

Laporan Desain dan Perancangan REST API

Sistem Manajemen Toko Online

Oleh:

- Oscar Haris N

-Nanda Irsyad

-M.Fikri

1. Deskripsi Proyek

Sistem Manajemen Toko Online ini bertujuan untuk memudahkan pengelolaan produk, pengguna, pesanan, dan transaksi pembayaran secara digital. Sistem ini akan diimplementasikan sebagai REST API menggunakan Node.js dan Express, serta menggunakan MySQL sebagai basis data. Selain itu, API ini akan terintegrasi dengan API publik untuk memperkaya informasi produk, dan menggunakan JWT untuk autentikasi dan otorisasi.

2. Desain Database

Database terdiri dari 4 tabel utama yang saling berelasi:

1. Tabel Users

Tabel ini menyimpan informasi tentang pengguna yang mendaftar di toko.

| Kolom | Tipe Data | Deskripsi |
|------------|--------------|-------------------------------|
| user_id | INT (PK) | ID unik untuk setiap pengguna |
| username | VARCHAR(50) | Nama pengguna |
| email | VARCHAR(100) | Alamat email pengguna |
| password | VARCHAR(255) | Kata sandi (hashed) |
| created_at | DATETIME | Waktu pendaftaran pengguna |
| updated_at | DATETIME | Waktu terakhir diperbarui |

2. Tabel Products

Tabel ini menyimpan informasi tentang produk yang tersedia di toko.

| Kolom | Tipe Data | Deskripsi |
|-------------|---------------|-----------------------------|
| product_id | INT (PK) | ID unik untuk setiap produk |
| name | VARCHAR(100) | Nama produk |
| description | TEXT | Deskripsi produk |
| price | DECIMAL(10,2) | Harga produk |
| stock | INT | Jumlah stok yang tersedia |
| created_at | DATETIME | Waktu penambahan produk |
| updated_at | DATETIME | Waktu terakhir diperbarui |

3. Tabel Orders

Tabel ini menyimpan catatan pemesanan produk oleh pengguna.

| Kolom | Tipe Data | Deskripsi |
|--------------|---------------|---|
| order_id | INT (PK) | ID unik untuk setiap pesanan |
| user_id | INT (FK) | ID pengguna yang melakukan pemesanan (relasi ke Users) |
| order_date | DATETIME | Tanggal dan waktu pemesanan |
| total_amount | DECIMAL(10,2) | Total harga dari semua produk dalam pesanan |
| status | VARCHAR(20) | Status pesanan (misalnya: pending, completed, canceled) |

4. Tabel Payments

Tabel ini menyimpan informasi transaksi pembayaran yang dilakukan oleh pengguna.

| Kolom | Tipe Data | Deskripsi |
|----------------|---------------|---|
| payment_id | INT (PK) | ID unik untuk setiap pembayaran |
| order_id | INT (FK) | ID pesanan yang dibayar (relasi ke Orders) |
| payment_date | DATETIME | Tanggal dan waktu pembayaran |
| amount | DECIMAL(10,2) | Jumlah yang dibayarkan |
| payment_method | VARCHAR(50) | Metode pembayaran (misalnya: kartu kredit, transfer bank) |
| status | VARCHAR(20) | Status pembayaran (misalnya: successful, failed, pending) |

Relasi Antar Tabel

- **Users ke Orders:** Satu pengguna dapat memiliki banyak pesanan. Relasi ini adalah satu ke banyak.
- **Products ke Orders:** Meskipun tidak ditampilkan langsung dalam tabel Orders, biasanya kita akan memiliki tabel tambahan seperti Order_Items untuk menyimpan detail setiap item dalam pesanan.
- **Orders ke Payments:** Setiap pesanan dapat memiliki satu atau lebih metode pembayaran. Relasi ini adalah satu ke satu atau satu ke banyak tergantung pada kebutuhan sistem.

3. Perancangan Endpoint

Perancangan Endpoint API untuk Toko Online

Berikut adalah perancangan endpoint API untuk sistem toko online yang mencakup berbagai operasi yang dapat dilakukan pada tabel Users, Products, Orders, dan Payments. Dalam perancangan ini, kami akan merinci endpoint yang diperlukan serta metode HTTP yang digunakan.

