Laporan Modul Praktik

Membuat REST API : Perancangan Sistem Pemesanan Makanan Online menggunakan Node.js

Diajukan Untuk Memenuhi Salah Satu Tugas Project UAS Mata Kuliah Pemograman Berbasis Platfrom

Dosen Pengampu : Muhammad Ikhsan Thohir, M.Kom

Disusun Oleh Kelompok 6 :

Muhamad Rizky Fadillah	20230040083
Muhamad Haris Dirmansyah	20230040148
Ilham Adi Muslim	20230040156

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS KOMPUTER DAN DESAIN

NUSA PUTRA UNIVERSITY

2024/2025

1. Pendahuluan

1.1 Latar Belakang

Dalam era digital saat ini, layanan pemesanan makanan secara online semakin berkembang pesat. Aplikasi pemesanan makanan memungkinkan pelanggan untuk dengan mudah memesan makanan dari berbagai restoran hanya dengan menggunakan perangkat mobile atau komputer. Untuk mendukung kebutuhan tersebut, diperlukan sistem yang efisien dalam mengelola data pengguna, restoran, menu, dan pesanan.

Dalam pengembangan sistem ini, pemilihan teknologi yang tepat sangat penting. Salah satu aspek utama dalam pengembangan aplikasi adalah manajemen basis data. Oleh karena itu, digunakan Prisma ORM sebagai alat untuk mempermudah interaksi dengan database, mengurangi kompleksitas query SQL, dan meningkatkan keamanan serta efisiensi dalam pengelolaan data.

1.2 Tujuan Penelitian

Tujuan dari laporan ini adalah:
Menjelaskan konsep ORM dan Prisma dalam pengelolaan database.
Menguraikan bagaimana Prisma digunakan dalam implementasi aplikasi pemesanan makanan online.
Menunjukkan bagaimana Prisma ORM dapat meningkatkan efisiensi dan kemudahan dalam pengelolaan basis data.

2.Kajian Teori

2.1 Object-Relational Mapping (ORM)

Object-Relational Mapping (ORM) adalah teknik dalam pemrograman yang memungkinkan pengembang untuk berinteraksi dengan database menggunakan kode pemrograman berorientasi objek tanpa harus menulis query SQL secara langsung. Dengan ORM, database dapat dikelola melalui representasi objek dalam bahasa pemrograman.

2.2 Prisma ORM

Prisma adalah ORM modern yang mendukung berbagai sistem database seperti PostgreSQL, MySQL, SQLite, dan MongoDB. Prisma menawarkan tiga komponen utama:
Prisma Client: Sebuah library yang digunakan untuk berinteraksi dengan database menggunakan JavaScript atau TypeScript.
Prisma Migrate: Alat untuk mengelola skema database dan migrasi data.
Prisma Studio: Antarmuka visual untuk melihat dan mengelola data dalam database.
Keunggulan Prisma dibanding ORM lain:
Menyediakan fitur otomatisasi query untuk CRUD (Create, Read, Update, Delete).
Mendukung validasi tipe data otomatis dan mencegah SQL Injection.
Mudah diintegrasikan dengan framework seperti Express.js.

2.3 REST API dalam Aplikasi Pemesanan Makanan

REST API adalah arsitektur yang digunakan untuk membangun layanan web yang memungkinkan komunikasi antara client dan server menggunakan metode HTTP seperti GET, POST, PUT, dan DELETE. Dalam aplikasi pemesanan makanan online, REST API digunakan untuk menghubungkan frontend dengan backend.

3. Landasan Teori dengan Implementasi

Kajian teori ini mendukung pendekatan sistematis dalam perancangan aplikasi pemesanan makanan online. REST API digunakan untuk memastikan komunikasi data yang efisien antara frontend dan backend. Framework backend yang dipilih dirancang untuk menangani permintaan dengan cepat dan mendukung skalabilitas sistem. Penggunaan database relasional memastikan pengelolaan data pesanan, pengguna, dan restoran secara terstruktur dan aman. Metode autentikasi yang diterapkan meningkatkan keamanan akses pengguna dalam sistem. Pengujian dilakukan untuk memastikan keandalan layanan dalam menangani transaksi pemesanan makanan secara real-time. Dengan pendekatan ini, sistem yang dirancang mampu menyediakan pengalaman pengguna yang optimal dan efisien dalam pemesanan makanan secara daring.

