1. Membuat Server Dasar & Express.js

Node.js adalah:

- runtime JavaScript di server
- dibangun menggunakan V8 engine (Chrome)
- bersifat single-thread, non-blocking
- sangat cepat untuk API, backend, dan real-time apps

Sederhananya: JavaScript yang biasanya berjalan di browser, sekarang bisa berjalan di server.

2. Membuat Server HTTP Tanpa Framework

Node.js bisa membuat server minimal tanpa Express.

1. **Membuat File Server**

   server.js

```
import http from "http";

const server = http.createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/plain" });
  res.write("Halo! Ini server Node.js pertamamu");
  res.end();
});

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

2. **Jalankan server**

```
node server.js
```

   Buka browser:
   ➡ http://localhost:3000/

Memahami Request & Response

req = permintaan client
res = jawaban dari server

Contoh:

```
// server.js
const http = require("http");

const server = http.createServer((req, res) => {
  if (req.url === "/api/user" && req.method === "GET") {
    // Header wajib jika mengirim JSON
    res.writeHead(200, {
      "Content-Type": "application/json",
    });

    const data = {
      success: true,
      message: "Data user berhasil diambil",
      data: {
        id: 1,
        name: "Jamal Apriadi",
        email: "jamal.apriadi@gmail.com",
      },
    };

    // Kirim JSON
    res.end(JSON.stringify(data));
  } else {
    res.writeHead(404, { "Content-Type": "application/json" });
    res.end(JSON.stringify({ message: "Route tidak ditemukan" }));
  }
});

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

Routing Sederhana Tanpa Framework

Buat routing manual berdasarkan `req.url`.

server.js

```
import http from "http";

const server = http.createServer((req, res) => {
  res.setHeader("Content-Type", "application/json");

  if (req.url === "/") {
    res.end(JSON.stringify({ message: "Home Page" }));
  }
  else if (req.url === "/about") {
    res.end(JSON.stringify({ message: "About Page" }));
  }
  else {
    res.writeHead(404);
    res.end(JSON.stringify({ error: "Not Found" }));
  }
});

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

Membaca Body Request (POST)

Untuk POST request, perlu mendengarkan event `data` dan `end`.

Contoh menerima data JSON:

```
if (req.url === "/api/data" && req.method === "POST") {
  let body = "";

  req.on("data", chunk => {
    body += chunk;
  });

  req.on("end", () => {
    const jsonData = JSON.parse(body);

    res.end(JSON.stringify({
      message: "Data diterima",
      data: jsonData
    }));
  });
}
```

3. EXPRESS.JS

Setelah memahami native server, kita naik level ke **Express.js**, framework paling populer di Node.

---

1. Apa Itu Express.js?

Express.js adalah framework Node.js minimalis yang membuat:

✔ Routing jauh lebih mudah
✔ Penanganan request sederhana
✔ Middleware powerful
✔ API development lebih cepat
✔ JSON handling otomatis

Dengan Express, kita tidak perlu:

- membuat routing manual
- parsing request body secara manual
- menulis banyak kode boilerplate

2. Instalasi Express.js

Masuk ke folder project:

```
npm install express
```

3. Membuat Server Express Pertama

Buat file:

app.js

```
import express from "express";

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

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

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

Jalankan:

```
node app.js
```

4. Routing di Express.js

Express sangat mudah untuk membuat routing.