1. Endpoint untuk Users

| Metode HTTP | Endpoint | Deskripsi |
|-------------|------------|---|
| POST | /register | Mendaftarkan pengguna baru. |
| POST | /login | Login pengguna. |
| GET | /users | Mendapatkan daftar semua pengguna. |
| GET | /users/:id | Mendapatkan detail pengguna berdasarkan ID. |
| PUT | /users/:id | Memperbarui data pengguna berdasarkan ID. |
| DELETE | /users/:id | Menghapus pengguna berdasarkan ID. |

2. Endpoint untuk Products

| Metode HTTP | Endpoint | Deskripsi |
|-------------|---------------|---|
| POST | /products | Menambahkan produk baru. |
| GET | /products | Mendapatkan daftar semua produk. |
| GET | /products/:id | Mendapatkan detail produk berdasarkan ID. |
| PUT | /products/:id | Memperbarui data produk berdasarkan ID. |
| DELETE | /products/:id | Menghapus produk berdasarkan ID. |

3. Endpoint untuk Orders

| Metode HTTP | Endpoint | Deskripsi |
|-------------|-------------|--|
| POST | /orders | Membuat pesanan baru. |
| GET | /orders | Mendapatkan daftar semua pesanan. |
| GET | /orders/:id | Mendapatkan detail pesanan berdasarkan ID. |
| PUT | /orders/:id | Memperbarui status pesanan berdasarkan ID. |
| DELETE | /orders/:id | Menghapus pesanan berdasarkan ID. |

4. Endpoint untuk Payments

| Metode HTTP | Endpoint | Deskripsi |
|-------------|---------------|---|
| POST | /payments | Membuat transaksi pembayaran baru. |
| GET | /payments | Mendapatkan daftar semua transaksi pembayaran. |
| GET | /payments/:id | Mendapatkan detail transaksi pembayaran berdasarkan ID. |
| PUT | /payments/:id | Memperbarui status transaksi pembayaran berdasarkan ID. |
| DELETE | /payments/:id | Menghapus transaksi pembayaran berdasarkan ID. |

5. Endpoint Tambahan (Opsional)

Untuk meningkatkan fungsionalitas API, berikut adalah beberapa endpoint tambahan yang bisa dipertimbangkan:

Untuk Users

- **POST** /users/logout: Logout pengguna.
- **GET** /users/profile: Mendapatkan profil pengguna yang sedang login.

Untuk Products

- **GET** /products/search: Mencari produk berdasarkan nama atau kategori.

Untuk Orders

- **GET /users/:id/orders:** Mendapatkan daftar pesanan untuk pengguna tertentu.

Untuk Payments

- **GET /orders/:id/payments:** Mendapatkan daftar pembayaran terkait dengan pesanan tertentu.

