Laporan Praktikum

MataKuliah

Pemrograman Web

Tugas Pertemuan 9

"REST API REPRESENTATIONAL STATE TRANSFER APPLICATION PROGRAMMING INTERFACE"

Dosen Pengampu:

Willdan Aprizal Arifin, S.Pd., M.Kom.

Disusun Oleh:

Habibirrohim

2300149

PROGRAM STUDI SISTEM INFORMASI KELAUTAN

UNIVERSITAS PENDIDIKAN INDONESIA

2024

I. PENDAHULUAN

REST API (Representational State Transfer Application Programming Interface) merupakan standar arsitektur komunikasi berbasis web yang sering diterapkan dalam pengembangan layanan berbasis web. Praktikum ini berfokus pada implementasi REST API menggunakan Node.js dan Express.js sebagai framework utama, dengan penambahan fitur autentikasi menggunakan JSON Web Token (JWT).

II. ALAT DAN BAHAN

2.1 Alat Dan Bahan

1. Node.js (versi terbaru)
2. Visual Studio Code atau IDE pilihan
3. MySQL Server
4. Postman (opsional, untuk pengujian API)
5. Web browser modern (Chrome, Firefox, Edge)

III. PENJELASAN

Pemahaman saya pada praktikum kali ini

Konsep Utama yang Akan Dipelajari

Dalam praktikum ini, kita akan mempelajari dan mengimplementasikan beberapa konsep penting dalam pengembangan REST API, termasuk:

1. Pembuatan endpoint API untuk operasi CRUD (Create, Read, Update, Delete)
2. Implementasi sistem autentikasi menggunakan JWT
3. Penanganan keamanan API melalui middleware
4. Manajemen koneksi database MySQL
5. Implementasi rate limiting untuk mengontrol akses API
6. Pencatatan aktivitas API melalui logging system

Tujuan dari praktikum ini adalah untuk memberikan pemahaman praktis tentang bagaimana membangun REST API yang aman dan terstruktur, serta menerapkan praktik-praktik terbaik dalam pengembangan web modern. Melalui praktikum ini, mahasiswa diharapkan dapat memahami konsep dasar REST API dan mampu mengimplementasikannya dalam pengembangan aplikasi web.

Hasil dari praktikum ini akan menghasilkan sebuah REST API yang memiliki fitur autentikasi, manajemen token, dan operasi CRUD terhadap database, yang dapat digunakan sebagai fondasi untuk pengembangan aplikasi web yang lebih kompleks.

1. Buka VSCode dan buat folder baru untuk proyek Anda.

2. Dalam folder tersebut, buka terminal dan jalankan perintah `npm init -y` untuk membuat file `package.json`.

3. Selanjutnya, install dependencies yang diperlukan dengan menjalankan perintah:

   ```
   npm install express jsonwebtoken
   ```

   - `express` adalah framework Node.js yang akan kita gunakan untuk membangun REST API.

   - `jsonwebtoken` adalah library untuk menghasilkan dan memverifikasi token JWT.

Sekarang, kita dapat mulai membuat kode untuk server REST API:

1. Dalam folder proyek, buat file baru bernama `app.js`.

2. Buka file `app.js` dan masukkan kode berikut:

```
const express = require("express");
const bodyParser = require("body-parser");
const koneksi = require("./config/Database.js");
const app = express();
const PORT = process.env.PORT || 3000;
const jwt = require("jsonwebtoken");
const rateLimit = require("express-rate-limit");
const secretKey = "fadli_secret_key";
const validTokens = new Set();
const revokedTokens = new Set();

// Set body parser
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

// Rate limiting middleware
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 menit
  max: 100 // batas 100 permintaan per IP
});

// Middleware untuk mencatat akses
const accessLogger = (req, res, next) => {
```

```
  console.log(`Akses ke ${req.originalUrl} oleh ${req.ip}`);
  next();
};

// Gunakan middleware
app.use(limiter);
app.use(accessLogger);

// Endpoint untuk login dan mendapatkan token
app.post("/api/login", (req, res) => {
  const { username, password } = req.body;

  // Validasi username dan password (ganti dengan logika autentikasi Anda)
  if (username === "admin" && password === "password") {
    const token = jwt.sign({ username }, secretKey, { expiresIn: "1h" });
    validTokens.add(token);
    return res.status(200).json({ token });
  }
  return res.status(401).json({ message: "Username atau password salah" });
});

// Middleware untuk memvalidasi token
const authenticateToken = (req, res, next) => {
  const token = req.headers["authorization"]?.split(" ")[1];
  if (!token || revokedTokens.has(token)) return res.sendStatus(403);

  jwt.verify(token, secretKey, (err, user) => {
    if (err) return res.sendStatus(403);
    req.user = user;
    next();
  });
};

// Endpoint untuk mengakses data yang dilindungi
app.get("/api/protected", authenticateToken, (req, res) => {
  res.status(200).json({ message: "Data yang dilindungi", user: req.user });
});

// Endpoint untuk merevoke token
app.post("/api/revoke", (req, res) => {
  const { token } = req.body;
  if (validTokens.has(token)) {
    validTokens.delete(token);
    revokedTokens.add(token);
    return res.status(200).json({ message: "Token berhasil direvoke" });
```

```
  }
  return res.status(400).json({ message: "Token tidak valid" });
});

// Insert data
app.post("/api/latihanrestapi", (req, res) => {
  const data = { ...req.body };
  const querySql = "INSERT INTO latihanrestapi SET ?";

  koneksi.query(querySql, data, (err) => {
    if (err) {
      return res.status(500).json({ message: "GAGAL Insert data!", error: err });
    }
    res.status(201).json({ success: true, message: "Berhasil insert data" });
  });
});

// Read data / get data
app.get("/api/latihanrestapi", (req, res) => {
  const querySql = "SELECT * FROM latihanrestapi";

  koneksi.query(querySql, (err, rows) => {
    if (err) {
      return res.status(500).json({ message: "Ada kesalahan", error: err });
    }
    res.status(200).json({ success: true, data: rows });
  });
});

// Update data
app.put("/api/latihanrestapi/:id", (req, res) => {
  const data = { ...req.body };
  const querySearch = "SELECT * FROM latihanrestapi WHERE id = ?";
  const queryUpdate = "UPDATE latihanrestapi SET ? WHERE id = ?";

  koneksi.query(querySearch, req.params.id, (err, rows) => {
