Routes with Router

Router memungkinkan kita untuk mengelola dan memecah rute aplikasi menjadi modul yang lebih kecil dan terorganisir. Ini sangat penting terutama saat mengembangkan aplikasi besar, di mana terdapat banyak rute yang harus dikelola.

Dengan menggunakan Router, kita bisa membuat kelompok rute terpisah untuk bagian-bagian tertentu dari aplikasi, seperti rute untuk pengguna, produk, atau artikel. Setiap kelompok rute bisa ditangani di file terpisah, sehingga kode menjadi lebih mudah dibaca dan dipelihara.

# Example using Router

user.js

```
router.get('/', (req, res) => {  
  res.send('Daftar Pengguna');  
});  
  
router.post('/', (req, res) => {  
  const newUser = req.body;  
  res.status(201).json(newUser);  
});
```

product.js

```
router.get('/', (req, res) => {  
  res.send('Daftar Produk');  
});  
  
router.post('/', (req, res) => {  
  const newProduct = req.body;  
  res.status(201).json(newProduct);  
});
```

```
const express = require('express');  
const app = express();  
const usersRouter = require('./routes/users');  
const productsRouter = require('./routes/products');  
  
app.use(express.json());  
  
app.use('/api/users', usersRouter);  
app.use('/api/products', productsRouter);  
  
app.listen(3000, () => {  
  console.log('Server berjalan pada port 3000');  
});
```

# Run Server Express



---

npm start



# Thanks !

---

Any questions?





# Task!

No task!