4. Integrasi API Publik

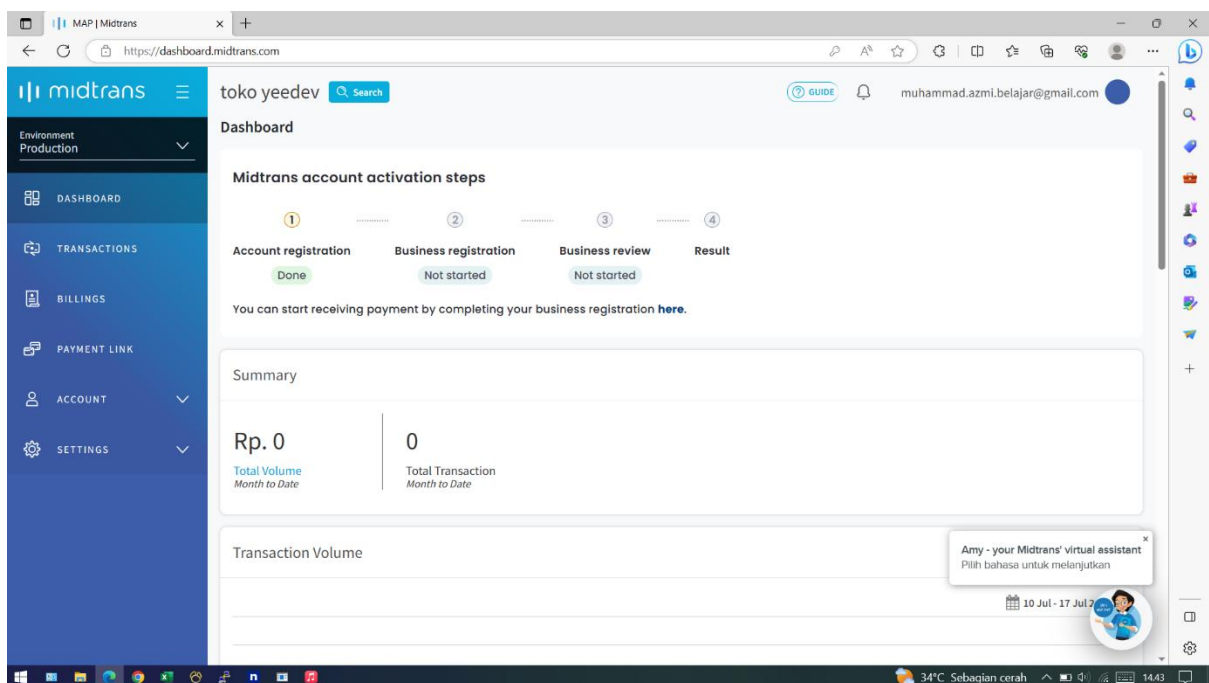
API ini akan terintegrasi dengan API pembayaran pihak ketiga seperti Midtrans atau Stripe untuk proses transaksi online.

Registrasi Akun Payment Gateway Midtrans

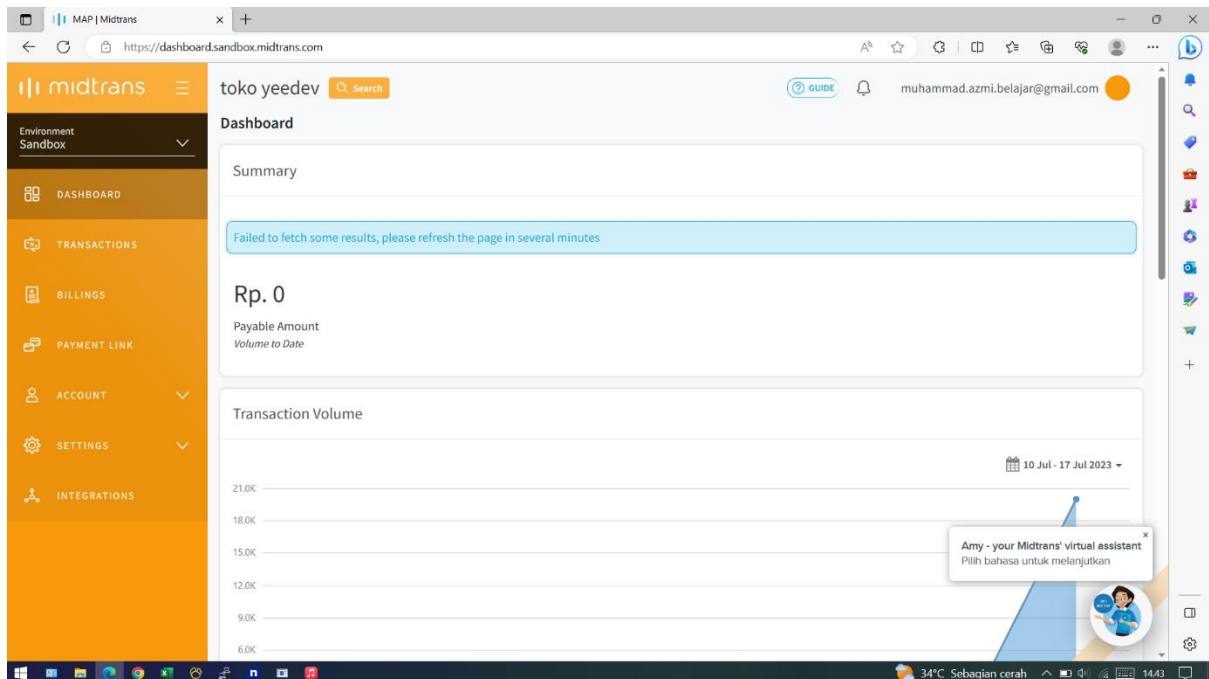
Sebelum melakukan integrasi dengan Midtrans, kita harus melakukan registrasi akun terlebih dahulu pada website midtrans.com untuk mendapatkan API Key (*client key* dan *server key*) yang akan digunakan pada aplikasi Node JS.

