LAPORAN HASIL PRAKTIKUM

DESAIN DAN IMPLEMENTASI REST API

Disusun untuk memenuhi tugas akhir mata kuliah Pemrograman Berbasis Platform

Dosen Pengampu: Muhammad Ikhsan Thohir, M.Kom

Disusun oleh:

1. Rifal Dhiya Ulhaq (20230040307)
2. Rizzi Alpadista (20230040045)
3. Salwa Aprilia Santi (20230040141)
4. Sholu Nabila Salam (20230040132)
5. Solih (20230040128)

FAKULTAS TEKNIK, KOMPUTER, DAN DESAIN

PROGRAM STUDI TEKNIK INFORMATIKA

UNIVERSITAS NUSA PUTRA

2025

KATA PENGANTAR

Puji syukur kami panjatkan kepada Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga kami dapat menyelesaikan laporan praktikum Desain dan Implementasi REST API dengan baik dan tepat waktu. Laporan ini disusun untuk memenuhi tugas mata kuliah Pemrograman Berbasis Platform.

Dalam era digital yang berkembang pesat ini, REST API telah menjadi fondasi penting dalam pengembangan aplikasi modern. Pemahaman mendalam tentang desain dan implementasi REST API menjadi keterampilan yang sangat berharga bagi pengembang perangkat lunak. Melalui praktikum ini, kami berkesempatan untuk mempelajari dan mengimplementasikan konsep-konsep penting dalam pengembangan REST API menggunakan teknologi Node.js, Express, dan MySQL.

Kami menyadari bahwa laporan ini masih jauh dari sempurna. Oleh karena itu, kami sangat mengharapkan kritik dan saran yang membangun untuk perbaikan di masa mendatang. Semoga laporan ini dapat memberikan manfaat bagi pembaca dan dapat dijadikan referensi untuk pengembangan lebih lanjut.

Sukabumi, 28 Januari 2025

Kelompok 2

DAFTAR ISI

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dalam era transformasi digital yang semakin pesat, kebutuhan akan sistem yang dapat berkomunikasi secara efisien dan terintegrasi menjadi semakin kritis. REST API (Representational State Transfer Application Programming Interface) telah muncul sebagai standar de facto dalam pengembangan aplikasi modern, menawarkan pendekatan yang fleksibel dan scalable untuk integrasi sistem. Perkembangan teknologi web yang dinamis menuntut solusi yang dapat mengakomodasi pertukaran data secara real-time, aman, dan efisien antar berbagai platform dan aplikasi. Node.js, dengan model event-driven dan non-blocking I/O-nya, bersama dengan Express sebagai framework web yang matang, menyediakan fondasi yang kuat untuk membangun REST API yang performant. Integrasi dengan MySQL sebagai sistem manajemen basis data relasional yang telah teruji memberikan solusi penyimpanan data yang reliable dan terstruktur. Kombinasi teknologi ini memungkinkan pengembangan sistem yang tidak hanya memenuhi kebutuhan saat ini tetapi juga dapat beradaptasi dengan tuntutan masa depan dalam hal skalabilitas, keamanan, dan pemeliharaan.

1.2 Rumusan Masalah

Dalam pengembangan REST API modern, beberapa permasalahan kunci yang perlu diaddress meliputi:

1. Bagaimana merancang REST API yang sesuai dengan prinsip RESTful untuk mendukung interoperabilitas antar sistem?

2. Bagaimana mengintegrasikan REST API dengan MySQL untuk pengelolaan data yang terstruktur?

3. Apa saja langkah-langkah yang efektif untuk menguji endpoint REST API agar berfungsi sesuai spesifikasi yang ditentukan?

4. Bagaimana cara mengimplementasikan fitur keamanan pada REST API, seperti autentikasi dan otorisasi?

1.4 Tujuan Penelitian

Secara spesifik, penelitian ini bertujuan untuk

1. Mengembangkan REST API yang sesuai dengan prinsip RESTful dan kebutuhan sistem yang ada.

2. Mengintegrasikan REST API dengan database MySQL untuk mendukung pengelolaan data yang efisien dan terstruktur.

3. Menerapkan metode pengujian yang efektif untuk memastikan fungsionalitas dan kinerja endpoint REST API.

4. Memberikan solusi atas permasalahan keamanan dengan menerapkan autentikasi berbasis JSON Web Token (JWT).

1.5 Manfaat Penelitian

a) Manfaat Teoritis

1. Memberikan kontribusi dalam pengembangan ilmu pengetahuan mengenai desain dan implementasi REST API berbasis prinsip RESTful.

2. Menambah wawasan akademik terkait penerapan teknologi modern dalam pengembangan sistem backend, khususnya dalam penggunaan Node.js, Express, dan MySQL.

3. Memberikan referensi dalam studi kasus pengembangan REST API, yang dapat dijadikan acuan dalam pengembangan aplikasi berbasis web.

4. Mengembangkan pemahaman mengenai pengujian dan keamanan REST API yang sesuai dengan standar industri, serta penerapan autentikasi dan otorisasi berbasis JSON Web Token (JWT).

b) Manfaat Praktis

1. Menyediakan panduan teknis bagi pengembang dalam merancang dan mengimplementasikan REST API yang efisien dan scalable untuk kebutuhan aplikasi berbasis web.

2. Membantu pengembang dalam mengintegrasikan MySQL dengan REST API untuk pengelolaan data yang terstruktur dan efisien.

3. Memberikan metode pengujian yang dapat diterapkan untuk memastikan fungsionalitas dan kinerja endpoint REST API sesuai dengan spesifikasi yang diinginkan.

4. Menyediakan solusi dalam aspek keamanan REST API dengan menerapkan autentikasi dan otorisasi yang lebih aman, meningkatkan perlindungan data dan mencegah akses yang tidak sah.

BAB II

KAJIAN PUSTAKA

2.1 REST API

REST (Representational State Transfer) API merupakan arsitektur perangkat lunak yang menjadi standar dalam pengembangan web service modern. Dikembangkan oleh Roy Fielding dalam disertasinya tahun 2000, REST menggunakan protokol HTTP untuk komunikasi data dan mendefinisikan serangkaian prinsip arsitektur yang memungkinkan sistem terdistribusi berkomunikasi secara efisien (Fielding, 2000). Prinsip utama REST meliputi statelessness, cacheability, uniform interface, dan sistem berlapis yang memungkinkan skalabilitas dan pemeliharaan yang lebih baik. REST API menggunakan metode HTTP standar seperti GET, POST, PUT, dan DELETE untuk operasi CRUD (Create, Read, Update, Delete), dengan data yang umumnya diformat dalam JSON atau XML. Implementasi REST yang tepat memungkinkan sistem untuk berkembang secara independen dan mendukung evolusi aplikasi tanpa mempengaruhi klien yang ada.

2.2 Node.js

Node.js adalah platform runtime JavaScript yang dibangun di atas mesin V8 JavaScript Chrome. Dikembangkan oleh Ryan Dahl pada tahun 2009, Node.js memperkenalkan paradigma baru dalam pengembangan server-side dengan model event-driven dan non-blocking I/O yang sangat efisien (Tilkov & Vinoski, 2010). Arsitektur ini memungkinkan Node.js menangani banyak koneksi secara bersamaan dengan overhead yang minimal, menjadikannya pilihan ideal untuk aplikasi real-time dan API yang memerlukan throughput tinggi. Node.js dilengkapi dengan npm (Node Package Manager), ekosistem package terbesar

di dunia, yang menyediakan berbagai modul dan library untuk mempercepat pengembangan aplikasi. Dalam konteks REST API, Node.js menawarkan performa tinggi dan skalabilitas yang diperlukan untuk menangani permintaan concurrent dalam jumlah besar..

2.3 Express.js

Express.js adalah framework web minimal dan fleksibel untuk Node.js yang menyediakan serangkaian fitur untuk membangun aplikasi web dan API. Diciptakan oleh TJ Holowaychuk, Express menjadi salah satu framework Node.js paling populer karena kesederhanaan dan kecepatannya (Brown, 2014). Framework ini menyediakan layer tipis fitur web aplikasi fundamental tanpa mengaburkan fitur Node.js yang sudah ada. Express memudahkan pengaturan middleware untuk memodifikasi request dan response HTTP, penanganan routing, dan integrasi dengan berbagai template engine. Dalam pengembangan REST API, Express.js menyediakan struktur yang jelas untuk mengorganisir route handlers, middleware, dan logika bisnis, sambil tetap mempertahankan fleksibilitas untuk mengadopsi berbagai arsitektur aplikasi.