4. Analisis Kebutuhan

➢ Alat dan Teknologi

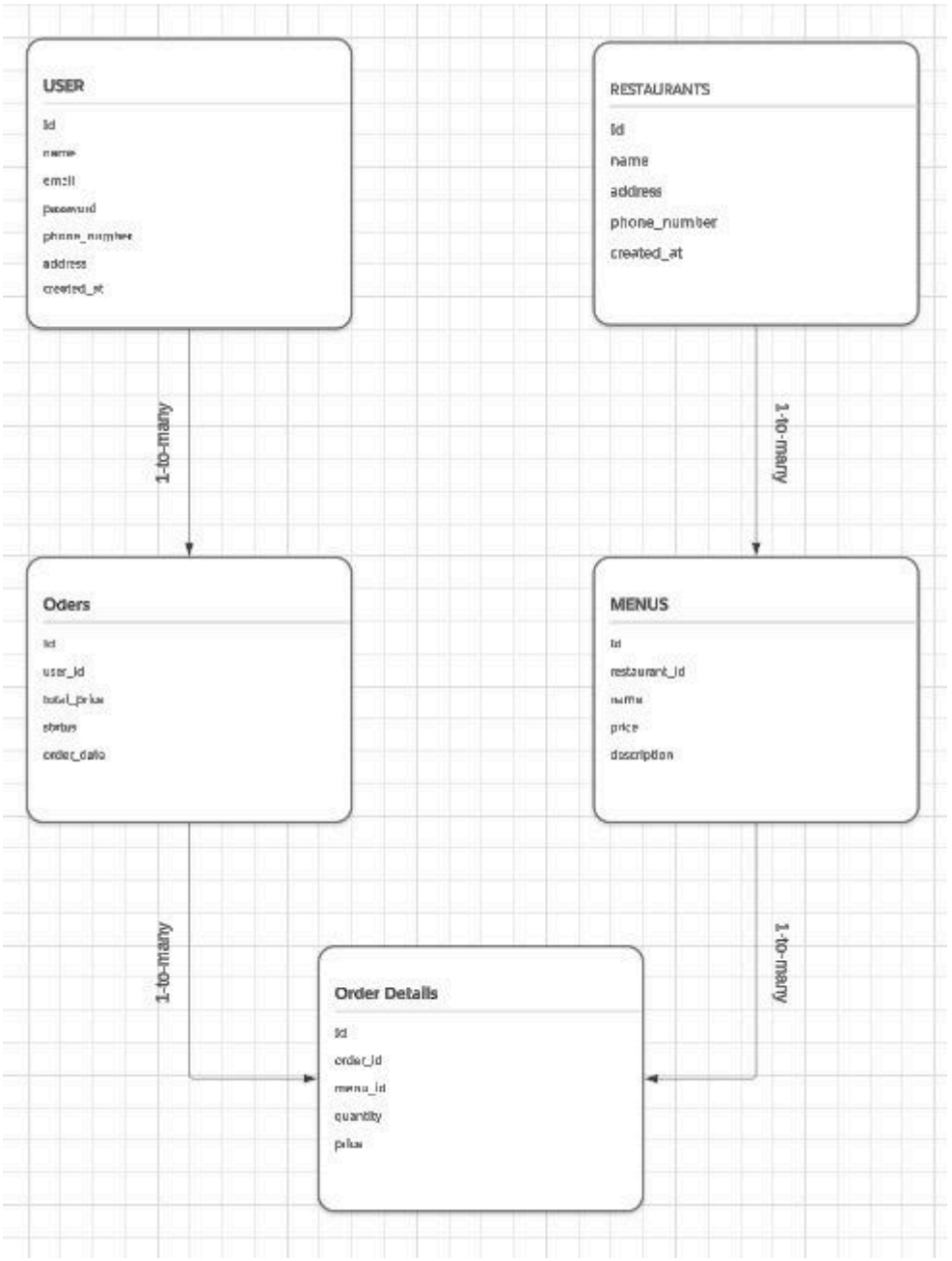
• Backend: Node.js, Express.js
• Database: MySQL
• Pengujian Endpoint: Postman
• Autentikasi: JWT untuk validasi pengguna

➢ Fitur Utama

• Manajemen Pengguna
  - Registrasi pengguna.
  - Login.
  - Pembaruan profil pengguna.
• Manajemen Menu Makanan:
  - Menampilkan daftar makanan (GET /menu).
  - Menampilkan detail makanan tertentu (GET /menu/{id}).
  - Menambahkan makanan baru (POST /menu) (admin).
  - Mengedit informasi makanan (PUT /menu/{id}) (admin).
  - Menghapus makanan (DELETE /menu/{id}) (admin).
• Manajemen Pesanan:
  - Membuat pesanan baru (POST /orders).
  - Melihat semua pesanan pengguna (GET /orders).
  - Melihat detail pesanan tertentu (GET /orders/{id}).
  - Membatalkan pesanan (DELETE /orders/{id}).
• Pembayaran:
  - Mendapatkan detail pembayaran (GET /payment/{order_id}).
  - Mengonfirmasi pembayaran (POST /payment/confirm).
• Pencarian dan Filter:
  - Mencari makanan berdasarkan nama atau kategori (GET /menu/search).
  - Menyaring makanan berdasarkan harga atau popularitas (GET /menu/filter).

• Manajemen Keranjang Belanja:
  - Menambahkan makanan ke keranjang (POST /cart).
  - Melihat isi keranjang (GET /cart).
  - Mengedit jumlah item dalam keranjang (PUT /cart/{item_id}).
  - Menghapus item dari keranjang (DELETE /cart/{item_id}).
• Notifikasi Pesanan:
  - Mendapatkan status real-time dari pesanan (GET /orders/{id}/status).

5. Rancangan Database



6. Skema Database Toko Makanan Online

Users (Pengguna)

user_id (PK):
name:
email:
password:
phone_number:
address:
role (admin/customer):
Menu (Makanan):

menu_id (PK):
name:
description:
price:
category:
image_url:
stock:
Orders (Pesanan):

order_id (PK):
user_id (FK → Users.user_id):
order_date:
total_price:
status (pending/confirmed/canceled/completed):
Order_Items (Detail Pesanan):

order_item_id (PK):
order_id (FK → Orders.order_id):
menu_id (FK → Menu.menu_id):
quantity:
price:

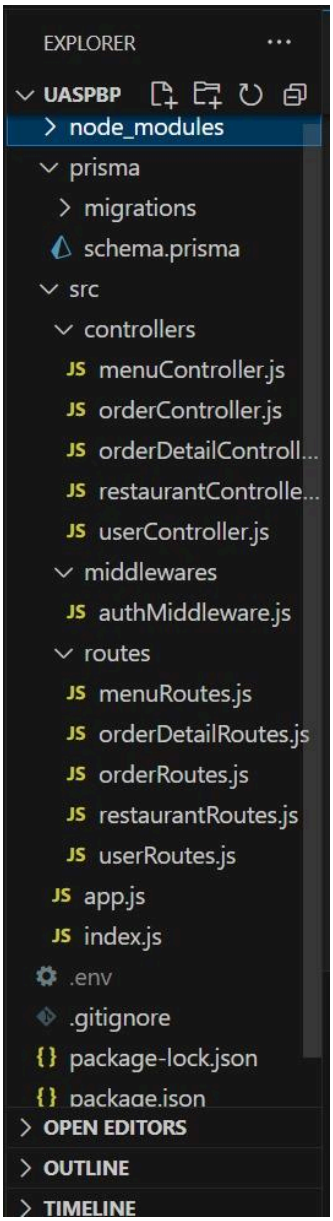
7. Data dan Analisis Pada analisis kebutuhan,

laporan merinci fitur inti, seperti autentikasi pengguna, manajemen produk, pesanan, dan detail pesanan. Struktur database dirancang sistematis untuk mendukung hubungan antar data menggunakan tabel utama seperti Pengguna, Produk, Pesanan, dan Detail Pesanan. Proses implementasi memanfaatkan metodologi pengujian API menggunakan Postman untuk menjamin keakuratan dan keandalan sistem.