- **Route GET**

```
app.get("/about", (req, res) => {
  res.json({ message: "Halaman About" });
});
```

- **Route POST**

```
app.post("/kirim", (req, res) => {
  res.json({ message: "Data berhasil dikirim" });
});
```

- **ALL METHOD (opsional)**

```
app.all("/tes", (req, res) => {
  res.send("Semua method diterima");
});
```

5. Mengirim Response JSON

Sangat mudah:

```
app.get("/json", (req, res) => {
  res.json({
    sukses: true,
    data: {
      nama: "Jamal Apriadi",
      kelas: "Backend 101"
    }
  });
});
```

6. Request Query & Params

- **Query**

  URL:

  http://localhost:3000/search?keyword=nodejs

  Code:

```
app.get("/search", (req, res) => {
  const keyword = req.query.keyword;
  res.json({ message: "Pencarian", keyword });
});
```

- **Params (bagian dari URL)**

  URL:

  /user/17

  Code:

```
app.get("/user/:id", (req, res) => {
  res.json({
    message: "Detail user",
    id: req.params.id
  });
});
```

7. Handling POST (Body Parser)

Untuk menerima body JSON:

```
app.use(express.json());
```

Lalu:

Route POST

```
app.post("/tambah", (req, res) => {
  const data = req.body;

  res.json({
    message: "Data diterima",
    data: data,
  });
});
```

Request body contoh:

```
{
  "nama": "Budi",
  "kelas": "TI-2B"
}
```

8. Middleware di Express

Middleware adalah fungsi yang dieksekusi **sebelum router** menangani permintaan.

Contoh universal middleware (logging):

```
app.use((req, res, next) => {
  console.log(`Request masuk: ${req.method} ${req.url}`);
  next();
});
```

Contoh middleware hanya untuk satu route:

```
const cekToken = (req, res, next) => {
  if (req.query.token === "123") next();
  else res.status(401).json({ error: "Token salah!" });
};
```

```
app.get("/admin", cekToken, (req, res) => {
  res.json({ message: "Halo admin!" });
});
```

9. Struktur Folder Express Lebih Rapi

Berikut adalah contoh folder yang digunakan pada project nodejs express:

```
project/
  ├─ app.js
  ├─ routes/
  │    └─ mahasiswa.js
  ├─ controllers/
  └─ config/
```

4. Koneksi Database MySQL + Query Dasar + CRUD Dasar di Node.js

- Menghubungkan Node.js + Express ke database MySQL
- Memahami perbedaan `mysql2` (native) & ORM (Sequelize)
- Melakukan query dasar: SELECT, INSERT, UPDATE, DELETE
- Membuat controller CRUD pertama yang terkoneksi database
- Menggunakan environment variable untuk konfigurasi DB
- Menguji CRUD menggunakan Postman / Thunder Client

A. Instalasi MySQL dan Tools

Mahasiswa harus sudah menginstal:

✔ XAMPP / MAMP / WAMP
✔ MySQL Server
✔ MySQL Workbench / phpMyAdmin

B. Intallasi Project

```
npm init -y
```

ubah file package.json dari type **commonjs** menjadi **module**.

C. Install ExpressJs

```
npm install express
```

D. Membuat Database Untuk Project

Jalankan:

```
CREATE DATABASE nodejs_workshop;
```

Buat tabel users:

```
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

E. Instalasi Library mysql2

Masuk ke project sebelumnya:

```
npm install mysql2
```

F. Struktur Folder

```
project/
 ├─ config/
 │   └─ db.js
 ├─ controllers/
 │   └─ userController.js
 ├─ routes/
 │   └─ userRoutes.js
 ├─ server.js
 └─ .env
```

G. Setup Environment Variable (.env)

Install dotenv

```
npm install dotenv
```

Buat file `.env`:

```
PORT=3000

DB_HOST=localhost
DB_USER=root
DB_PASS=
DB_NAME=nodejs_workshop
```

NOTE:

- Sesuaikan DB_PASS dengan password MySQL kalian
- Jika XAMPP: default password kosong

H. DEBUGGING NODE MENGGUNAKAN nodemon

Instalasi:

```
npm install -D nodemon
```

Edit package.json:

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

Jalankan:

```
npm run dev
```

I. Membuat Koneksi Database (config/db.js)

```
import mysql from 'mysql2';
import dotenv from 'dotenv';

dotenv.config();

const db = mysql.createConnection({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME
});

db.connect((err) => {
  if (err) {
    console.error("Error koneksi ke database:", err);
    return;
  }
  console.log("MySQL Connected!");
});

export default db;
```

J. Membuat Controller CRUD (controllers/userController.js)

- **Get All Users (SELECT)**