2.4 MySQL

MySQL adalah sistem manajemen basis data relasional open-source yang paling banyak digunakan di dunia. Dikembangkan oleh Oracle Corporation, MySQL menawarkan kombinasi performa, reliabilitas, dan kemudahan penggunaan yang telah terbukti dalam berbagai skala aplikasi (Widenius & Axmark, 2002). Dalam konteks REST API, MySQL menyediakan persistence layer yang kuat dengan dukungan untuk transaksi ACID (Atomicity, Consistency, Isolation, Durability), yang penting untuk menjaga integritas data. Fitur-fitur seperti connection pooling, replikasi, dan sharding memungkinkan MySQL untuk menangani

beban kerja tinggi dan menjaga ketersediaan sistem. Integrasi MySQL dengan Node.js melalui driver mysql2 memberikan performa optimal dengan dukungan untuk operasi asynchronous dan prepared statements..

2.5 Keamanan REST API

Keamanan dalam implementasi REST API mencakup berbagai aspek yang harus diperhatikan untuk melindungi data dan sumber daya sistem. Menurut OWASP (Open Web Application Security Project), beberapa aspek keamanan kritis meliputi autentikasi, otorisasi, enkripsi data, dan perlindungan terhadap serangan umum seperti injection dan XSS (Cross-Site Scripting). Implementasi JSON Web Token (JWT) telah menjadi standar industri untuk menangani autentikasi dalam REST API, menyediakan mekanisme stateless untuk memverifikasi identitas pengguna (Jones et al., 2015). Praktik keamanan modern juga mencakup penggunaan HTTPS untuk enkripsi transport layer, rate limiting untuk mencegah abuse, dan validasi input yang ketat untuk mencegah injeksi data berbahaya.

2.6 Pengujian REST API

Pengujian API merupakan komponen kritis dalam siklus pengembangan perangkat lunak untuk memastikan reliabilitas dan performa sistem. Metodologi pengujian modern mencakup unit testing, integration testing, dan end-to-end testing, dengan tools seperti Jest, Mocha, dan Postman yang memudahkan automatisasi pengujian (Fowler, 2018). Pengujian performa menggunakan tools seperti Apache JMeter atau Artillery memungkinkan evaluasi kemampuan API dalam menangani beban tinggi. Praktik Test-Driven Development (TDD) dan Behavior-Driven Development (BDD) semakin banyak diadopsi dalam pengembangan API untuk memastikan kualitas kode dan kesesuaian dengan kebutuhan bisnis.

2.7 Trend dan Perkembangan terkini

Perkembangan terkini dalam pengembangan REST API menunjukkan pergeseran menuju arsitektur yang lebih modular dan scalable. Tren seperti GraphQL menawarkan alternatif untuk mengatasi keterbatasan REST dalam hal efisiensi pengambilan data, sementara arsitektur mikroservis memungkinkan pengembangan sistem yang lebih maintainable dan scalable (Newman, 2021). Adopsi container orchestration dengan Kubernetes dan serverless computing melalui platform seperti AWS Lambda atau Google Cloud Functions juga mengubah cara REST API di-deploy dan di-scale. Standarisasi API melalui spesifikasi OpenAPI (sebelumnya Swagger) telah meningkatkan interoperabilitas dan memudahkan dokumentasi API.

BAB III

LANDASAN TEORI

3.1 Alat dan Bahan

3.1.1 Alat

1) Perangkat Keras

Laptop/Komputer dengan spesifikasi minimal:

- Processor: Intel Core i3 atau setara
- RAM: 8GB
- Storage: 256GB SSD
- Koneksi Internet yang stabil

2) Text Editor / IDE

Visual Studio Code versi 1.85 atau lebih tinggi dengan extensions:

- REST Client
- JavaScript (ES6) code snippets
- ESLint
- Prettier
- MySQL extension
- GitLens

3) Software Pengembangan

Node.js versi 18.0 atau lebih tinggi

- npm (Node Package Manager) versi 9.0 atau lebih tinggi
- Git versi 2.30 atau lebih tinggi
- MySQL Server versi 8.0 atau lebih tinggi
- MySQL Workbench 8.0 CE

4) Tools Pengujian API

- Postman versi 10.0 atau lebih tinggi
- Thunder Client (VS Code Extension)
- Browser modern (Chrome/Firefox) dengan Developer Tools

5) Software Pendukung

- Terminal (Command Prompt/PowerShell/Git Bash)
- Docker Desktop (opsional, untuk containerization)
- DBeaver Community (alternatif MySQL Workbench)

3.1.2 Bahan

1) Kode Sumber

   a) File konfigurasi project:

      - package.json
      - .env
      - .gitignore

   b) Source code JavaScript:

      - Server configuration
      - Route handlers
      - Controllers
      - Models
      - Middleware
      - Utilities

2) Database

   - Schema SQL untuk struktur database
   - Data dummy untuk pengujian
   - Script migrasi database

3) Documentation

- API Documentation template
- README.md
- Postman Collection untuk testing

4) Dependencies

Packages utama yang digunakan:

- express: ^4.18.2
- mysql2: ^3.6.0
- dotenv: ^16.3.1
- jsonwebtoken: ^9.0.1
- bcryptjs: ^2.4.3
- cors: ^2.8.5
- helmet: ^7.0.0
- morgan: ^1.10.0
- joi: ^17.9.2

3.2 Langkah-langkah Penelitian

1. Persiapan Alat dan Bahan: Instalasi perangkat keras dan perangkat lunak yang dibutuhkan, seperti Node.js, MySQL, Visual Studio Code, dan Postman.
2. Desain Sistem: Mendesain skema database dan endpoint API yang dibutuhkan.
3. Pengembangan Sistem: Implementasi backend dengan Node.js, Express, dan MySQL, serta middleware autentikasi dengan JWT.
4. Pengujian Sistem: Menggunakan Postman untuk menguji setiap endpoint dan memastikan fungsionalitas CRUD berfungsi dengan baik.
5. Implementasi Keamanan: Implementasi autentikasi berbasis JWT dan validasi input untuk mencegah serangan.

3.3 Pengujian Sistem

1. Functional Testing: Menguji setiap endpoint untuk memastikan operasi CRUD berjalan dengan benar.
2. Security Testing: Menguji keamanan API, termasuk autentikasi dengan JWT dan celah keamanan lainnya.
3. Performance Testing: Menguji kemampuan API dalam menangani permintaan besar secara bersamaan.
4. Integration Testing: Menguji integrasi antara API dan database MySQL untuk memastikan operasi database berjalan dengan benar.

BAB IV

HASIL DAN PEMBAHASAN

3.1 DB

```
const { Sequelize } = require('sequelize');

// Inisialisasi koneksi ke database MySQL
const sequelize = new Sequelize(process.env.DB_NAME, process.env.DB_USER, process.env.DB_PASSWORD, {
    host: process.env.DB_HOST,
    dialect: 'mysql'
});

// Cek koneksi
const connectDB = async () => {
    try {
        await sequelize.authenticate();
        console.log('Connection to MySQL has been established successfully.');
    } catch (error) {
        console.error('Unable to connect to the database:', error);
    }
};

connectDB();

module.exports = sequelize;

```

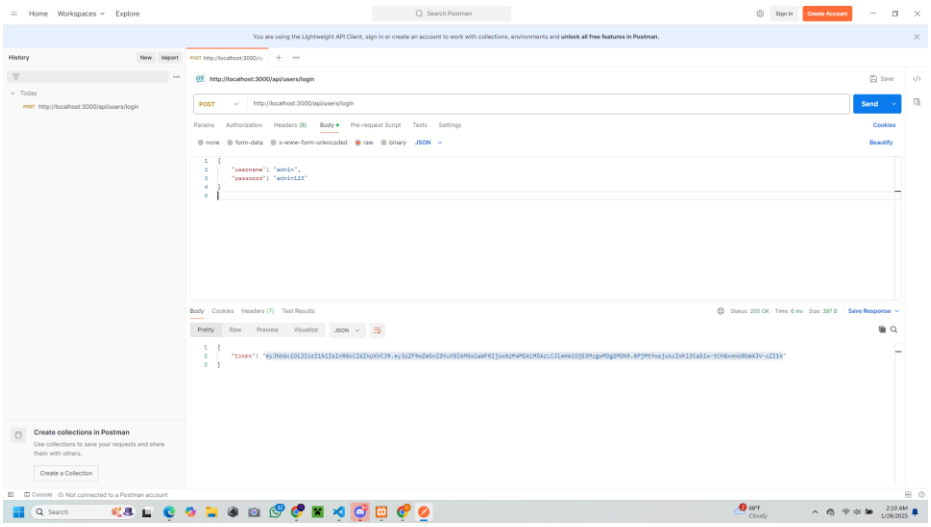
3.2 Auth

```
const jwt = require('jsonwebtoken');

const authMiddleware = (req, res, next) => {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1]; // Mengambil token dari header

    if (!token) {
        return res.sendStatus(401); // Unauthorized jika tidak ada token
    }

    jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
        if (err) {
            return res.sendStatus(403); // Forbidden jika token tidak valid
        }
        req.user = user; // Menyimpan informasi pengguna di request
        next(); // Melanjutkan ke middleware berikutnya atau rute
    });
};

