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:

Nelita Maharani

2307052

PROGRAM STUDI SISTEM INFORMASI KELAUTAN

UNIVERSITAS PENDIDIKAN INDONESIA

2024

I. PENDAHULUAN

REST API (Representational State Transfer Application Programming Interface) adalah standar arsitektur yang digunakan untuk membangun komunikasi berbasis web secara sederhana, fleksibel, dan efisien. Arsitektur ini sangat umum diterapkan dalam pengembangan layanan web modern karena sifatnya yang ringan dan mudah diintegrasikan. Dalam praktikum ini, kita akan mempelajari dan mengimplementasikan REST API dengan menggunakan **Node.js** sebagai runtime JavaScript yang cepat dan fleksibel, serta **Express.js** sebagai framework utama untuk pengelolaan rute dan middleware. Selain itu, akan ditambahkan fitur autentikasi berbasis **JSON Web Token (JWT)** untuk meningkatkan keamanan dengan memastikan setiap permintaan yang diterima oleh server telah terverifikasi.

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

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 utama dari praktikum ini adalah memberikan pemahaman yang mendalam dan keterampilan praktis dalam membangun REST API yang aman, terstruktur, dan sesuai dengan standar pengembangan web modern. Dengan pendekatan hands-on, mahasiswa akan diajak untuk memahami konsep fundamental REST API, mulai dari pengaturan rute hingga pengelolaan data melalui protokol HTTP. Selain itu, mahasiswa juga akan mempelajari cara mengintegrasikan fitur keamanan menggunakan **JSON Web Token (JWT)**, yang melibatkan proses autentikasi dan otorisasi untuk memastikan setiap akses ke API memiliki validitas dan hak akses yang sesuai.

Melalui praktikum ini, mahasiswa diharapkan dapat menguasai teknik implementasi REST API yang efisien, termasuk penerapan **praktik terbaik** (best practices) dalam pengelolaan sumber daya, struktur kode, dan keamanan aplikasi. Dengan begitu, mereka akan memiliki dasar yang kuat untuk mengembangkan aplikasi web yang skalabel dan berorientasi pada kebutuhan industri.

Hasil akhir dari praktikum ini adalah sebuah REST API fungsional yang mencakup beberapa fitur inti, seperti autentikasi pengguna, pengelolaan dan validasi token, serta kemampuan untuk melakukan operasi **CRUD** (Create, Read, Update, Delete) terhadap database. API yang dihasilkan akan dirancang untuk mudah diintegrasikan ke dalam aplikasi web atau mobile yang lebih kompleks, menjadikannya sebagai fondasi yang kokoh untuk pengembangan sistem berbasis web di masa depan.

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
```

```
    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:

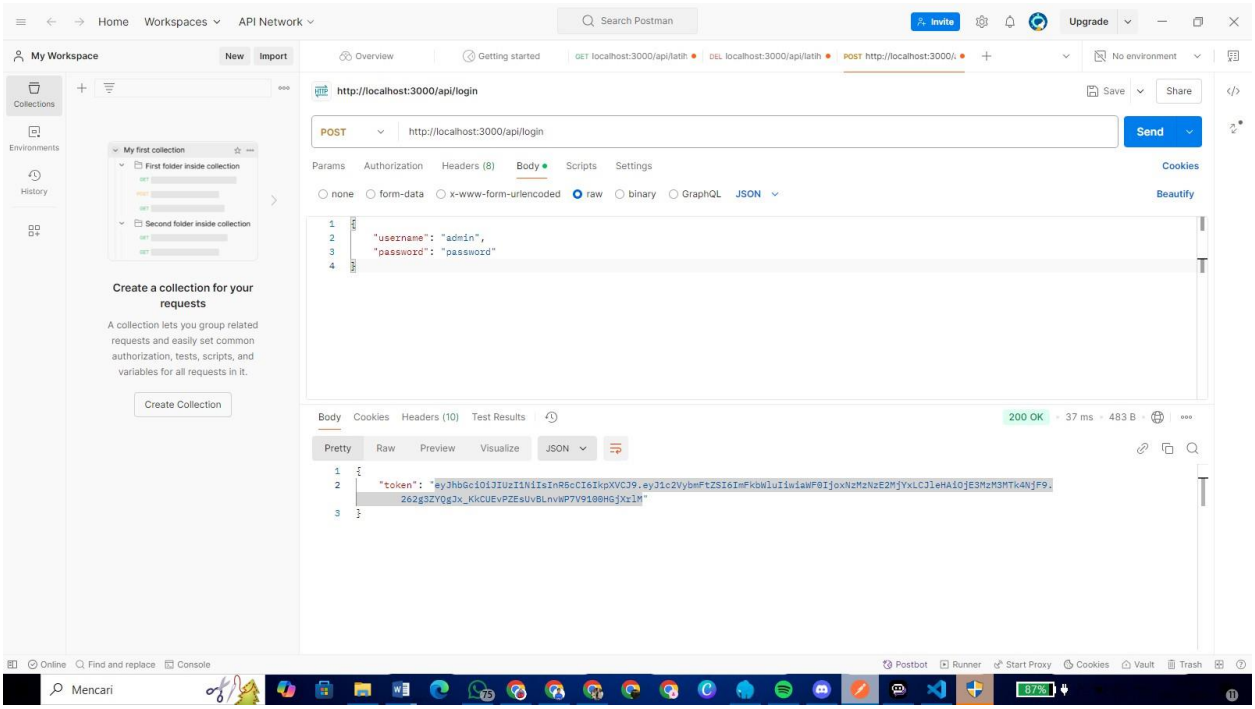
1. Memulai server: Jalankan perintah `node server.js` di terminal VSCode. Server akan berjalan di port 3000.

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, **JSON Web Token (JWT)** digunakan sebagai mekanisme autentikasi untuk memverifikasi identitas pengguna dan memberikan akses ke endpoint yang dilindungi. Ketika pengguna mencoba mengakses endpoint `/protected`, token yang dikirimkan akan divalidasi untuk memastikan keasliannya dan izin akses pengguna. Jika token tersebut valid, pengguna akan diberikan akses ke sumber daya yang diminta, sementara setiap aktivitas pengguna dicatat di **console log** sebagai bagian dari pemantauan sistem. Implementasi ini dapat dikembangkan lebih lanjut dengan menambahkan berbagai fitur keamanan dan efisiensi tambahan. Salah satunya adalah mekanisme **revoke token**, yang memungkinkan sistem mencabut token yang sudah tidak valid, seperti saat pengguna logout atau ketika terdeteksi potensi pelanggaran keamanan. Selain itu, fitur **rate limiting** dapat diintegrasikan untuk membatasi jumlah permintaan dari setiap pengguna atau alamat IP dalam periode waktu tertentu, guna melindungi sistem dari ancaman seperti **Denial of Service (DoS)**. Untuk meningkatkan pengelolaan izin, kontrol akses berbasis peran (**Role-Based Access Control/RBAC**) dapat diterapkan, sehingga setiap pengguna memiliki akses sesuai dengan peran mereka, seperti admin, moderator, atau pengguna biasa. Masa berlaku token juga bisa diatur dengan menambahkan **refresh token**, sehingga autentikasi dapat diperpanjang tanpa memaksa pengguna untuk login ulang. Selain itu, log audit yang terstruktur dapat digunakan untuk mencatat aktivitas pengguna secara rinci, yang berguna untuk analisis keamanan dan kepatuhan terhadap standar. Keamanan tambahan juga perlu diperhatikan, seperti menggunakan HTTPS untuk mengenkripsi komunikasi antara klien dan server, serta menambahkan proteksi ekstra menggunakan **HTTP Headers Security** melalui paket seperti `helmet` di Express.js. Dengan semua fitur ini, sistem tidak hanya menjadi lebih aman tetapi juga lebih andal dan siap untuk digunakan dalam pengembangan aplikasi berskala besar. Implementasi JWT yang fleksibel juga memungkinkan solusi ini diterapkan pada berbagai platform, baik web maupun mobile, menjadikannya pilihan yang efisien dan skalabel untuk memenuhi kebutuhan pengembangan modern.