Bukal website midtrans.com dan lakukan registrasi dengan mengisi data-data yang diperlukan.

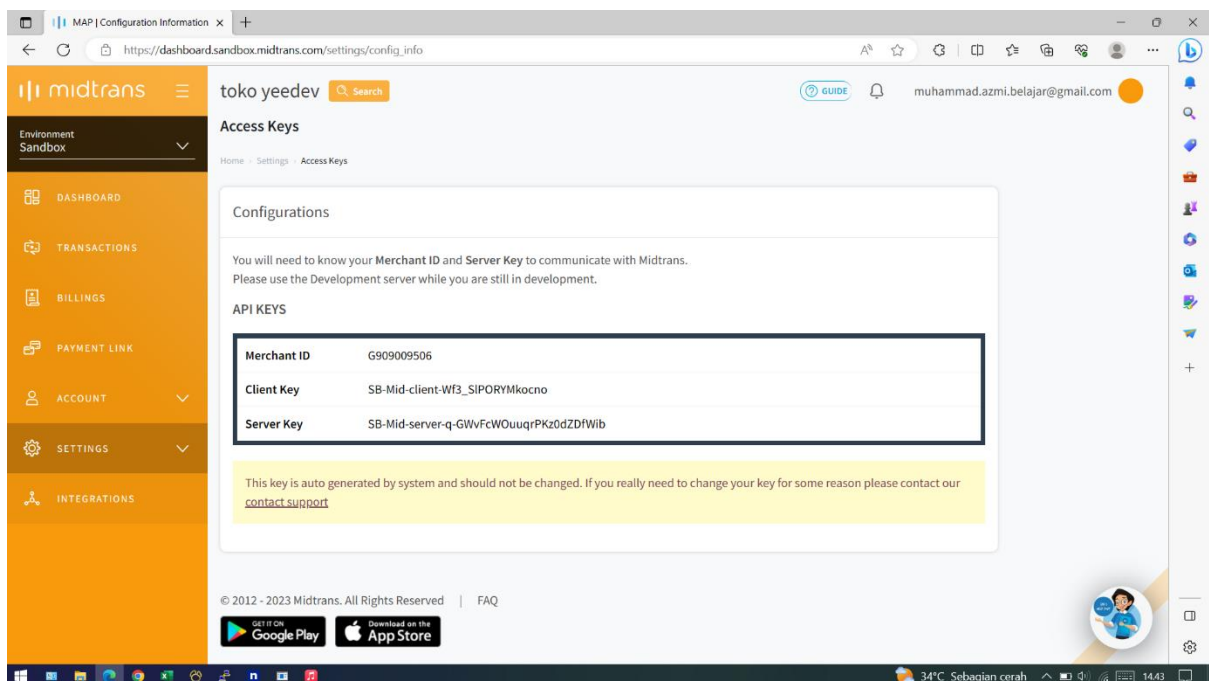
Jika sudah punya akun Midtrans, kita bisa langsung saja *login* ke website Midtrans dan akan tampil halaman seperti berikut ini.



Terlihat pada halaman *dashboard* bahwa *environment* adalah *production*, ubahlah menjadi mode *sandbox* karena aplikasi kita masih dalam percobaan.



Pada halaman *environment sandbox*, bukalah menu **Settings > Access Keys**, disini akan menampilkan API key yang akan dipakai pada aplikasi Node JS kita nanti.



Setup Project Node JS

Buka aplikasi *code editor*, pada contoh ini penulis menggunakan aplikasi VS. Code, kamu bisa menggunakan aplikasi *code editor* manapun yang ingin kamu gunakan.