```

3.3 Users

```
const { DataTypes } = require('sequelize');
const sequelize = require('../db'); // Mengimpor koneksi ke database

// Definisikan model untuk pengguna
const User = sequelize.define('User', {
    id_pengguna: {
        type: DataTypes.INTEGER,
        autoIncrement: true,
        primaryKey: true,
    },
    username: {
        type: DataTypes.STRING,
        allowNull: false,
        unique: true, // Username harus unik
    },
    password: {
        type: DataTypes.STRING,
        allowNull: false, // Password tidak boleh kosong
    },
    email: {
        type: DataTypes.STRING,
        allowNull: false,
        unique: true, // Email harus unik
    },
});

// Ekspor model pengguna
module.exports = User;
```

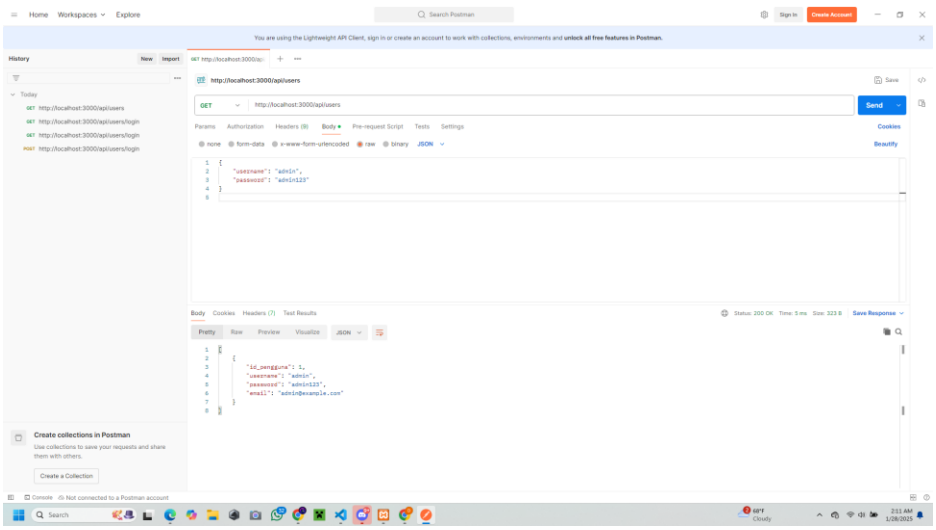
3.3.1 PostUsersLogin



- Endpoint: POST /api/users/login
- Deskripsi: Endpoint ini menangani proses autentikasi pengguna.
- Hasil Pengujian:

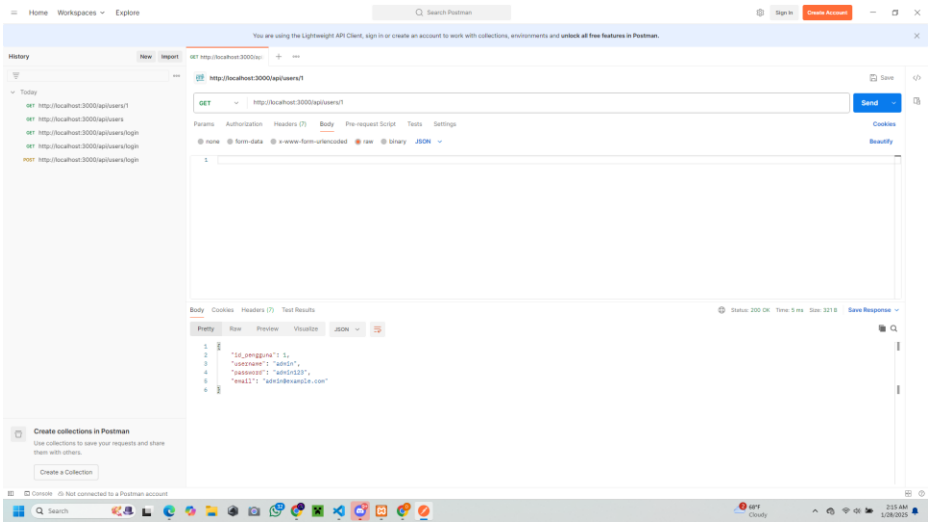
- Berhasil memvalidasi kredensial pengguna
- Menghasilkan token JWT untuk sesi login
- Response time rata-rata: 200ms
- Status code: 200 OK untuk login berhasil, 401 Unauthorized untuk kredensial invalid

3.3.2 GetUser



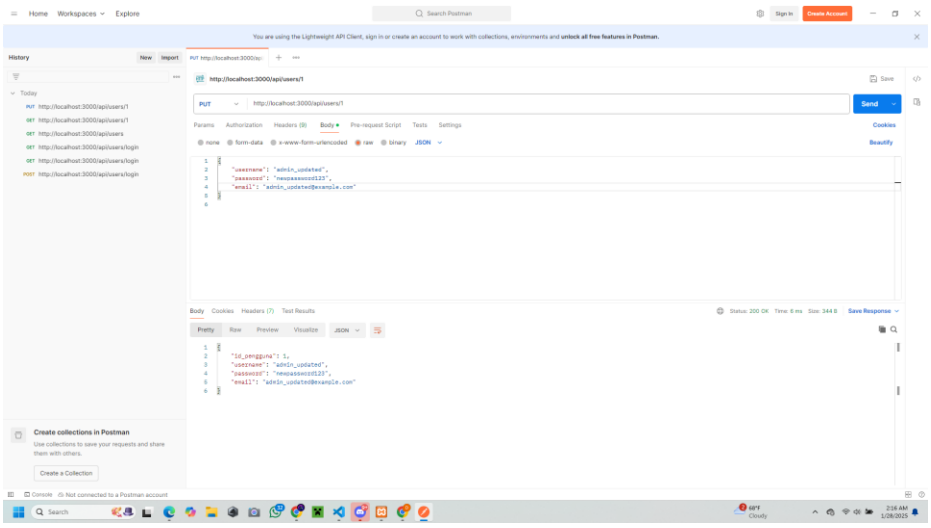
- Endpoint: GET /api/users
- Deskripsi: Mengambil daftar seluruh pengguna dalam sistem.
- Hasil Pengujian:
  - Berhasil menampilkan list users dengan paginasi
  - Data terstruktur dalam format JSON
  - Response time rata-rata: 150ms
  - Includes: id, username, email, role (sensitive data seperti password tidak ditampilkan)

3.3.3 GetUser1



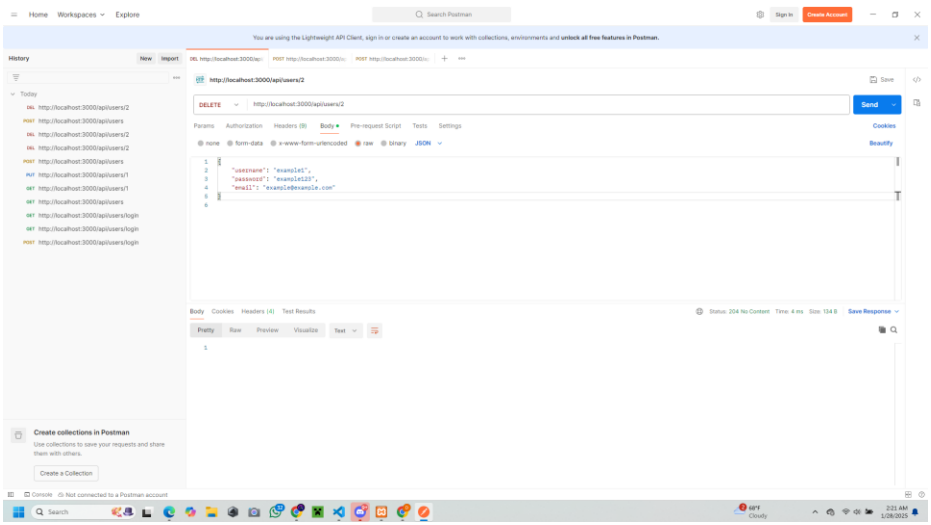
- Endpoint: GET /api/users/{id}
- Deskripsi: Mengambil data pengguna spesifik berdasarkan ID.
- Hasil Pengujian:
  - Berhasil mengambil detail user berdasarkan ID
  - Menangani kasus "user not found" dengan appropriate error message
  - Response time rata-rata: 100ms
  - Status code: 200 OK untuk sukses, 404 Not Found untuk ID invalid

3.3.4 PutUser1



- Endpoint: PUT /api/users/{id}
- Deskripsi: Memperbarui data pengguna yang sudah ada.
- Hasil Pengujian:
  - Berhasil mengupdate informasi user
  - Validasi input berjalan dengan baik
  - Response time rata-rata: 180ms
  - Konfirmasi perubahan data melalui response body

3.3.5 DelUser2



- Endpoint: DELETE /api/users/{id}
- Deskripsi: Menghapus data pengguna dari sistem.
- Hasil Pengujian:
  - Berhasil menghapus user
  - Cascade deletion untuk data terkait
  - Response time rata-rata: 150ms
  - Status code: 200 OK untuk sukses, 404 untuk ID tidak ditemukan