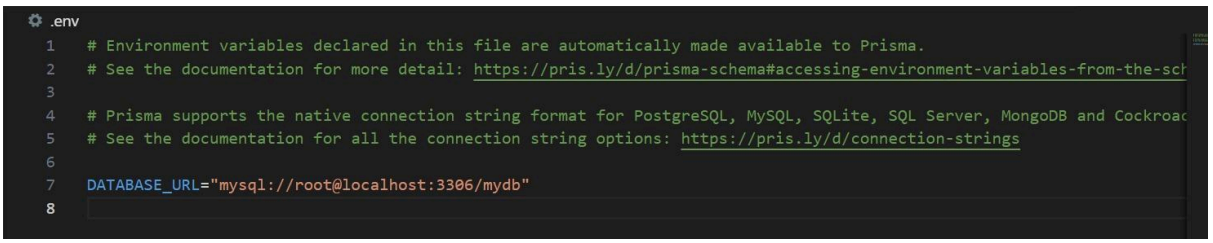
Pengujian yang dilakukan membuktikan bahwa setiap fitur, termasuk operasi CRUD (Create, Read, Update, Delete), bekerja dengan baik dalam lingkungan lokal. Dengan demikian, laporan ini tidak hanya mendokumentasikan tahap perancangan tetapi juga menyoroti keberhasilan implementasi sistem secara menyeluruh.

8.Implementasi

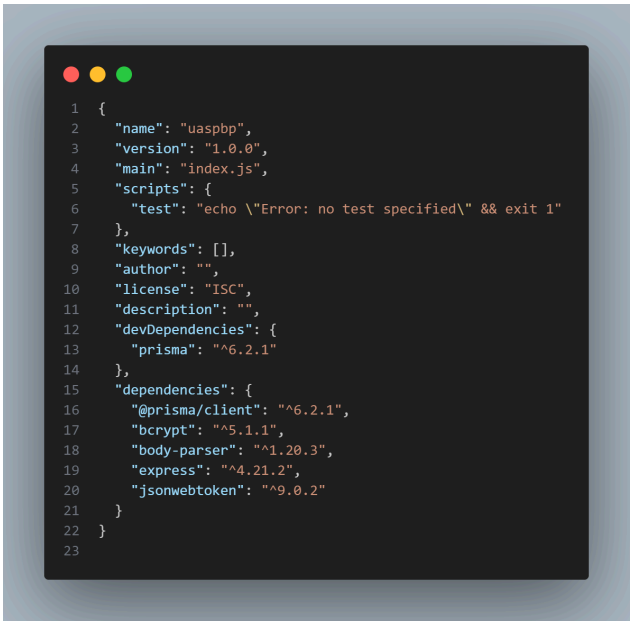
➢Konfigurasi Koneksi Database



➢Struktur direktori proyek



➢**Installasi paket node.js**



```
{
  "name": "uaspbp",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "devDependencies": {
    "prisma": "^6.2.1"
  },
  "dependencies": {
    "@prisma/client": "^6.2.1",
    "bcrypt": "^5.1.1",
    "body-parser": "^1.20.3",
    "express": "^4.21.2",
    "jsonwebtoken": "^9.0.2"
  }
}
```

➢**Data base menggunakan prisma**

```
// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

// Looking for ways to speed up your queries, or scale easily with your serverless or edge functions?
// Try Prisma Accelerate: https://pris.ly/cli/accelerate-init

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mysql"
  url      = env("DATABASE_URL")
}

model User {
  id Int @id @default(autoincrement())
  name String
  email String @unique
  password String
  phone_number Int
  address String
  created_at DateTime @default(now())
  orders order[]
}

model Restaurant {
  id Int @id @default(autoincrement())
  name String
  address String
  phone_number Int
  created_at DateTime @default(now())
  menus Menu[]
}

model Menu {
  id Int @id @default(autoincrement())
  restaurant_id Int
  restaurant Restaurant @relation(fields: [restaurant_id], references: [id])
  name String
  price Int
  description String
  order_details order_detail[]
}

model order {
  id Int @id @default(autoincrement())
  user_id Int
  user User @relation(fields: [user_id], references: [id])
  total_price Int
  status String
  order_date DateTime @default(now())
  order_details order_detail[]
}

model order_detail {
  id Int @id @default(autoincrement())
  order_id Int
  menu_id Int
  quantity Int
  price Int
  order order @relation(fields: [order_id], references: [id])
  menu Menu @relation(fields: [menu_id], references: [id])
}
```

➢server utama menggunakan Express.js, yang menghubungkan berbagai routes

```
const express = require('express');
const bodyParser = require('body-parser');
const userRoutes = require('./routes/userRoutes');
const restaurantRoutes = require('./routes/restaurantRoutes')
const menuRoutes = require('./routes/menuRoutes')
const orderRoutes = require('./routes/orderRoutes')
const orderDetail = require('./routes/orderDetailRoutes')

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

app.use('/users', userRoutes);
app.use('/restaurant', restaurantRoutes);
app.use('/menu', menuRoutes)
app.use('/order', orderRoutes)
app.use('/orderDetail', orderDetail)

const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});

```

➢Autentikasi menggunakan JWT (JSON Web Token)

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

const authenticateToken = (req, res, next) => {
  const token = req.header('Authorization')?.split(' ')[1];
  if (!token) return res.status(401).json({ error: 'Access denied' });

  try {
    const verified = jwt.verify(token, 'SECRET_KEY');
    req.user = verified;
    next();
  } catch (error) {
    res.status(403).json({ error: 'Invalid token' });
  }
};

module.exports = { authenticateToken };
```

➢controlorderdetail

```
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

const createOrderDetail = async (req, res) => {
  const { order_id, menu_id, quantity, price } = req.body;
  try {
    const newOrderDetail = await prisma.order_detail.create({
      data: {
        order_id,
        menu_id,
        quantity,
        price,
      },
    });
    res.status(201).json({ message: 'Order detail created successfully', newOrderDetail });
  } catch (error) {
    res.status(500).json({ error: 'Error creating order detail', details: error.message });
  }
};