Buatlah sebuah folder sebagai tempat proyek kita, lalu buatlah sebuah file bernama “package.json” dan copy-kan kode di bawah ini kedalamnya.

```
{
  "name": "03-midtrans",
  "version": "1.0.0",
  "description": "belajar payment gateway midtrans",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1",
    "start": "nodemon index.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^10.0.0",
    "express": "^4.17.1",
    "helmet": "^5.1.1",
    "midtrans-client": "^1.3.1",
    "mysql": "^2.18.1",
    "mysql2": "^2.2.5",
    "npm": "^7.19.0",
    "sequelize": "^6.3.5"
  },
  "devDependencies": {
    "morgan": "^1.10.0",
    "nodemon": "^2.0.19"
  }
}
```

Pada *code* di atas dapat kita lihat ada beberapa *dependencies* yang kita perlukan dalam project ini, misalnya “midtrans-client” yang berisi fungsi-fungsi yang diperlukan untuk akses API Midtrans.

Terdapat juga “mysql” dan “sequelize” untuk melakukan operasi CRUD ke database yang akan kita buat nantinya.

Sekarang kita membuat sebuah file bernama “index.js”, lalu *copy* dan *paste* kode dibawah ini ke file tersebut.

```
1.  const express = require("express");
2.  const app = express();
3.  const PORT = 5000;
4.
5.  app.use(express.json());
6.
7.  //routes
8.  app.get("/", (req, res) => {
9.    res.status(200).json({ message: "Welcome to Yeedev Shop API (midtrans)" });
10. });
11.
12. app.use((req, res) => {
13.   return res.status(404).send('404 Not Found !');
14. });
15.
16. // Server
17. app.listen(PORT, () => {
18.   console.log(`Server is running on port ${PORT}.`);
19. });
```

Buka aplikasi Terminal di *code editor*, pastikan kamu berada pada *directory* atau folder utama dari *project* yang kita buat.

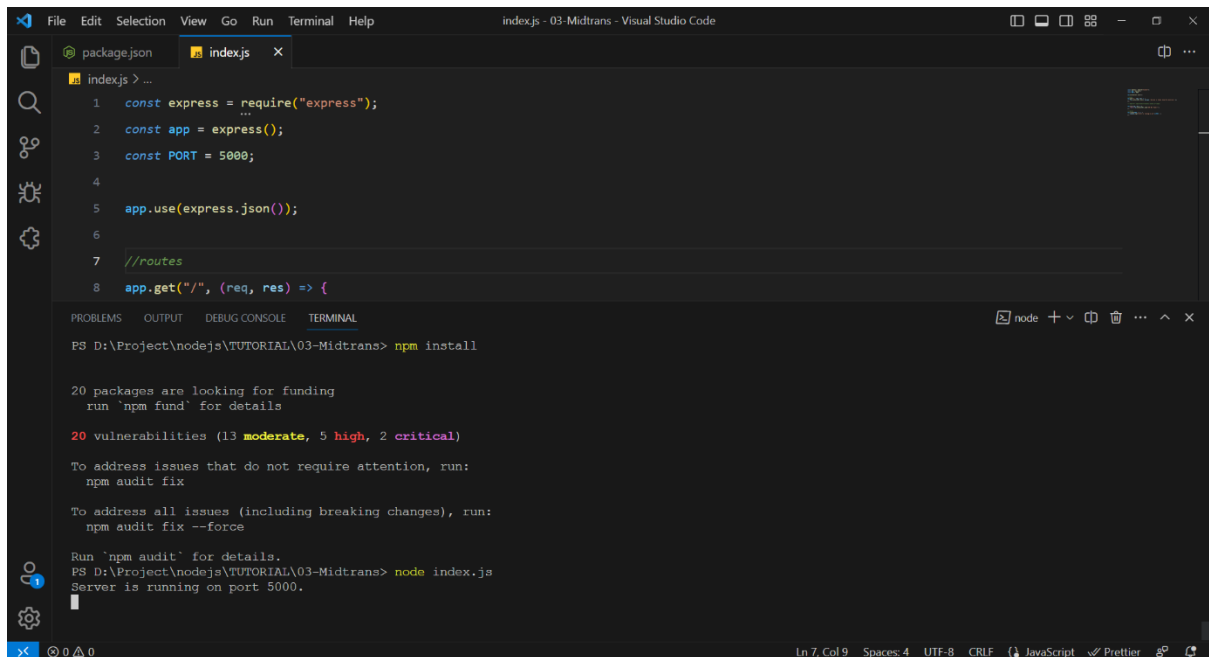
Kemudian ketikkan perintah ‘npm install’ dan enter. Tunggu beberapa saat hingga proses instalasi selesai dijalankan.

```
1.  npm install
```