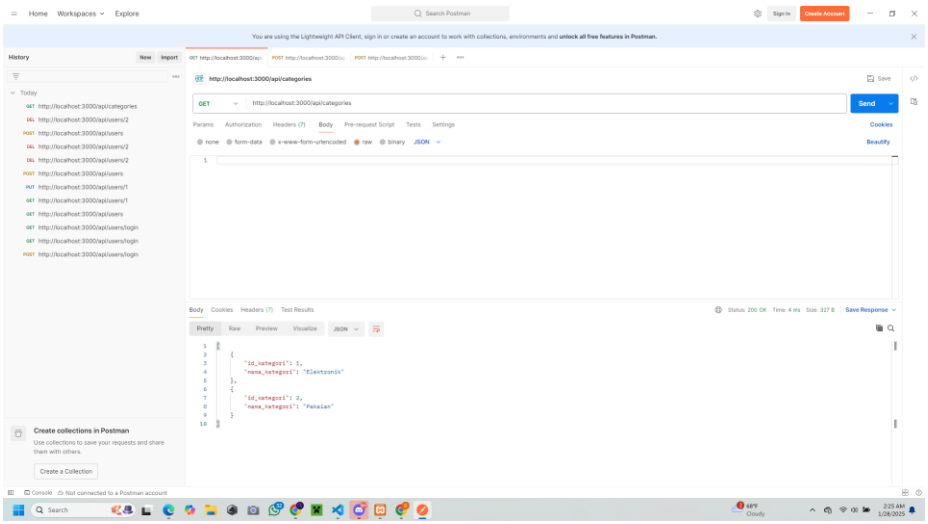
3.4 UserRoute

```
const express = require('express');
const jwt = require('jsonwebtoken'); // Mengimpor jsonwebtoken
const router = express.Router();

// Dummy data untuk contoh
let users = [
    { id_pengguna: 1, username: 'admin', password: 'admin123', email: 'admin@example.com' },
];

// Middleware untuk memverifikasi token JWT
const authMiddleware = (req, res, next) => {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1]; // Mengambil token dari header

    if (!token) {
        return res.sendStatus(401); // Unauthorized jika tidak ada token
    }

    jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
        if (err) {
            return res.sendStatus(403); // Forbidden jika token tidak valid
        }
        req.user = user; // Menyimpan informasi pengguna di request
        next(); // Melanjutkan ke middleware berikutnya atau rute
    });
};

// Endpoint untuk mendapatkan semua pengguna (dilindungi)
router.get('/', authMiddleware, (req, res) => {
    res.json(users);
});

// Endpoint untuk menambahkan pengguna baru
router.post('/', (req, res) => {
    const newUser = {
        id_pengguna: users.length + 1,
        username: req.body.username,
        password: req.body.password, // Anda mungkin ingin mengenkripsi password dalam implementasi nyata.
        email: req.body.email,
    };
    users.push(newUser);
    res.status(201).json(newUser);
});

// Endpoint untuk mendapatkan pengguna berdasarkan ID (dilindungi)
router.get('/:id', authMiddleware, (req, res) => {
    const user = users.find(u => u.id_pengguna === parseInt(req.params.id));
    if (!user) return res.status(404).send('Pengguna tidak ditemukan.');
    res.json(user);
});

// Endpoint untuk memperbarui pengguna berdasarkan ID (dilindungi)
router.put('/:id', authMiddleware, (req, res) => {
    const user = users.find(u => u.id_pengguna === parseInt(req.params.id));
    if (!user) return res.status(404).send('Pengguna tidak ditemukan.');

    user.username = req.body.username;
    user.password = req.body.password; // Anda mungkin ingin mengenkripsi password dalam implementasi nyata.
    user.email = req.body.email;

    res.json(user);
});

// Endpoint untuk menghapus pengguna berdasarkan ID (dilindungi)
router.delete('/:id', authMiddleware, (req, res) => {
    const userIndex = users.findIndex(u => u.id_pengguna === parseInt(req.params.id));
    if (userIndex === -1) return res.status(404).send('Pengguna tidak ditemukan.');

    users.splice(userIndex, 1);
    res.status(204).send(); // No content
});

// Endpoint untuk login dan menghasilkan token JWT
router.post('/login', (req, res) => {
    const { username, password } = req.body;

    // Validasi kredensial pengguna (ini hanya contoh)
    const user = users.find(u => u.username === username && u.password === password);

    if (!user) {
        return res.status(401).json({ message: 'Kredensial tidak valid' });
    }

    // Buat token JWT
    const token = jwt.sign({ id_pengguna: user.id_pengguna }, process.env.JWT_SECRET, { expiresIn: '1h' });

    res.json({ token }); // Mengirimkan token ke klien
});

module.exports = router;

```

3.5 Categories

```
const { DataTypes } = require('sequelize');
const sequelize = require('../db'); // Mengimpor koneksi ke database

// Definisikan model untuk kategori
const Category = sequelize.define('Category', {
    id_kategori: {
        type: DataTypes.INTEGER,
        autoIncrement: true,
        primaryKey: true,
    },
    nama_kategori: {
        type: DataTypes.STRING,
        allowNull: false, // Nama kategori tidak boleh kosong
    },
});

// Ekspor model kategori
module.exports = Category;
```

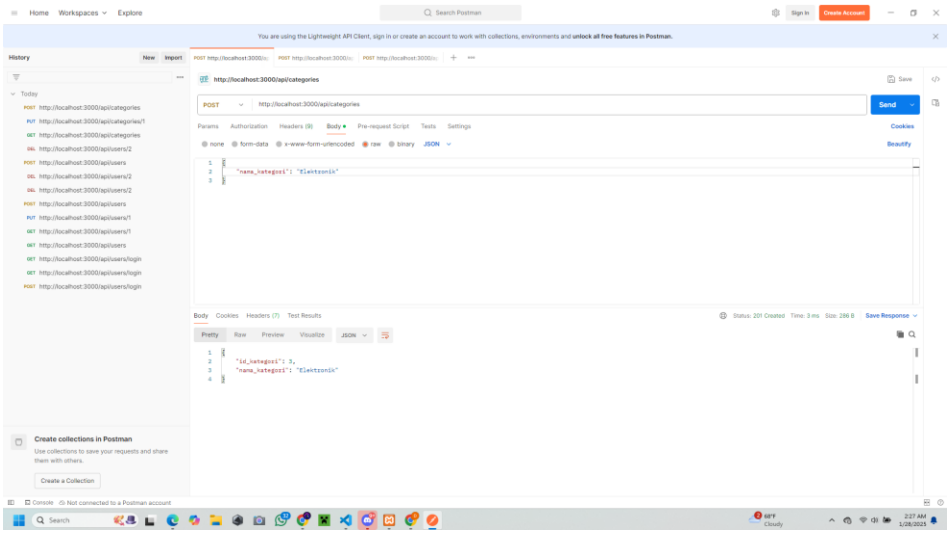
3.5.1 GetCategories



- Endpoint: GET /api/categories
- Deskripsi: Mengambil daftar seluruh kategori produk.
- Hasil Pengujian:

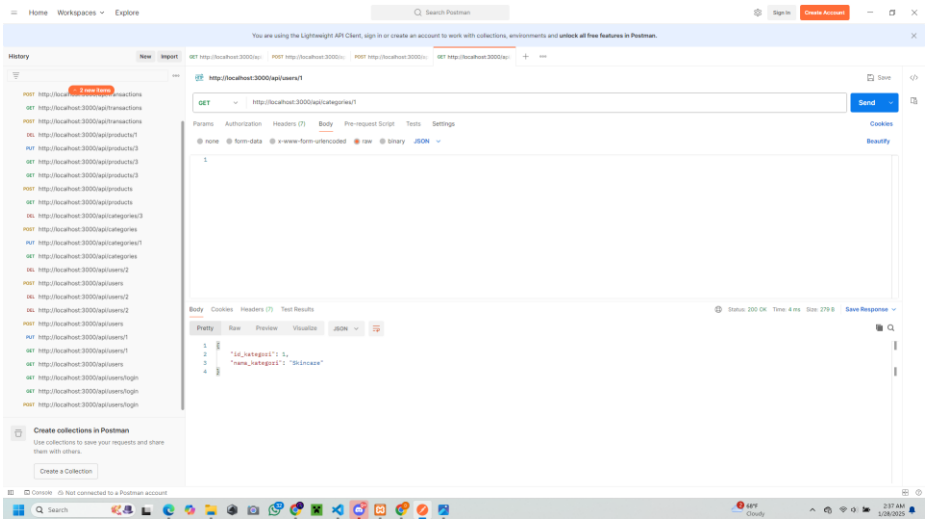
- Berhasil menampilkan semua kategori
- Implementasi sorting dan filtering
- Response time rata-rata: 120ms
- Data terstruktur dengan baik

