Rakamin Academy

Application Structure

API architecture Design pattern introduction

Raih Mimpi #TanpaBatas

Outline Pembelajaran

Rakamin Academy

Application Structure

- [ ] Design pattern introduction
- [ ] MVC (Model View Controller)
- [ ] Monolith Architecture

Objektif sesi

- Memahami dan mengimplementasikan Design pattern
- Memahami dan mengimplementasikan MVC (Model View Controller)

Design Pattern

Solusi umum untuk masalah yang sering muncul dalam pengembangan perangkat lunak. Mereka memberikan panduan atau templat untuk memecahkan masalah desain yang umum, membantu pengembang menyusun kode mereka dengan cara yang terstruktur, terorganisir, dan mudah dimengerti.

Berikut adalah beberapa kategori pola desain yang umum digunakan untuk pembuatan Restful API menggunakan node js.

1. Singleton Pattern
2. Repository Pattern
3. Dependency Injection

Contoh: Singleton Pattern

sihngleton.js

```
// singleton.js
class Database {
    constructor() {
        if (Database.instance) {
            return Database.instance;
        }
        this.data = [];
        Database.instance = this;
    }

    getData() {
        return this.data;
    }

    addData(item) {
        this.data.push(item);
    }
}

module.exports = new Database();

```

server.js

```
// server.js
const express = require('express');
const database = require('./singleton');

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

app.get('/api/data', (req, res) => {
    res.json(database.getData());
});

app.post('/api/data', (req, res) => {
    const { item } = req.body;
    database.addData(item);
    res.status(201).send('Data added successfully');
});

app.listen(3000, () => {
    console.log('Server is running on port 3000');
});

```

Contoh: Repository Pattern

repository.js

```js
// repository.js
class UserRepository {
    constructor() {
        this.data = [];
    }

    getAll() {
        return this.data;
    }

    add(user) {
        this.data.push(user);
    }
}

module.exports = new UserRepository();

```

server.js

```js
// server.js
const express = require('express');
const userRepository = require('./repository');

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

app.get('/api/users', (req, res) => {
    const users = userRepository.getAll();
    res.json(users);
});

app.post('/api/users', (req, res) => {
    const { user } = req.body;
    userRepository.add(user);
    res.status(201).send('User added successfully');
});

app.listen(3000, () => {
    console.log('Server is running on port 3000');
});

```

Contoh: Dependency Injection

database.js

```js
// database.js
class Database {
    constructor() {
        this.data = [];
    }

    getData() {
        return this.data;
    }

    addData(item) {
        this.data.push(item);
    }
}

module.exports = Database;
```

repository.js

```js
// repository.js
class UserRepository {
    constructor(database) {
        this.database = database;
    }

    getAll() {
        return this.database.getData();
    }

    add(user) {
        this.database.addData(user);
    }
}

module.exports = UserRepository;
```

server.js

```js
// server.js
const express = require('express');
const Database = require('./database');
const UserRepository = require('./repository');

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

const database = new Database();
const userRepository = new UserRepository(database);

app.get('/api/users', (req, res) => {
    const users = userRepository.getAll();
    res.json(users);
});

app.post('/api/users', (req, res) => {
    const { user } = req.body;
    userRepository.add(user);
    res.status(201).send('User added successfully');
});

app.listen(3000, () => {
    console.log('Server is running on port 3000');
});
```

Outline Pembelajaran

Rakamin Academy

Application Structure

- [x] Design pattern introduction
- [ ] MVC (Model View Controller)
- [ ] Monolith Architecture

Architecture Pattern: MVC

MVC (Model-View-Controller) adalah sebuah pola desain arsitektur perangkat lunak yang digunakan untuk memisahkan logika aplikasi menjadi tiga komponen utama: Model, View, dan Controller. Setiap komponen memiliki tanggung jawab tertentu dalam aplikasi.

1. Model
2. View
3. Controller

MVC: Inisialisasi Proyek

➔ Buat direktori baru untuk proyek Anda dan inisialisasikan proyek Node.js dengan menggunakan npm

bash

```bash
mkdir node-mvc-mongodb
cd node-mvc-mongodb
npm init -y
```

MVC: Instalasi Modul

➔ Instal modul-modul yang dibutuhkan, termasuk ***express*** untuk server HTTP, ***mongoose*** untuk berinteraksi dengan MongoDB, serta ***body-parser*** untuk parsing body permintaan HTTP:

bash

```
npm install express mongoose body-parser
```

MVC: Struktur Proyek

➔ Buat struktur direktori untuk mengikuti pola MVC dan memisahkan kode berdasarkan fungsionalitas:

```
node-mvc-mongodb/
  |- controllers/
  |    |- mainController.js
  |- models/
  |    |- bookModel.js
  |- routes/
  |    |- mainRoutes.js
  |- app.js
```