const getAllOrderDetails = async (req, res) => {
  try {
    const orderDetails = await prisma.order_detail.findMany({
      include: {
        order: true,
        menu: true,
      },
    });
    res.status(200).json(orderDetails);
  } catch (error) {
    res.status(500).json({ error: 'Error fetching order details', details: error.message });
  }
};

const getOrderDetailById = async (req, res) => {
  const { id } = req.params;
  try {
    const orderDetail = await prisma.order_detail.findUnique({
      where: { id: parseInt(id) },
      include: {
        order: true,
        menu: true,
      },
    });
    if (!orderDetail) return res.status(404).json({ error: 'Order detail not found' });
    res.status(200).json(orderDetail);
  } catch (error) {
    res.status(500).json({ error: 'Error fetching order detail', details: error.message });
  }
};

const deleteOrderDetail = async (req, res) => {
  const { id } = req.params;
  try {
    await prisma.order_detail.delete({
      where: { id: parseInt(id) },
    });
    res.status(200).json({ message: 'Order detail deleted successfully' });
  } catch (error) {
    res.status(500).json({ error: 'Error deleting order detail', details: error.message });
  }
};

module.exports = {
  createOrderDetail,
  getAllOrderDetails,
  getOrderDetailById,
  deleteOrderDetail,
};

```

➢controlrestaurant

```
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

const createRestaurant = async (req, res) => {
  const { name, address, phone_number } = req.body;
  try {
    const newRestaurant = await prisma.restaurant.create({
      data: {
        name,
        address,
        phone_number,
      },
    });
    res.status(201).json({ message: 'Restaurant created successfully', newRestaurant });
  } catch (error) {
    res.status(500).json({ error: 'Error creating restaurant', details: error.message });
  }
};

const getAllRestaurants = async (req, res) => {
  try {
    const restaurants = await prisma.restaurant.findMany();
    res.status(200).json(restaurants);
  } catch (error) {
    res.status(500).json({ error: 'Error fetching restaurants', details: error.message });
  }
};

const getRestaurantById = async (req, res) => {
  const { id } = req.params;
  try {
    const restaurant = await prisma.restaurant.findUnique({
      where: { id: parseInt(id) },
    });
    if (!restaurant) {
      return res.status(404).json({ error: 'Restaurant not found' });
    }
    res.status(200).json(restaurant);
  } catch (error) {
    res.status(500).json({ error: 'Error fetching restaurant', details: error.message });
  }
};

const updateRestaurant = async (req, res) => {
  const { id } = req.params;
  const { name, address, phone_number } = req.body;
  try {
    const updatedRestaurant = await prisma.restaurant.update({
      where: { id: parseInt(id) },
      data: {
        name,
        address,
        phone_number,
      },
    });
    res.status(200).json({ message: 'Restaurant updated successfully', updatedRestaurant });
  } catch (error) {
    res.status(500).json({ error: 'Error updating restaurant', details: error.message });
  }
};

const deleteRestaurant = async (req, res) => {
  const { id } = req.params;
  try {
    await prisma.restaurant.delete({ where: { id: parseInt(id) } });
    res.status(200).json({ message: 'Restaurant deleted successfully' });
  } catch (error) {
    res.status(500).json({ error: 'Error deleting restaurant', details: error.message });
  }
};

module.exports = {
  createRestaurant,
  getAllRestaurants,
  getRestaurantById,
  updateRestaurant,
  deleteRestaurant,
};

```

➢**controluser**

```js
const { PrismaClient } = require('@prisma/client');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');

const prisma = new PrismaClient();

const register = async (req, res) => {
  const { name, email, password, phone_number, address } = req.body;
  try {
    const hashedPassword = await bcrypt.hash(password, 10);
    const user = await prisma.user.create({
      data: {
        name,
        email,
        password: hashedPassword,
        phone_number,
        address,
      },
    });
    res.status(201).json({ message: 'User registered successfully', user });
  } catch (error) {
    res.status(500).json({ error: 'Error registering user', details: error });
  }
};

const login = async (req, res) => {
  const { email, password } = req.body;
  try {
    const user = await prisma.user.findUnique({ where: { email } });
    if (!user) return res.status(404).json({ error: 'User not found' });

    const isPasswordValid = await bcrypt.compare(password, user.password);
    if (!isPasswordValid)
      return res.status(401).json({ error: 'Invalid credentials' });

    const token = jwt.sign({ id: user.id, email: user.email }, 'SECRET_KEY', {
      expiresIn: '1h',
    });
    res.status(200).json({ message: 'Login successful', token });
  } catch (error) {
    res.status(500).json({ error: 'Error logging in', details: error });
  }
};

const getAllUsers = async (req, res) => {
  try {
    const users = await prisma.user.findMany();
    res.status(200).json(users);
  } catch (error) {
    res.status(500).json({ error: 'Error fetching users', details: error });
  }
};

const updateUser = async (req, res) => {
    const { id } = req.params;
    const data = req.body;
    try {
      const updatedUser = await prisma.user.update({
        where: { id: parseInt(id) },
        data,
      });
      res.status(200).json({ message: 'User updated successfully', updatedUser });
    } catch (error) {
      res.status(500).json({ error: 'Error updating user', details: error });
    }
  };

const deleteUser = async (req, res) => {
  const { id } = req.params;
  try {
    await prisma.user.delete({ where: { id: parseInt(id) } });
    res.status(200).json({ message: 'User deleted successfully' });
  } catch (error) {
    res.status(500).json({ error: 'Error deleting user', details: error });
  }
};

module.exports = { register, login, getAllUsers, updateUser, deleteUser };

```

➢**controlmenu**