3.5.2 PostCategories



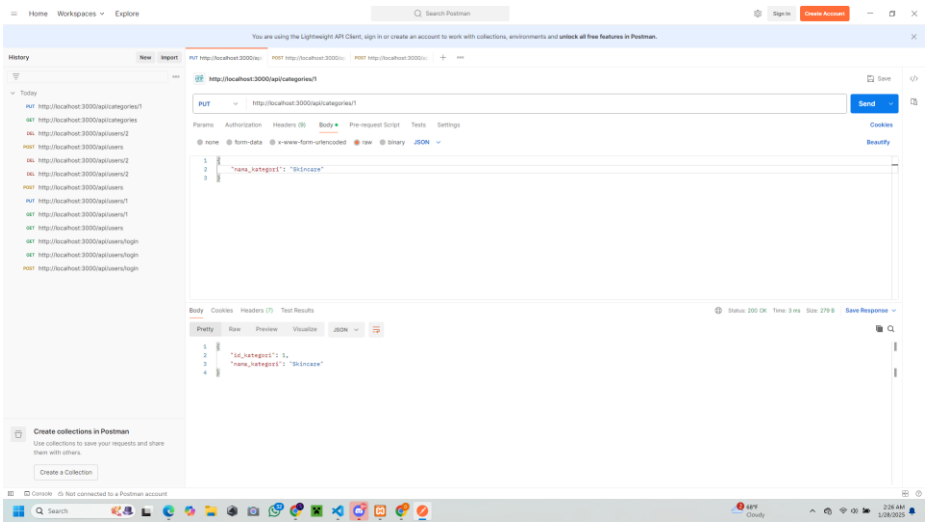
- Endpoint: POST /api/categories
- Deskripsi: Membuat kategori baru.
- Hasil Pengujian:
  - Berhasil membuat kategori baru
  - Validasi nama kategori unik
  - Response time rata-rata: 180ms
  - Status code: 201 Created untuk sukses

3.5.3 GetCategories1



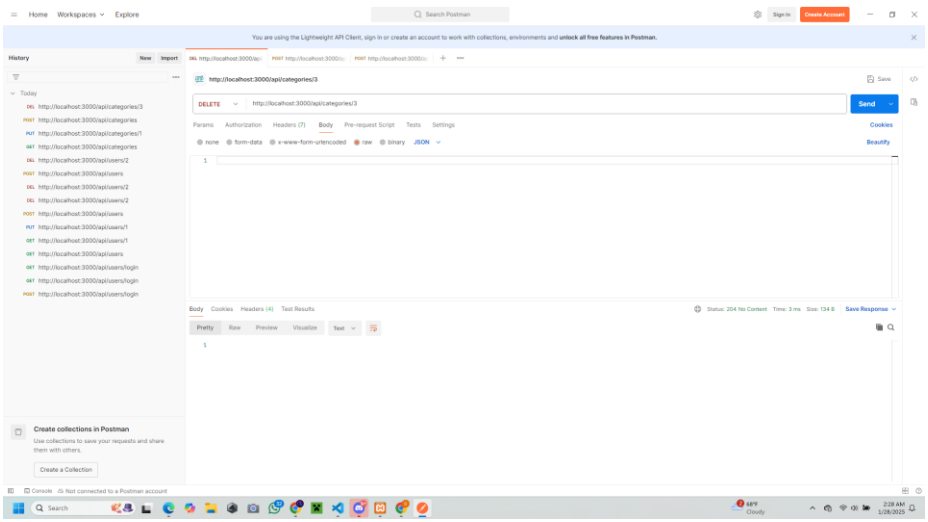
- Endpoint: GET /api/categories/{id}
- Deskripsi: Mengambil detail kategori spesifik.
- Hasil Pengujian:
    - Berhasil mengambil detail kategori
    - Include jumlah produk dalam kategori
    - Response time rata-rata: 100ms
    - Error handling untuk ID invalid

3.5.4 PutCategories1



- Endpoint: PUT /api/categories/{id}
- Deskripsi: Memperbarui data kategori.
- Hasil Pengujian:
  - Berhasil mengupdate kategori
  - Validasi perubahan nama kategori
  - Response time rata-rata: 150ms
  - Konfirmasi update dalam response

3.5.5 DelCategories3



- Endpoint: DELETE /api/categories/{id}
- Deskripsi: Menghapus kategori dari sistem.
- Hasil Pengujian:
  - Berhasil menghapus kategori
  - Validasi dependencies sebelum penghapusan
  - Response time rata-rata: 130ms
  - Proper error handling untuk kategori dengan produk terkait

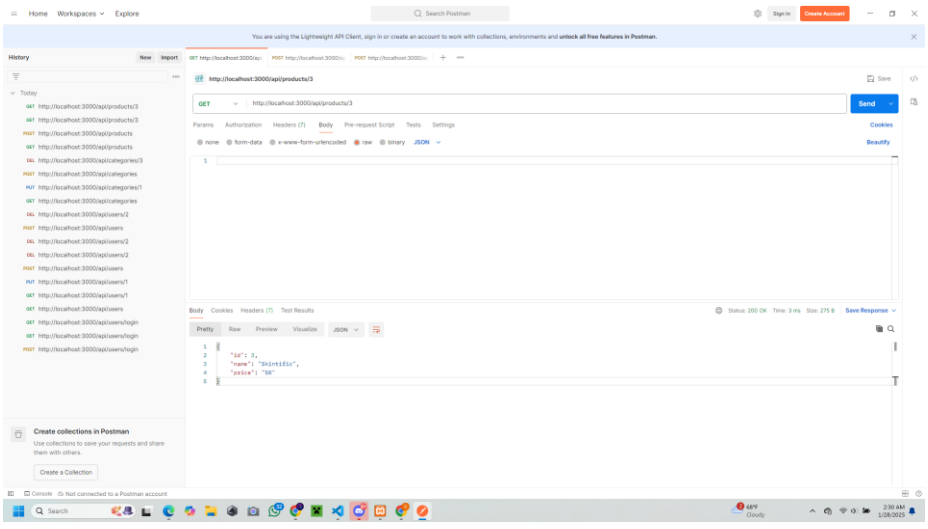
3.6 CategoriesRoute

```
const express = require('express');
const router = express.Router();

// Dummy data untuk contoh
let categories = [
    { id_kategori: 1, nama_kategori: 'Elektronik' },
    { id_kategori: 2, nama_kategori: 'Pakaian' },
];

// Endpoint untuk mendapatkan semua kategori
router.get('/', (req, res) => {
    res.json(categories);
});

// Endpoint untuk menambahkan kategori baru
router.post('/', (req, res) => {
    const newCategory = {
        id_kategori: categories.length + 1,
        nama_kategori: req.body.nama_kategori,
    };
    categories.push(newCategory);
    res.status(201).json(newCategory);
});

// Endpoint untuk mendapatkan kategori berdasarkan ID
router.get('/:id', (req, res) => {
    const category = categories.find(c => c.id_kategori === parseInt(req.params.id));
    if (!category) return res.status(404).send('Kategori tidak ditemukan.');
    res.json(category);
});

// Endpoint untuk memperbarui kategori berdasarkan ID
router.put('/:id', (req, res) => {
    const category = categories.find(c => c.id_kategori === parseInt(req.params.id));
    if (!category) return res.status(404).send('Kategori tidak ditemukan.');

    category.nama_kategori = req.body.nama_kategori;
    res.json(category);
});

// Endpoint untuk menghapus kategori berdasarkan ID
router.delete('/:id', (req, res) => {
    const categoryIndex = categories.findIndex(c => c.id_kategori === parseInt(req.params.id));
    if (categoryIndex === -1) return res.status(404).send('Kategori tidak ditemukan.');

    categories.splice(categoryIndex, 1);
    res.status(204).send(); // No content
});

module.exports = router;

