# 📆 Hari 4 (Kamis) – Struktur Folder MVC

## 🎯 Tujuan Pembelajaran

- Memahami konsep arsitektur Model-View-Controller (MVC).
- Mampu menerapkan struktur folder MVC pada project backend Express.js.
- Menyusun kode ke dalam layer Model, Controller, dan Route yang terpisah.

---

## 🧠 Materi Inti (2 Jam)

### 1. Konsep Arsitektur MVC

- **Apa itu MVC?** MVC adalah pola arsitektur perangkat lunak yang memisahkan aplikasi menjadi tiga bagian utama yang saling terhubung:
  - **Model:** Merepresentasikan data dan logika bisnis. Bertanggung jawab untuk berinteraksi dengan database (CRUD operations) dan validasi data.
  - **View:** Merepresentasikan antarmuka pengguna. Dalam konteks backend API, View seringkali diabaikan atau digantikan oleh JSON response yang dikirimkan oleh Controller.
  - **Controller:** Bertindak sebagai perantara antara Model dan View. Menerima input dari pengguna (request), memprosesnya (memanggil Model untuk data atau logika bisnis), dan menyiapkan output (memilih View atau mengirimkan response JSON).
- **Manfaat MVC:**
  - **Separation of Concerns:** Memisahkan logika aplikasi menjadi bagian-bagian yang lebih terkelola.
  - **Maintainability:** Kode lebih mudah dipahami dan diperbaiki.
  - **Scalability:** Memudahkan penambahan fitur baru tanpa mengganggu bagian lain.
  - **Testability:** Setiap komponen (Model, Controller) dapat diuji secara independen.

### 2. Menerapkan Struktur Folder MVC pada Express.js

- Meskipun Express.js tidak secara ketat memaksa pola MVC, kita bisa mengadopsi struktur folder yang mencerminkan prinsip MVC.
- **Contoh Struktur Folder:**

```
backend-app/
  ├── src/
  │   ├── models/       # Interaksi dengan database (menggunakan pg,
prisma, atau ORM lain)
  │   ├── controllers/ # Logika bisnis, memproses request, memanggil
models
  │   ├── routes/      # Definisi endpoint dan menghubungkannya ke
controllers
  │   ├── middlewares/ # Fungsi perantara (autentikasi, otorisasi,
logging)
  │   ├── config/      # Konfigurasi aplikasi (database, secret keys)
  │   └── app.js       # File utama, setup Express, koneksi database,
```

```
dll.
  ├── package.json
  └── .env
```

- **Alur Request-Response dalam Struktur MVC:**
  1. Request masuk ke `app.js`.
  2. Request diteruskan ke Router yang sesuai di folder `routes/`.
  3. Router memanggil Middleware (jika ada, misalnya otorisasi).
  4. Middleware memanggil Controller yang sesuai di folder `controllers/`.
  5. Controller memanggil Model di folder `models/` untuk berinteraksi dengan database.
  6. Model mengembalikan data ke Controller.
  7. Controller menyiapkan response (misalnya, JSON) dan mengirimkannya kembali ke klien.

## 3. Menyusun Kode ke dalam Layer Terpisah

- **Routes:** Hanya berisi definisi endpoint dan memanggil fungsi Controller yang relevan.

```
// src/routes/userRoutes.js
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');
const authMiddleware = require('../middlewares/auth');

router.get('/:id', authMiddleware, userController.getUserById);
router.put('/:id', authMiddleware, userController.updateUser);
// ... endpoint lain

module.exports = router;
```

- **Controllers:** Berisi logika bisnis utama. Menerima `req` dan `res`, memanggil fungsi dari Model, dan mengirimkan response.

```
// src/controllers/userController.js
const userModel = require('../models/userModel');

exports.getUserById = async (req, res) => {
  try {
    const userId = req.params.id;
    const user = await userModel.findById(userId);
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.status(200).json(user);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

```
exports.updateUser = async (req, res) => {
  // ... logika update user, panggil userModel.update
};
```

- **Models:** Berisi logika interaksi database. Fungsi-fungsi di sini akan melakukan query ke database.

```
// src/models/userModel.js
const db = require('../config/database'); // Asumsikan ada koneksi
database

exports.findById = async (id) => {
  // ... logika query database untuk mencari user berdasarkan ID
  const result = await db.query('SELECT * FROM users WHERE id = $1',
[id]);
  return result.rows[0];
};

exports.update = async (id, userData) => {
  // ... logika query database untuk update user
};
```

---

## 📝 Praktik Mandiri (8 Jam)

1. **Buat Struktur Folder:** Buat folder `src/`, `src/models/`, `src/controllers/`, `src/routes/`, `src/middlewares/`, `src/config/` di project backend Anda.
2. **Pindahkan Kode:** Pindahkan kode yang sudah ada dari hari-hari sebelumnya ke folder yang sesuai:
   - Middleware otorisasi ke `src/middlewares/`.
   - Logika register dan login ke file controller baru (misalnya, `src/controllers/authController.js`).
   - Logika CRUD products, users, carts ke file controller yang sesuai (misalnya, `src/controllers/productController.js`, `src/controllers/userController.js`, `src/controllers/cartController.js`).
   - Buat file model untuk setiap resource (misalnya, `src/models/userModel.js`, `src/models/productModel.js`, `src/models/cartModel.js`) dan pindahkan logika interaksi database ke sana.
   - Buat file route untuk setiap resource (misalnya, `src/routes/authRoutes.js`, `src/routes/productRoutes.js`, dll.) dan definisikan endpoint di sana, panggil controller yang sesuai.
3. **Update File Utama (`app.js` atau `index.js`):** Impor dan gunakan router dari folder `src/routes/`.

```
// src/app.js
const express = require('express');
const app = express();
const authRoutes = require('./routes/authRoutes');
const productRoutes = require('./routes/productRoutes');
// ... import routes lain
```

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

app.use('/api/auth', authRoutes); // Gunakan router auth
app.use('/api/products', productRoutes); // Gunakan router products
// ... gunakan routes lain

// ... error handling middleware, server listen
```

4. **Uji Aplikasi:** Pastikan semua endpoint (register, login, CRUD) masih berfungsi setelah refactoring.

---

## 💡 Tips untuk Pemula

- Jangan terburu-buru memindahkan semua kode sekaligus. Lakukan secara bertahap per resource atau per fitur.
- Pastikan path impor (`require()`) sudah benar setelah memindahkan file.
- Gunakan nama file dan folder yang konsisten dan deskriptif.
- Struktur MVC adalah panduan, bukan aturan kaku. Sesuaikan dengan kebutuhan project Anda.

---

## 📚 Referensi

- MVC Pattern Explained
- Structuring Express Applications
- Node.js Project Structure Best Practices