Sekarang kita coba menjalankan file “index.js” dengan cara ketik ‘node index.js’ dan enter.

```
1.  node index.js
```


Jika berhasil akan muncul hasil seperti gambar berikut:



```
index.js > ...
1  const express = require("express");
2  const app = express();
3  const PORT = 5000;
4
5  app.use(express.json());
6
7  //routes
8  app.get("/", (req, res) => {

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS D:\Project\nodejs\TUTORIAL\03-Midtrans> npm install

20 packages are looking for funding
  run `npm fund` for details

20 vulnerabilities (13 moderate, 5 high, 2 critical)

To address issues that do not require attention, run:
  npm audit fix

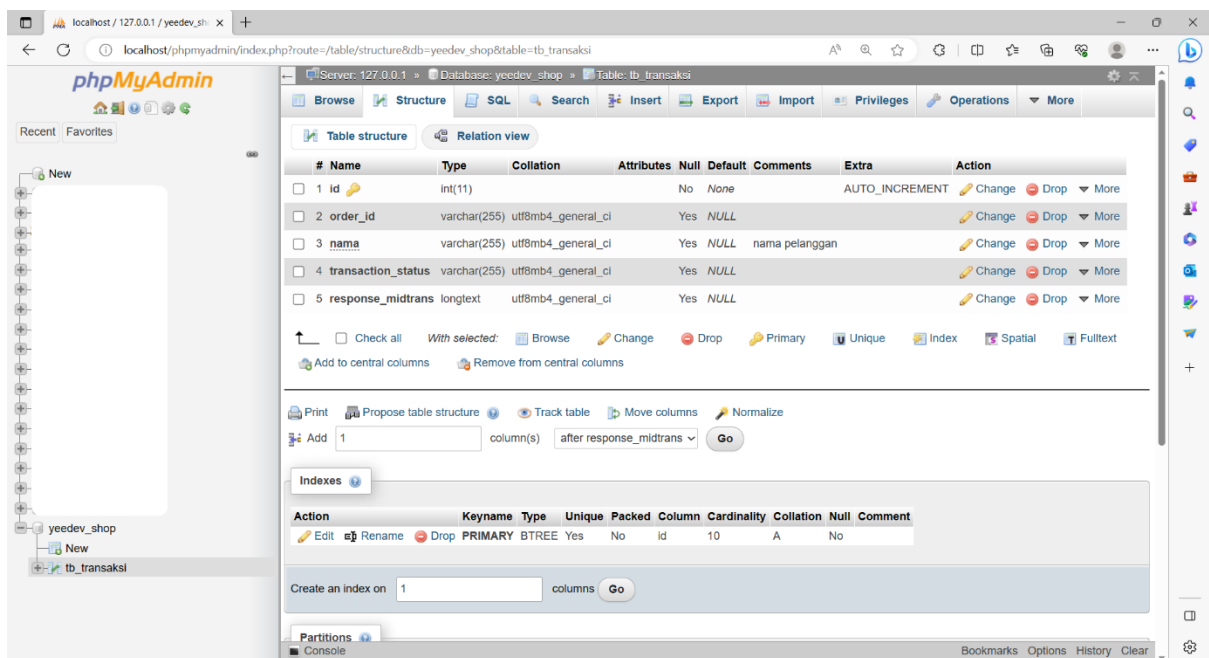
To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
PS D:\Project\nodejs\TUTORIAL\03-Midtrans> node index.js
Server is running on port 5000.
```

Sekarang kita telah berhasil melakukan setup awal project Node JS. Selanjutnya kita akan membuat sebuah *database* MySQL untuk menyimpan hasil *response* dari API Midtrans.

Membuat Database Menggunakan MySQL

Kemudian buat sebuah *database* MySQL bernama “yeedev_shop” dan selanjutnya buat tabel dengan nama “tb_transaksi”. Ikutilah penamaan variabel dan tipe data sesuai dengan gambar berikut:



abel transaksi ini berfungsi untuk menyimpan data hasil yang dikembalikan setelah melakukan pemanggilan ke API Midtrans.