    if (err) {
      return res.status(500).json({ message: "Ada kesalahan", error: err });
    }

    // Pastikan data ditemukan
    if (rows.length) {
      koneksi.query(queryUpdate, [data, req.params.id], (err) => {
        if (err) {
          return res.status(500).json({ message: "Ada kesalahan", error: err });
```

```
        }
        res.status(200).json({ success: true, message: "Berhasil update data!"
});
      });
    } else {
        return res
        .status(404)
        .json({ message: "Data tidak ditemukan!", success: false });
    }
  });
});

// Endpoint untuk menghapus data
app.delete("/api/latihanrestapi/:id", (req, res) => {
  const querySearch = "SELECT * FROM latihanrestapi WHERE id = ?";
  const queryDelete = "DELETE FROM latihanrestapi WHERE id = ?";

  // Jalankan query untuk mencari data
  koneksi.query(querySearch, req.params.id, (err, rows) => {
    // Error handling
    if (err) {
      return res.status(500).json({ message: "Ada kesalahan", error: err });
    }

    // Jika id yang dimasukkan sesuai dengan data yang ada di db
    if (rows.length) {
      // Jalankan query delete
      koneksi.query(queryDelete, req.params.id, (err) => {
        // Error handling
        if (err) {
          return res.status(500).json({ message: "Ada kesalahan", error: err });
        }
        // Jika delete berhasil
        res.status(200).json({ success: true, message: "Berhasil hapus data!" });
      });
    } else {
      return res
        .status(404)
        .json({ message: "Data tidak ditemukan!", success: false });
    }
  });
});

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

Alur penggunaan:

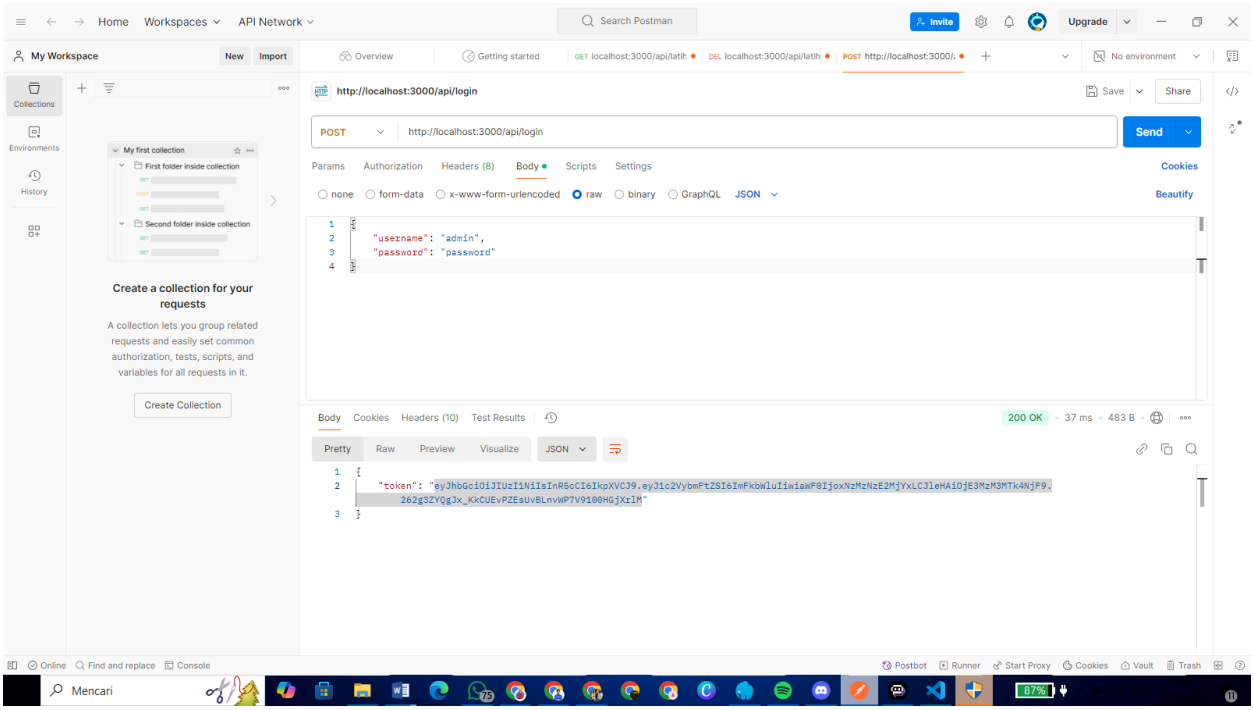
1. Memulai server: Jalankan perintah `node server.js` di terminal VSCode. Server akan berjalan di port 3000.

2. Login: Buka Postman dan buat sebuah POST request ke `http://localhost:3000/login`. Pada body request, masukkan `username` dan `password` (misalnya `user1` dan `password1`). Jika kredensial valid, server akan mengembalikan token JWT.

3. Mengakses endpoint protected: Buat sebuah GET request ke `http://localhost:3000/protected`. Pada header request, tambahkan `Authorization: Bearer <token>`, di mana `<token>` adalah token yang didapatkan dari langkah login. Jika token valid, server akan mengembalikan pesan "This is a protected route" dan mencatat aktivitas pengguna di console.

4. Mencoba mengakses tanpa token: Buat sebuah GET request ke `http://localhost:3000/protected` tanpa menambahkan header `Authorization`. Server akan mengembalikan respons 403 Forbidden dengan pesan "No token provided".

5. Mencoba mengakses dengan token yang tidak valid: Buat sebuah GET request ke `http://localhost:3000/protected` dengan menambahkan header `Authorization: Bearer <token_yang_tidak_valid>`. Server akan mengembalikan respons 403 Forbidden dengan pesan "Failed to authenticate token".



IV. KESIMPULAN

Dalam contoh ini, kita menggunakan JWT untuk mengautentikasi pengguna dan memberikan akses ke endpoint yang dilindungi. Setiap kali endpoint `/protected` diakses, aktivitas pengguna akan dicatat di

console. Anda dapat memperluas contoh ini dengan menambahkan fitur-fitur lain, seperti revoke token, rate limiting, dan manajemen hak akses yang lebih kompleks.