```js
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

const createMenu = async (req, res) => {
  const { restaurant_id, name, price, description } = req.body;
  try {
    const newMenu = await prisma.menu.create({
      data: {
        restaurant_id,
        name,
        price,
        description,
      },
    });
    res.status(201).json({ message: 'Menu created successfully', newMenu });
  } catch (error) {
    res.status(500).json({ error: 'Error creating menu', details: error.message });
  }
};

const getAllMenus = async (req, res) => {
  try {
    const menus = await prisma.menu.findMany({
      include: {
        restaurant: true,
      },
    });
    res.status(200).json(menus);
  } catch (error) {
    res.status(500).json({ error: 'Error fetching menus', details: error.message });
  }
};

const getMenuById = async (req, res) => {
  const { id } = req.params;
  try {
    const menu = await prisma.menu.findUnique({
      where: { id: parseInt(id) },
      include: {
        restaurant: true,
      },
    });
    if (!menu) return res.status(404).json({ error: 'Menu not found' });

    res.status(200).json(menu);
  } catch (error) {
    res.status(500).json({ error: 'Error fetching menu', details: error.message });
  }
};

const updateMenu = async (req, res) => {
  const { id } = req.params;
  const data = req.body;
  try {
    const updatedMenu = await prisma.menu.update({
      where: { id: parseInt(id) },
      data,
    });
    res.status(200).json({ message: 'Menu updated successfully', updatedMenu });
  } catch (error) {
    res.status(500).json({ error: 'Error updating menu', details: error.message });
  }
};

const deleteMenu = async (req, res) => {
  const { id } = req.params;
  try {
    await prisma.menu.delete({
      where: { id: parseInt(id) },
    });
    res.status(200).json({ message: 'Menu deleted successfully' });
  } catch (error) {
    res.status(500).json({ error: 'Error deleting menu', details: error.message });
  }
};

module.exports = { createMenu, getAllMenus, getMenuById, updateMenu, deleteMenu };

```

➢**controlorder**

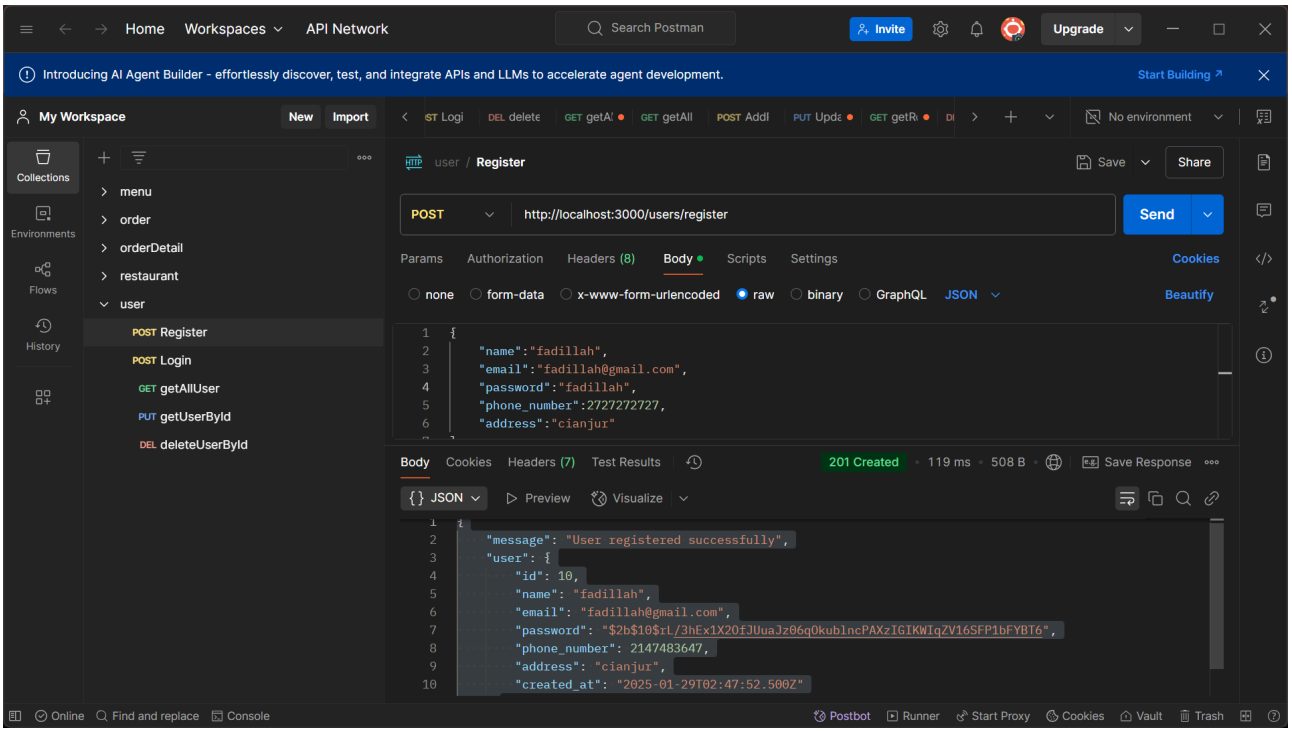
```
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

const createOrder = async (req, res) => {
  const { user_id, total_price, status } = req.body;
  try {
    const newOrder = await prisma.order.create({
      data: {
        user_id,
        total_price,
        status,
      },
    });
    res.status(201).json({ message: 'Order created successfully', newOrder });
  } catch (error) {
    res.status(500).json({ error: 'Error creating order', details: error.message });
  }
};

const getAllOrders = async (req, res) => {
  try {
    const orders = await prisma.order.findMany({
      include: {
        user: true,
      },
    });
    res.status(200).json(orders);
  } catch (error) {
    res.status(500).json({ error: 'Error fetching orders', details: error.message });
  }
};

const getOrderById = async (req, res) => {
  const { id } = req.params;
  try {
    const order = await prisma.order.findUnique({
      where: { id: parseInt(id) },
      include: {
        user: true,
      },
    });
    if (!order) return res.status(404).json({ error: 'Order not found' });
    res.status(200).json(order);
  } catch (error) {
    res.status(500).json({ error: 'Error fetching order', details: error.message });
  }
};

const updateOrderStatus = async (req, res) => {
  const { id } = req.params;
  const { status } = req.body;
  try {
    const updatedOrder = await prisma.order.update({
      where: { id: parseInt(id) },
      data: { status },
    });
    res.status(200).json({ message: 'Order status updated successfully', updatedOrder });
  } catch (error) {
    res.status(500).json({ error: 'Error updating order', details: error.message });
  }
};

module.exports = { createOrder, getAllOrders, getOrderById, updateOrderStatus };

```

➢**routesuser.png**