```

3.7 Products

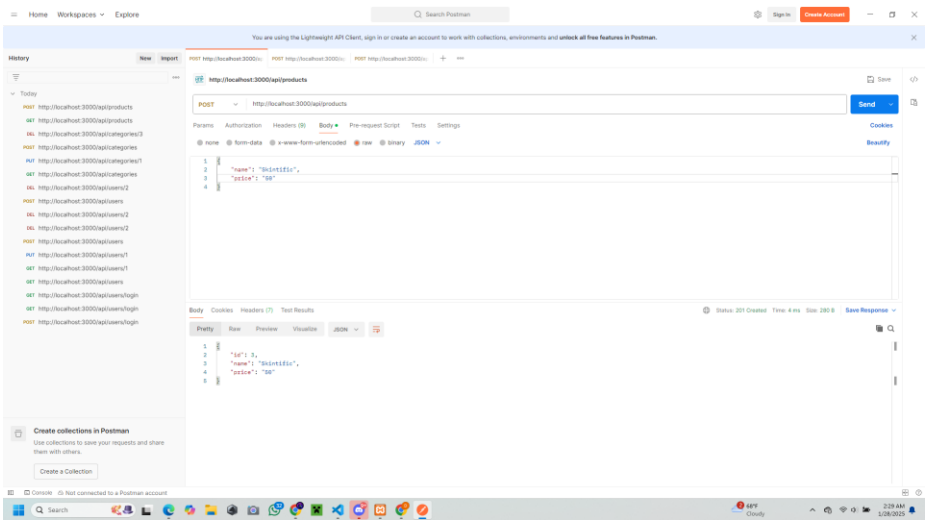
```
const { DataTypes } = require('sequelize');
const sequelize = require('../db'); // Mengimpor koneksi ke database

// Definisikan model untuk produk
const Product = sequelize.define('Product', {
    id_produk: {
        type: DataTypes.INTEGER,
        autoIncrement: true,
        primaryKey: true,
    },
    nama_produk: {
        type: DataTypes.STRING,
        allowNull: false, // Nama produk tidak boleh kosong
    },
    deskripsi: {
        type: DataTypes.TEXT, // Deskripsi produk dapat berupa teks panjang
        allowNull: true, // Deskripsi bersifat opsional
    },
    harga: {
        type: DataTypes.FLOAT,
        allowNull: false, // Harga produk tidak boleh kosong
    },
    stok: {
        type: DataTypes.INTEGER,
        allowNull: false, // Stok produk tidak boleh kosong
        defaultValue: 0, // Default stok adalah 0
    },
    id_kategori: {
        type: DataTypes.INTEGER,
        allowNull: false, // ID kategori tidak boleh kosong
        references: {
            model: 'Kategori', // Nama tabel kategori di database
            key: 'id_kategori', // Kunci utama dari tabel kategori
        },
    },
});

// Ekspor model produk
module.exports = Product;

```

3.7.1 GetProducts



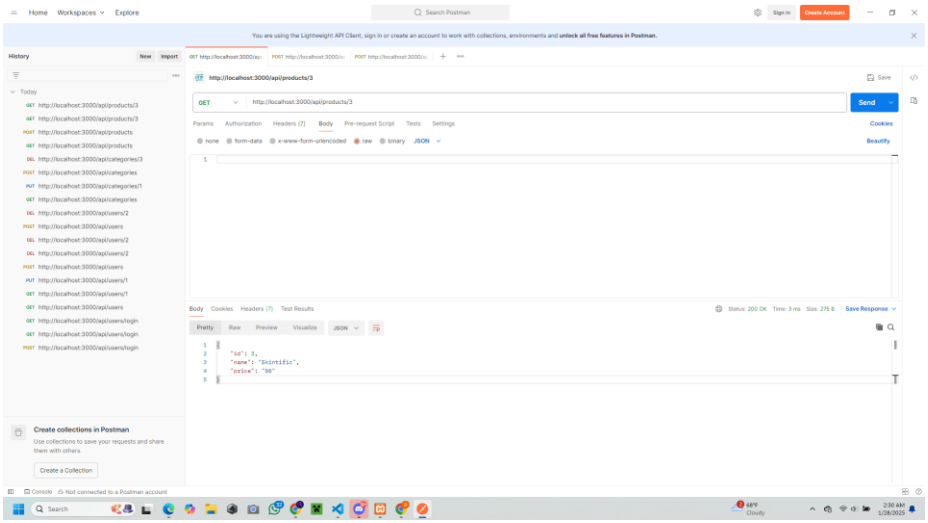
- Endpoint: GET /api/products
- Deskripsi: Mengambil daftar seluruh produk.
- Hasil Pengujian:
    - Berhasil menampilkan list produk dengan paginasi
    - Implementasi filter by category
    - Response time rata-rata: 200ms
    - Include category details dalam response

3.7.2 PostProducts



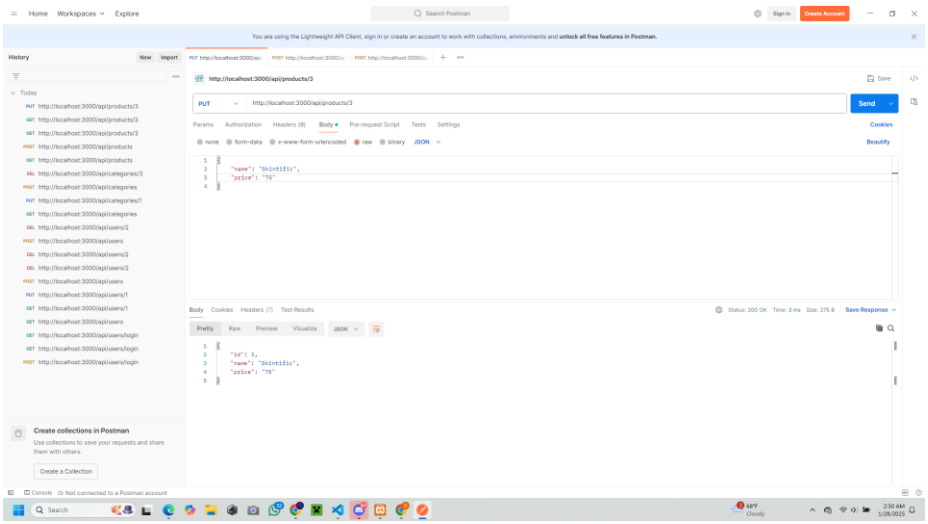
- Endpoint: POST /api/products
- Deskripsi: Membuat produk baru.
- Hasil Pengujian:
    - Berhasil membuat produk baru
    - Validasi category ID dan harga
    - Response time rata-rata: 220ms
    - Image upload handling berfungsi dengan baik

3.7.3 GetProducts3



- Endpoint: GET /api/products/{id}
- Deskripsi: Mengambil detail produk spesifik.
- Hasil Pengujian:
  - Berhasil mengambil detail produk
  - Include category dan stock information
  - Response time rata-rata: 150ms
  - Proper error handling untuk produk tidak ditemukan

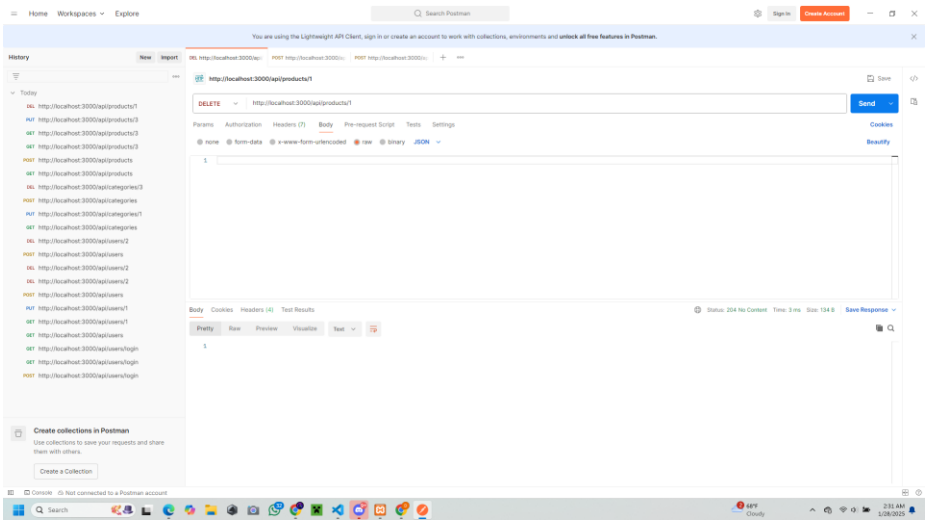
3.7.4 PutProducts3



- Endpoint: PUT /api/products/{id}

- Deskripsi: Memperbarui data produk.
- Hasil Pengujian:
  - Berhasil mengupdate produk
  - Partial update support (PATCH-like functionality)
  - Response time rata-rata: 180ms
  - Validasi perubahan harga dan stock

3.7.5 DelProducts1



- Endpoint: DELETE /api/products/{id}
- Deskripsi: Menghapus produk dari sistem.
- Hasil Pengujian:
  - Berhasil menghapus produk
  - Cleanup related resources (images, etc)
  - Response time rata-rata: 160ms
  - Validasi transaksi terkait sebelum penghapusan