MVC: Koneksi ke MongoDB

➔ Dalam app.js, tambahkan kode untuk menghubungkan aplikasi Node.js Anda dengan database MongoDB

app.js

```
// app.js

const express = require("express");
const bodyParser = require("body-parser");
const mongoose = require("mongoose");
const mainRoutes = require("./routes/mainRoutes");

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

// Middleware
app.use(bodyParser.json());

// MongoDB Connection
mongoose.connect("mongodb://localhost:27017/mydatabase", {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
const db = mongoose.connection;
db.on("error", console.error.bind(console, "MongoDB connection error:"));
db.once("open", function () {
  console.log("Connected to MongoDB");
});

// Routes
app.use("/api", mainRoutes);

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

```

MVC: Definisikan Model

➔ Dalam ***bookModel.js***, definisikan model untuk data buku yang akan disimpan di MongoDB

bookModel.js

```
1  // models/bookModel.js
2
3  const mongoose = require("mongoose");
4
5  const bookSchema = new mongoose.Schema({
6    title: String,
7    author: String,
8    publishedYear: Number,
9  });
10
11 const Book = mongoose.model("Book", bookSchema);
12
13 module.exports = Book;
14
```

MVC: Buat Controller

➔ Dalam ***bookModel.js***, definisikan model untuk data buku yang akan disimpan di MongoDB

mainController.js

```
// controllers/mainController.js

const Book = require("../models/bookModel");

class MainController {
  async getAllData(req, res) {
    try {
      const books = await Book.find();
      res.json(books);
    } catch (error) {
      res.status(500).json({ message: error.message });
    }
  }

  async getDataById(req, res) {
    const { id } = req.params;
    try {
      const book = await Book.findById(id);
      if (!book) {
        return res.status(404).json({ message: "Book not found" });
      }
      res.json(book);
    } catch (error) {
      res.status(500).json({ message: error.message });
    }
  }
}

module.exports = MainController;

```

MVC: Definisikan Route

➔ Dalam **mainRoutes.js**, definisikan rute untuk aplikasi Anda

mainRoutes.js

```
// routes/mainRoutes.js

const express = require("express");
const MainController = require("../controllers/mainController");

const router = express.Router();
const mainController = new MainController();

router.get("/data", mainController.getAllData.bind(mainController));
router.get("/data/:id", mainController.getDataById.bind(mainController));

module.exports = router;

```

Outline Pembelajaran

Rakamin Academy

Application Structure

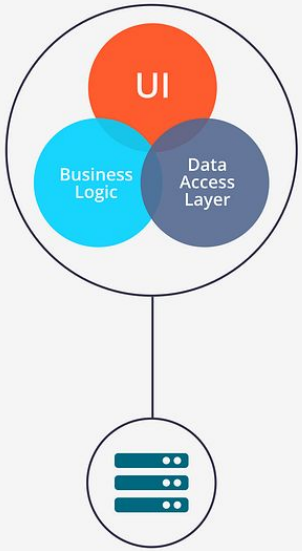
- [x] Design pattern introduction
- [x] MVC (Model View Controller)
- [ ] Monolith Architecture

Monolith Architecture

Monolith Architecture adalah pendekatan arsitektur perangkat lunak di mana seluruh aplikasi dibangun dan diimplementasikan sebagai satu entitas tunggal yang terintegrasi.

Dalam arsitektur monolit, komponen aplikasi seperti UI, logika bisnis, dan basis data ditempatkan dalam satu kode sumber dan dideploy sebagai satu unit.

Ini adalah pendekatan tradisional di mana seluruh aplikasi terkandung dalam satu "monolit" yang besar.



Monolithic Architecture

Monolith Architecture: Karakteristik

- **Tight Coupling**: Komponen-komponen aplikasi saling terkait erat, yang membuat perubahan di satu bagian dapat memengaruhi bagian lainnya.
- **Single Codebase**: Seluruh kode aplikasi terletak dalam satu proyek atau repositori.
- **Single Deployment Unit**: Aplikasi diimplementasikan dan dideploy sebagai satu kesatuan.
- **Scalability Challenges**: Keseluruhan aplikasi harus diperbesar secara keseluruhan, bahkan jika hanya beberapa bagian yang membutuhkan peningkatan kapasitas.

Outline Pembelajaran

Rakamin Academy

Application Structure

- Design pattern introduction
- MVC (Model View Controller)
- Monolith Architecture


Rakamin
Academy
Thank you
Raih Mimpi #TanpaBatas