```
const express = require('express');
const {
    register,
    login,
    getAllUsers,
    updateUser,
    deleteUser,
} = require('../controllers/userController');
const { authenticateToken } = require('../middlewares/authMiddleware');

const router = express.Router();

router.post('/register', register);
router.post('/login', login);
router.get('/', authenticateToken, getAllUsers);
router.put('/:id', authenticateToken,updateUser);
router.delete('/:id', authenticateToken, deleteUser);

module.exports = router;
```

Berikut adalah penjelasan lengkap dan terperinci untuk pengujian API menggunakan Postman, termasuk “User”, “Restaurant”, “Menu”, “Order”, dan “OrderDetail”

1. Pengujian Registrasi dan Login User
- Tujuan: Menguji apakah proses pendaftaran dan login user berfungsi dengan benar.
Proses:
 - Registrasi:User mengirimkan data seperti nama, email, dan password untuk mendaftar. Jika email belum terdaftar, akun baru akan dibuat.
- Login: Setelah registrasi, user dapat melakukan login untuk mendapatkan token autentikasi.
-Keterangan: Token yang diterima setelah login akan digunakan untuk mengakses endpoint yang memerlukan autentikasi (misalnya untuk membuat restoran atau menambahkan menu).



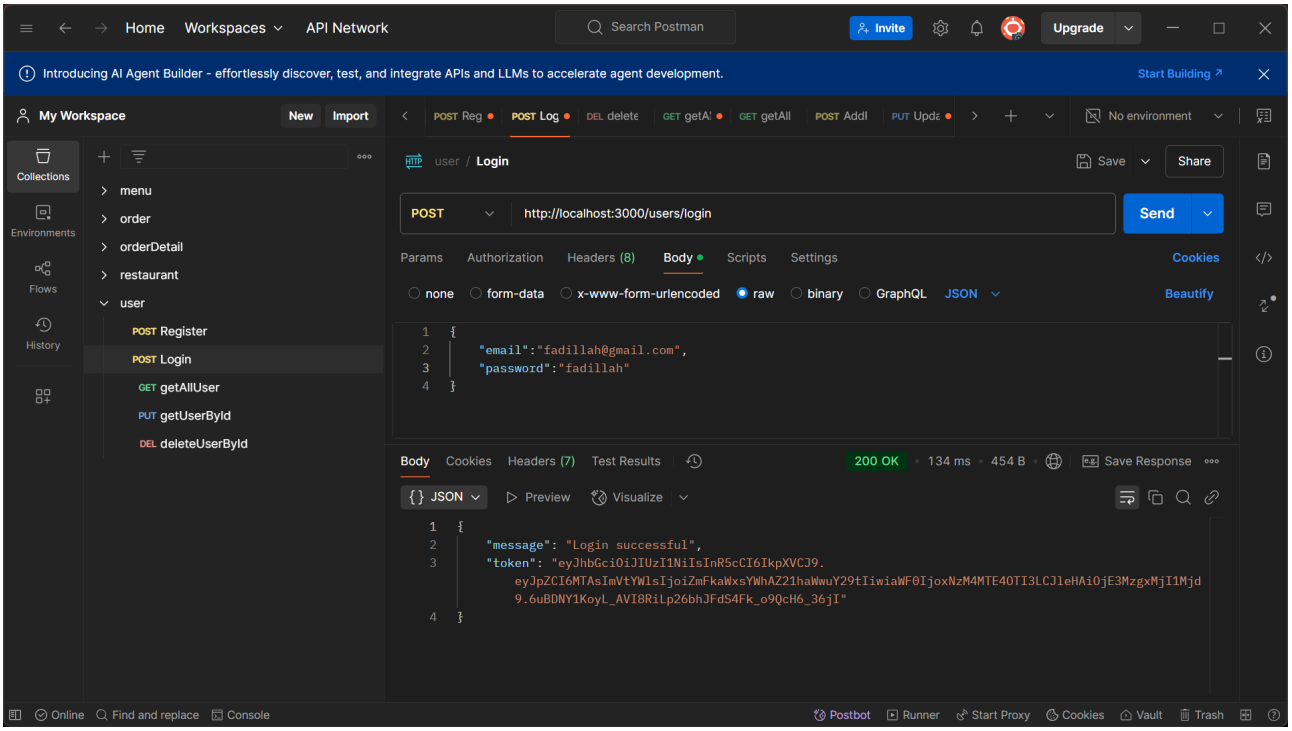
Contoh Pengujian:
1. Request Registrasi:
   - Endpoint: POST /register
   - Body:
     json
     {
       "name": "John Doe",
       "email": "john@example.com",
       "password": "password123"
     }

2. Request Login:
   - Endpoint: POST /login
   - Body:
     json
     {

"email": "john@example.com",
"password": "password123"
}

- Response:
json
{
"token": "your_token_here"
}

2. Pengujian Registrasi Restoran
- Tujuan: Menguji apakah restoran dapat didaftarkan oleh user yang sudah login.
Proses:
- User harus mengirimkan token autentikasi yang didapat dari login dalam header permintaan.
- Setelah login, user dapat menambahkan restoran baru dengan nama dan lokasi restoran.
- Keterangan:Hanya user yang sudah login yang dapat membuat restoran. Restoran yang dibuat akan dikaitkan dengan user yang membuatnya.

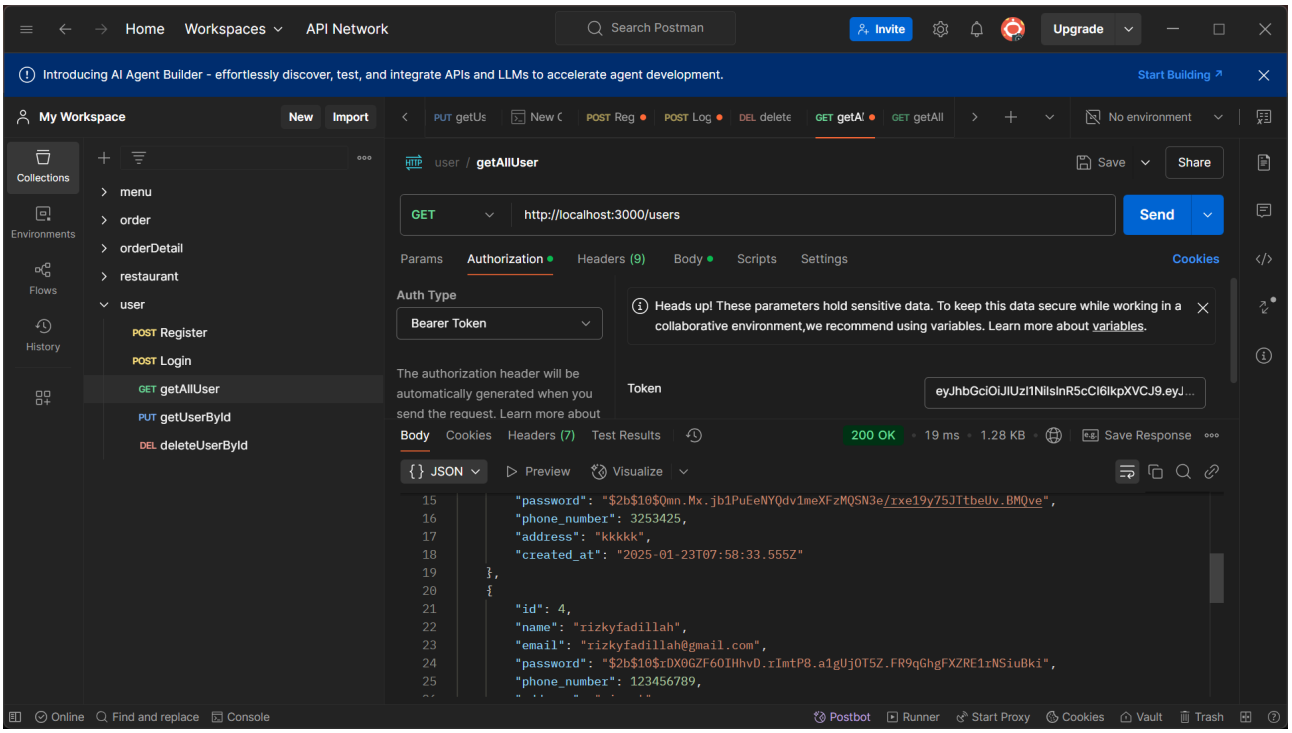


Contoh Pengujian:
1. Request Registrasi Restoran:
- Endpoint: POST /restaurant
- Header:
json
{
"Authorization": "Bearer your_token_here"
}

- Body:
  json
  {
    "name": "Sukabumi Resto",
    "location": "Jalan Merdeka No. 10, Sukabumi"
  }

- Response:
  json
  {