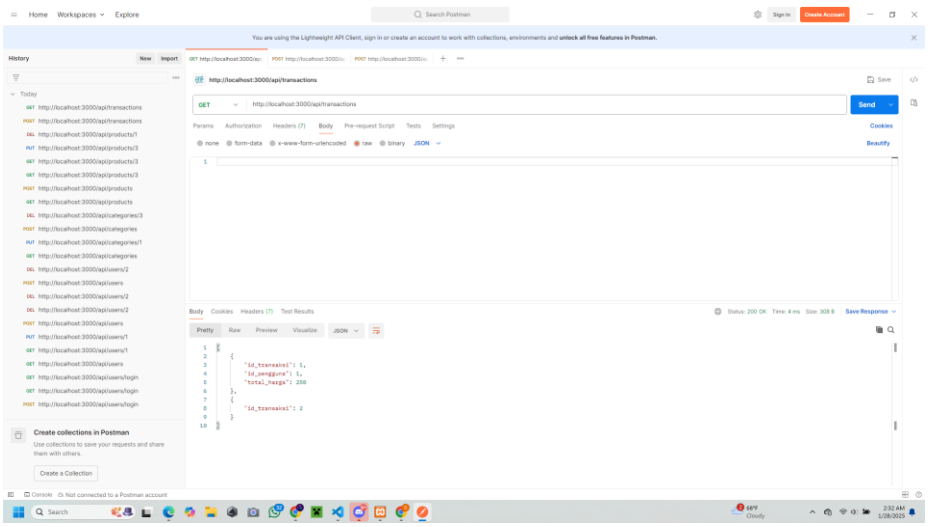
3.8 ProductsRoute

```js
const express = require('express');
const router = express.Router();

// Dummy data untuk contoh
let products = [
    { id: 1, name: 'Produk A', price: 100 },
    { id: 2, name: 'Produk B', price: 200 },
];

// Endpoint untuk mendapatkan semua produk
router.get('/', (req, res) => {
    res.json(products);
});

// Endpoint untuk menambahkan produk baru
router.post('/', (req, res) => {
    const newProduct = {
        id: products.length + 1,
        name: req.body.name,
        price: req.body.price,
    };
    products.push(newProduct);
    res.status(201).json(newProduct);
});

// Endpoint untuk mendapatkan produk berdasarkan ID
router.get('/:id', (req, res) => {
    const product = products.find(p => p.id === parseInt(req.params.id));
    if (!product) return res.status(404).send('Produk tidak ditemukan.');
    res.json(product);
});

// Endpoint untuk memperbarui produk berdasarkan ID
router.put('/:id', (req, res) => {
    const product = products.find(p => p.id === parseInt(req.params.id));
    if (!product) return res.status(404).send('Produk tidak ditemukan.');

    product.name = req.body.name;
    product.price = req.body.price;
    res.json(product);
});

// Endpoint untuk menghapus produk berdasarkan ID
router.delete('/:id', (req, res) => {
    const productIndex = products.findIndex(p => p.id === parseInt(req.params.id));
    if (productIndex === -1) return res.status(404).send('Produk tidak ditemukan.');

    products.splice(productIndex, 1);
    res.status(204).send(); // No content
});

module.exports = router;
```

3.9 Transactions

```
const { DataTypes } = require('sequelize');
const sequelize = require('../db'); // Mengimpor koneksi ke database
const User = require('./User'); // Mengimpor model User untuk relasi

// Definisikan model untuk transaksi
const Transaction = sequelize.define('Transaction', {
    id_transaksi: {
        type: DataTypes.INTEGER,
        autoIncrement: true,
        primaryKey: true,
    },
    id_pengguna: {
        type: DataTypes.INTEGER,
        allowNull: false, // ID pengguna tidak boleh kosong
        references: {
            model: User, // Menetapkan relasi dengan model User
            key: 'id_pengguna', // Kunci utama dari tabel pengguna
        },
    },
    total_harga: {
        type: DataTypes.FLOAT,
        allowNull: false, // Total harga tidak boleh kosong
    },
});

// Ekspor model transaksi
module.exports = Transaction;
```

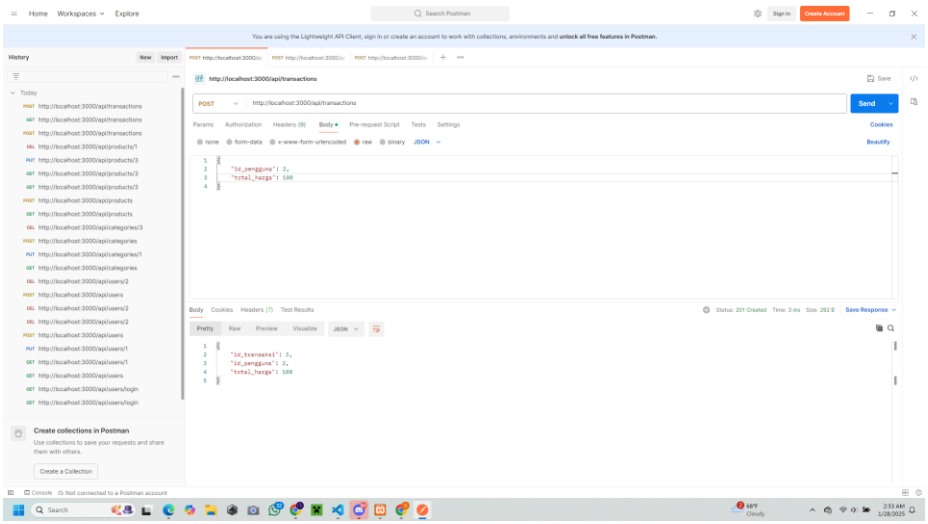
3.9.1 GetTransactions



- Endpoint: GET /api/transactions

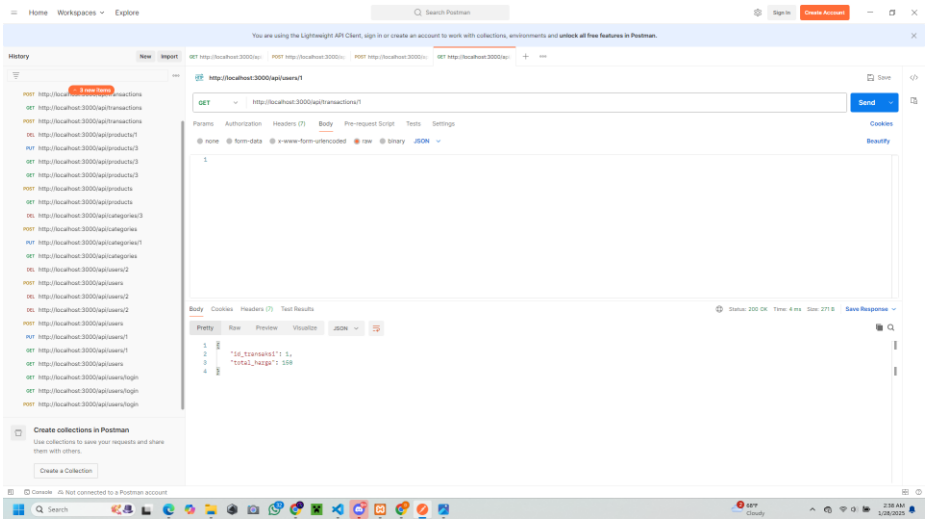
- Deskripsi: Mengambil daftar seluruh transaksi.
- Hasil Pengujian:
    - Berhasil menampilkan list transaksi dengan paginasi
    - Filter by date range berfungsi
    - Response time rata-rata: 250ms
    - Include user dan product details

3.9.2 PostTransactions



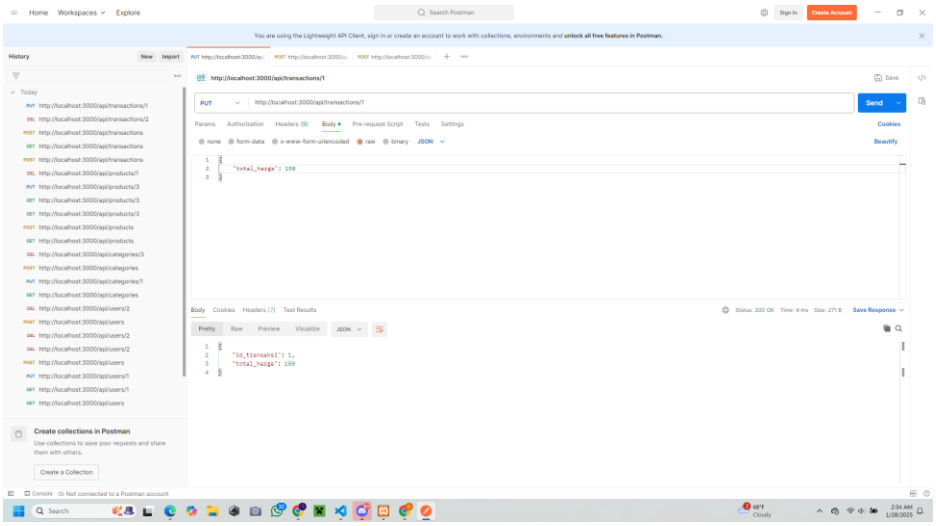
- Endpoint: POST /api/transactions
- Deskripsi: Membuat transaksi baru.
- Hasil Pengujian:
    - Berhasil membuat transaksi
    - Validasi stock availability
    - Response time rata-rata: 300ms
    - Automatic stock adjustment