```
import db from '../config/db.js';

export const getUsers = (req, res) => {
  db.query("SELECT * FROM users", (err, results) => {
    if (err) return res.status(500).json({ message: err });

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

- **Get User by ID**

```
export const getUserById = (req, res) => {
  const { id } = req.params;

  db.query("SELECT * FROM users WHERE id = ?", [id], (err,
results) => {
    if (err) return res.status(500).json({ message: err });

    if (results.length === 0)
      return res.status(404).json({ message: "User tidak
ditemukan" });

    res.json(results[0]);
  });
};
```

- **Create User (INSERT)**

```
export const createUser = (req, res) => {
  const { name, email } = req.body;

  db.query("INSERT INTO users (name, email) VALUES (?, ?)",
    [name, email],
    (err, results) => {
      if (err) return res.status(500).json({ message: err });

      res.json({ id: results.insertId, name, email });
    }
  );
};
```

- **Update User (UPDATE)**

```
export const updateUser = (req, res) => {
  const { id } = req.params;
  const { name, email } = req.body;

  db.query("UPDATE users SET name=?, email=? WHERE id=?",
    [name, email, id],
    (err, results) => {
      if (err) return res.status(500).json({ message: err });

      res.json({ message: "User berhasil diupdate" });
    }
  );
};
```

- **Delete User (DELETE)**

```
export const deleteUser = (req, res) => {
  const { id } = req.params;

  db.query("DELETE FROM users WHERE id=?", [id], (err, results)
=> {
    if (err) return res.status(500).json({ message: err });

    res.json({ message: "User berhasil dihapus" });
  });
};
```

K. Membuat Routing CRUD (routes/userRoutes.js)

```
import express from 'express';
import {
  getUsers,
  getUserById,
  createUser,
  updateUser,
  deleteUser
} from '../controllers/userController.js';

const router = express.Router();

router.get('/', getUsers);
router.get('/:id', getUserById);
router.post('/', createUser);
router.put('/:id', updateUser);
router.delete('/:id', deleteUser);

export default router;
```

L. Registrasi Router Pada server.js

```
import express from 'express';
import userRoutes from './routes/userRoutes.js';
import dotenv from 'dotenv';

dotenv.config();

const app = express();
app.use(express.json());

app.use('/users', userRoutes);

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

M. Pengujian Menggunakan Postman / Thunder Client

Method	Endpoint	Body (JSON)
GET	/users	-
GET	/users/1	-
POST	/users	{ "name": "Budi", "email": "budi@mail.com" }
PUT	/users/1	{ "name": "Budi Edit", "email": "new@mail.com" }
DELETE	/users/1	-

Tugas Praktikum Node.js & Express.js — CRUD API

Silakan kerjakan tugas berikut:

1. **Buat dua buah tabel database**:
    - **categories**
      Kolom minimal: `id`, `name`, `created_at`, `updated_at`
    - **products**
      Kolom minimal: `id`, `category_id`, `name`, `price`, `created_at`, `updated_at`
2. **Buat REST API menggunakan Node.js + Express.js** yang terdiri dari fitur CRUD untuk:
    - **Categories**
        - Create kategori
        - Read (list seluruh kategori & detail kategori)
        - Update kategori
        - Delete kategori
    - **Products**
        - Create produk
        - Read (list seluruh produk & detail produk)
        - Update produk
        - Delete produk
3. API harus menggunakan format **JSON** untuk request dan response.
4. Struktur project harus rapi dan mudah dipahami (routing, controller, dan koneksi database dipisah).
5. **Upload seluruh project ke GitHub** dengan nama repository:
   `nodejs-crud-restapi`
6. Sertakan pada repository:
    - README.md berisi cara menjalankan project
    - Screenshot hasil uji coba API (Postman/Insomnia)