    "id": 1,
    "name": "Sukabumi Resto",
    "location": "Jalan Merdeka No. 10, Sukabumi",
    "owner": "John Doe"
  }

3. Pengujian Menambahkan Menu ke Restoran
- Tujuan: Menguji apakah menu baru dapat ditambahkan ke restoran yang sudah ada.
- Proses:
  - User harus mengirimkan id restoran untuk menambahkan menu baru ke restoran tersebut.
  - Menu yang ditambahkan harus mencakup nama menu dan harga.
  - Token autentikasi harus disertakan dalam header agar permintaan diterima.
- Keterangan: Jika restoran dengan id restoran yang diberikan tidak ditemukan, API akan mengembalikan error.



Contoh Pengujian:

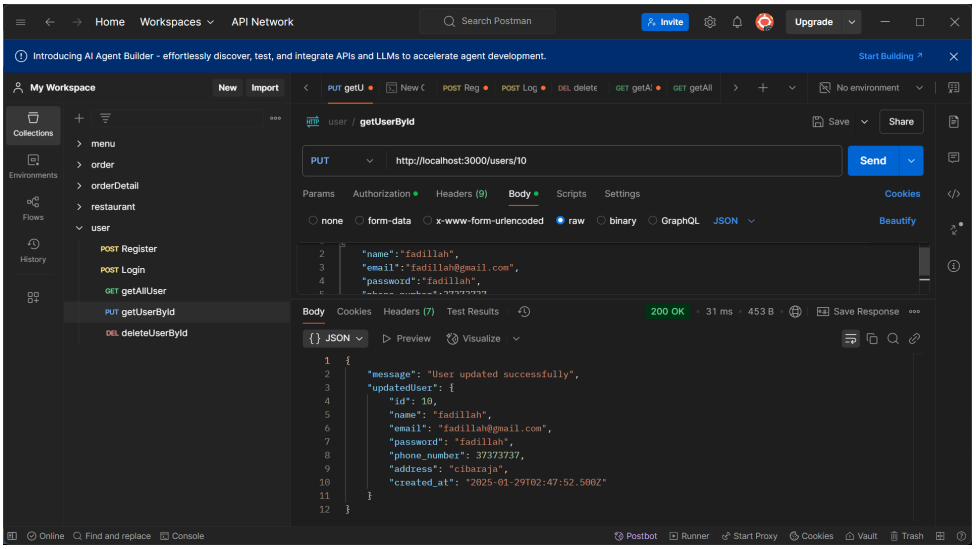
1. Request Menambahkan Menu:
   - Endpoint: POST /restaurant/{restaurantId}/menu
   - Header:
     json
     {
       "Authorization": "Bearer your_token_here"
     }

   - Body:
     json
     {
       "name": "Nasi Goreng Spesial",
       "price": 25000
     }

   - Response:
     json
     {
       "id": 1,
       "name": "Nasi Goreng Spesial",
       "price": 25000,
       "restaurantId": 1
     }

4. Pengujian Pemesanan (Order)

- Tujuan: Menguji apakah pemesanan makanan dapat dilakukan oleh user yang telah terautentikasi.
- Proses:
  - User harus mengirimkan token autentikasi yang diterima saat login untuk memverifikasi identitas.
  - Pemesanan membutuhkan menuId dan quantity (jumlah makanan yang dipesan).
  - Pemesanan juga harus mencantumkan id restoran yang menjual menu tersebut.
- Keterangan: Setelah pemesanan berhasil, sistem akan mengembalikan informasi pesanan, termasuk orderId dan status pemesanan. Order ini terkait dengan user yang melakukan pemesanan.



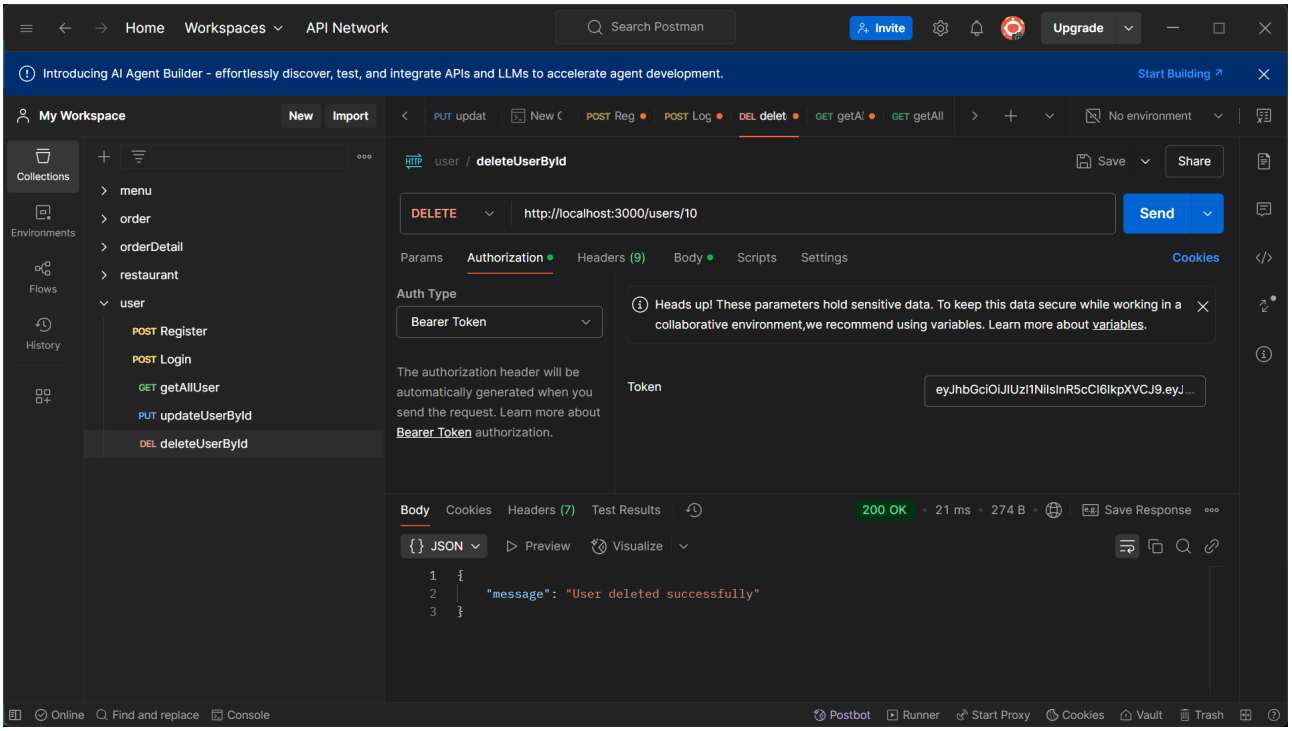
Contoh Pengujian:

1. Request Pemesanan (Order):
   - Endpoint: POST /order
   - Header:
     json
     {
       "Authorization": "Bearer your_token_here"
     }

   - Body:
     json
     {
       "restaurantId": 1,
       "menuId": 1,
       "quantity": 2
     }

   - Response:
     json
     {
       "orderId": 123,
       "restaurantId": 1,
       "menuId": 1,
       "quantity": 2,
       "totalPrice": 50000,
       "status": "pending",
       "userId": 1
     }

5. Pengujian Detail Pemesanan (OrderDetail)
- Tujuan: Menguji apakah informasi detail pesanan dapat dilihat setelah pemesanan dilakukan.
- Proses:
  - User yang sudah melakukan pemesanan dapat melihat detail pemesanan mereka menggunakan orderId.