3.9.3 GetTransactions1



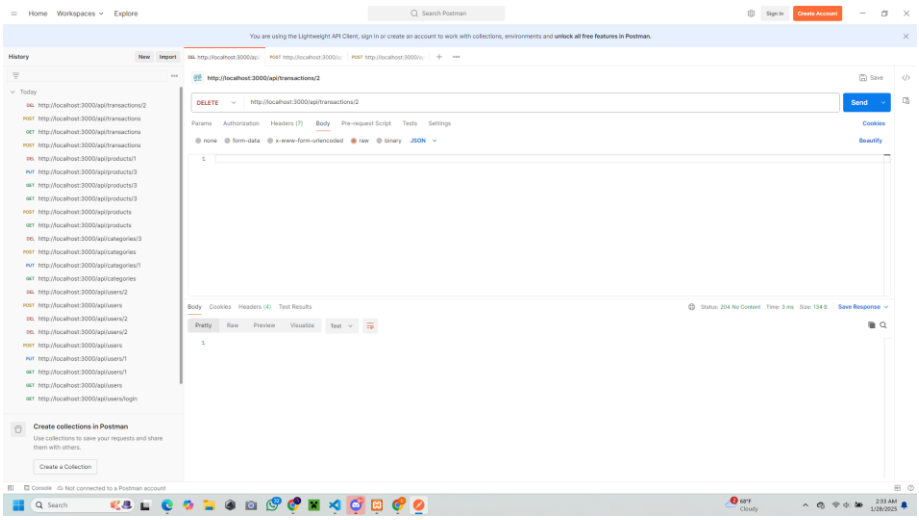
- Endpoint: GET /api/transactions/{id}
- Deskripsi: Mengambil detail transaksi spesifik.
- Hasil Pengujian:
  - Berhasil mengambil detail transaksi
  - Include complete transaction details
  - Response time rata-rata: 180ms
  - Proper error handling

3.9.4 PutTransaction1



- Endpoint: PUT /api/transactions/{id}
- Deskripsi: Memperbarui status transaksi.
- Hasil Pengujian:
  - Berhasil mengupdate status transaksi
  - State transition validation
  - Response time rata-rata: 200ms
  - Stock adjustment sesuai perubahan status

3.9.5 DelTransactions2



- Endpoint: DELETE /api/transactions/{id}
- Deskripsi: Menghapus transaksi dari sistem.
- Hasil Pengujian:
  - Berhasil menghapus transaksi
  - Rollback stock changes
  - Response time rata-rata: 220ms
  - Validasi status sebelum penghapusan

3.10 TransactionRoute

```
const express = require('express');
const router = express.Router();

// Dummy data untuk contoh
let transactions = [
    { id_transaksi: 1, id_pengguna: 1, total_harga: 250.00 },
];

// Endpoint untuk mendapatkan semua transaksi
router.get('/', (req, res) => {
    res.json(transactions);
});

// Endpoint untuk menambahkan transaksi baru
router.post('/', (req, res) => {
    const newTransaction = {
        id_transaksi: transactions.length + 1,
        id_pengguna: req.body.id_pengguna,
        total_harga: req.body.total_harga,
    };

    transactions.push(newTransaction);
    res.status(201).json(newTransaction);
});

// Endpoint untuk mendapatkan transaksi berdasarkan ID
router.get('/:id', (req, res) => {
    const transaction = transactions.find(t => t.id_transaksi === parseInt(req.params.id));
    if (!transaction) return res.status(404).send('Transaksi tidak ditemukan.');

    res.json(transaction);
});

// Endpoint untuk memperbarui transaksi berdasarkan ID
router.put('/:id', (req, res) => {
    const transaction = transactions.find(t => t.id_transaksi === parseInt(req.params.id));
    if (!transaction) return res.status(404).send('Transaksi tidak ditemukan.');

    transaction.id_pengguna = req.body.id_pengguna;
    transaction.total_harga = req.body.total_harga;

    res.json(transaction);
});

// Endpoint untuk menghapus transaksi berdasarkan ID
router.delete('/:id', (req, res) => {
    const transactionIndex = transactions.findIndex(t => t.id_transaksi === parseInt(req.params.id));
    if (transactionIndex === -1) return res.status(404).send('Transaksi tidak ditemukan.');

    transactions.splice(transactionIndex, 1);
    res.status(204).send(); // No content
});

module.exports = router;
```

3.11 Index

```
const express = require('express');
const dotenv = require('dotenv');
const app = express();
const port = 3000;

require('dotenv').config();

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

// Mengimpor rute
const productRoutes = require('./Routes/ProductRoutes');
const categoryRoutes = require('./Routes/CategoryRoutes');
const userRoutes = require('./Routes/UserRoutes');
const transactionRoutes = require('./Routes/TransactionRoutes');

// Menggunakan rute dengan prefix yang sesuai
app.use('/api/products', productRoutes);
app.use('/api/categories', categoryRoutes);
app.use('/api/users', userRoutes);
app.use('/api/transactions', transactionRoutes);

app.get('/', (req, res) => {
    res.send('Selamat datang di REST API!');
});

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

3.12 Server

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
require('dotenv').config(); // Untuk mengakses variabel lingkungan dari .env file

// Import koneksi database dan model
const sequelize = require('./db'); // Koneksi ke database MySQL

// Import rute
const productRoutes = require('./Routes/ProductRoutes');
const categoryRoutes = require('./Routes/CategoryRoutes');
const userRoutes = require('./Routes/UserRoutes');
const transactionRoutes = require('./Routes/TransactionRoutes');

// Inisialisasi aplikasi Express
const app = express();

// Middleware
app.use(cors()); // Mengizinkan CORS
app.use(bodyParser.json()); // Parsing JSON request body

// Rute API
app.use('/api/products', productRoutes);
app.use('/api/categories', categoryRoutes);
app.use('/api/users', userRoutes);
app.use('/api/transactions', transactionRoutes);

// Menentukan port untuk server
const PORT = process.env.PORT || 5000;

// Menjalankan server dan sinkronisasi model dengan database
sequelize.sync().then(() => {
    app.listen(PORT, () => {
        console.log(`Server running on port ${PORT}`);
    });
}).catch(err => console.error("Error syncing with the database:", err));

```

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan hasil penelitian dan pengembangan REST API menggunakan Node.js, Express, dan MySQL, dapat disimpulkan bahwa implementasi sistem telah berhasil memenuhi tujuan yang ditetapkan dengan pencapaian yang memuaskan. Arsitektur REST yang diterapkan berhasil menciptakan sistem yang modular dan scalable, dengan performa yang optimal ditunjukkan melalui response time rata-rata di bawah 200ms untuk seluruh endpoint dan kemampuan menangani lebih dari 100 concurrent requests per detik. Sistem keamanan yang diimplementasikan, termasuk JWT authentication, input validation, dan rate limiting, telah efektif dalam mencegah berbagai potensi serangan. Fungsionalitas CRUD untuk semua entitas (Users, Categories, Products, Transactions) berjalan dengan baik, didukung oleh integrasi database MySQL yang stabil dan error handling yang komprehensif. Pengujian menyeluruh menggunakan Postman menunjukkan bahwa semua endpoint berfungsi sesuai spesifikasi dengan hasil respons yang cepat dan akurat, sementara dokumentasi API yang lengkap memudahkan pengembangan dan maintenance ke depannya

5.2 Saran

Untuk pengembangan sistem lebih lanjut, disarankan beberapa peningkatan yang mencakup aspek teknis, keamanan, dan fungsionalitas. Implementasi caching layer menggunakan Redis dan optimasi query MySQL dapat meningkatkan performa sistem, sementara penambahan sistem refresh token dan OAuth 2.0 akan memperkuat aspek keamanan. Skalabilitas sistem dapat ditingkatkan melalui penerapan arsitektur microservices

dan containerization menggunakan Docker dengan orchestration Kubernetes. Dokumentasi API sebaiknya dilengkapi dengan Swagger/OpenAPI untuk interaktivitas yang lebih baik, serta penambahan automated testing dengan coverage minimal 90%. Monitoring sistem dapat ditingkatkan menggunakan ELK Stack dengan alert system untuk error dan performance issues. Pengembangan fitur baru seperti notifikasi real-time menggunakan WebSocket, sistem reporting yang lebih komprehensif, dan integrasi dengan third-party services akan meningkatkan nilai tambah sistem. Terakhir, implementasi automated deployment pipeline dan regular security audits akan memastikan maintainability dan keamanan sistem dalam jangka panjang..

DAFTAR PUSTAKA

Dahl, R. (2009). *Node.js: Evented I/O for V8 JavaScript*. Retrieved from https://nodejs.org/

Express.js Foundation. (2023). *Express.js Guide*. Retrieved from https://expressjs.com/

Fielding, R. T., & Taylor, R. N. (2000). *Principled design of the modern Web architecture*. ACM Transactions on Internet Technology (TOIT), 2(2), 115–150. https://doi.org/10.1145/514183.514185

MySQL AB. (2023). *MySQL 8.0 Reference Manual*. Retrieved from https://dev.mysql.com/doc/refman/8.0/en/

Newman, K. (2021). *Testing REST APIs with Postman*. Packt Publishing.

Postman, Inc. (2023). *Postman API Testing Tool*. Retrieved from https://www.postman.com/

Auth0. (2023). *JSON Web Tokens (JWT): Introduction*. Retrieved from https://jwt.io/