  - Endpoint ini memberikan informasi tentang menu yang dipesan, jumlah pesanan, harga per item, dan total harga dari pesanan tersebut.
- Keterangan: User hanya dapat melihat detail dari pesanan yang mereka buat menggunakan token autentikasi.



Contoh Pengujian:

1. Request Detail Pemesanan (OrderDetail):
   - Endpoint: GET /order/{orderId}/detail
   - Header:
     json
     {
       "Authorization": "Bearer your_token_here"
     }

   - Response:
     json
     {
       "orderId": 123,
       "items": [
         {
           "menuId": 1,
           "name": "Nasi Goreng Spesial",
           "quantity": 2,
           "price": 25000,
           "totalPrice": 50000
         }
       ],
       "totalPrice": 50000,
       "status": "pending"
     }

Ringkasan:

- Registrasi & Login User: Mendaftar dan mendapatkan token autentikasi.
- Registrasi Restoran: Menggunakan token autentikasi untuk mendaftarkan restoran baru.
- Menambahkan Menu: Menambahkan menu ke restoran yang sudah ada, dengan menyertakan id restoran.
- Pemesanan (Order): User dapat memesan makanan dengan menyertakan menuId, restaurantId, dan quantity. Token autentikasi harus disertakan untuk verifikasi.
- Detail Pemesanan (OrderDetail): Setelah pemesanan berhasil, user dapat mengecek detail pesanan dengan menyertakan orderId untuk melihat daftar item yang dipesan dan total harga.

Dengan penjelasan ini, pengujian untuk seluruh sistem REST API, dari User, Restaurant, Menu, Order, hingga OrderDetail, akan lebih lengkap dan mudah dimengerti dalam dokumen pengujian API menggunakan Postman.

9.Kesimpulan

Laporan ini mengimplementasikan proses perancangan sistem pemesanan makanan online berbasis REST API dengan menggunakan teknologi modern seperti Node.js, Express.js, dan MySQL. Sistem ini dirancang untuk menyediakan fitur utama seperti autentikasi pengguna, manajemen produk, pengelolaan pesanan, serta detail pesanan. Setiap fitur didukung dengan validasi data yang aman menggunakan JsonWebToken (JWT).

Rancangan database yang sistematis memastikan integritas data melalui relasi antar tabel utama, seperti tabel pengguna, produk, pesanan, dan detail pesanan. Seluruh operasi CRUD diuji menggunakan Postman untuk menjamin keakuratan dan kestabilan API.

Laporan ini membuktikan bahwa teknologi yang digunakan dapat diimplementasikan secara efektif untuk menciptakan solusi digital berbasis platform