Keterangan:

- id: id data
- order_id: adalah data order_id yang didapat dari response midtrans
- nama: nama pelanggan
- transaction_status: adalah data transaction_status yang didapat dari response midtrans
- response_midtrans: menyimpan seluruh respon data yang didapat dari midtrans dalam bentuk json

Membuat Model Transaksi di Node JS

Pada aplikasi *code editor*, buatlah sebuah folder bernama “model”, lalu di dalamnya buatlah sebuah file bernama “transaksi.model.js”. *Copy* dan *paste code* di bawah ini kedalam file tersebut:

```

1.  module.exports = (sequelize, Sequelize) => {
2.      const Transaksi = sequelize.define('tb_transaksi', {
3.          id: {
4.              type: Sequelize.BIGINT,
5.              primaryKey: true,
6.              autoIncrement: true,
7.          },
8.          order_id: {
9.              type: Sequelize.STRING,
10.         },
11.         nama: {
12.             type: Sequelize.STRING,
13.         },
14.         response_midtrans: {
15.             type: Sequelize.TEXT('long'),
16.         },
17.         transaction_status: {
18.             type: Sequelize.STRING,
19.         },
20.     },
21.     {
22.         freezeTableName: true,
23.         timestamps: false,
24.     });
25.
26.     return Transaksi;
27. }
```

Pastikan nama dan tipe data sesuai dengan tabel yang telah kita buat pada database MySQL. Lalu buatlah sebuah file lagi pada folder model dengan nama “index.js”.

Copy dan paste kode dibawah ini ke file tersebut:

```

1.  const config = require("../config/db.config.js");
2.  const Sequelize = require("sequelize");
3.  const db = {};
4.
5.  const sequelize = new Sequelize(config.DB, config.USER, config.PASSWORD, {
6.    host: config.HOST,
7.    port: config.PORT,
8.    dialect: config.dialect,
9.    operatorsAliases: config.operatorsAliases,
10.    dialectOptions: {
11.      socketPath: config.dialectOptions.socketPath,
12.      supportBigNumbers: config.dialectOptions.supportBigNumbers,
13.      bigNumberStrings: config.dialectOptions.bigNumberStrings,
14.      dateStrings: true,
15.      typeCast: true
16.    },
17.    define: {
18.      underscored: true, // use underscore in table name
19.    },
20.    timezone: '+07:00',
21.    socketPath: config.socketPath,
22.    pool: {
23.      max: config.pool.max,
24.      min: config.pool.min,
25.      acquire: config.pool.acquire,
26.      idle: config.pool.idle,
27.    },
28.  });
29.
30.  db.Sequelize = Sequelize;
31.  db.sequelize = sequelize;
32.  const Op = db.Sequelize.Op;
33.
34.  db.transaksi = require("../transaksi.model.js")(sequelize, Sequelize);
35.
36.  module.exports = db;
37.

```

Membuat File Config Database dan Config API Midtrans

Kemudian buat folder dengan nama "config". Tambahkan file dengan nama "db.config.js", kemudian *copy* dan *paste* kode dibawah ke file tersebut.

```

1.  module.exports = {
2.    HOST: process.env.DB_HOST || "localhost",
3.    USER: process.env.DB_USER || "root",
4.    PASSWORD: process.env.DB_PASSWORD || "",
5.    DB: process.env.DB_NAME || "yeedev_shop",
6.    PORT: process.env.DB_PORT || 3306,
7.    dialect: "mysql",
8.    dialectOptions: {
9.      supportBigNumbers: true,
10.     bigNumberStrings: true,
11.    },
12.    operatorsAliases: 0,
13.    pool: {
14.      max: 5,
15.      min: 0,
16.      acquire: 30000,
17.      idle: 10000,
18.    },
19.  };

```

Pastikan “Host”, “User”, “Password” dan “Name” telah sesuai dengan *database* yang telah kita buat sebelumnya.

Jika sudah, sekarang buat lagi file pada folder “config” dengan nama “apiMidtrans.config.js”, lalu *copy* dan *paste code* dibawah ini ke file tersebut:

```
1. const midtransClient = require('midtrans-client');
2.
3. exports.coreApi = new midtransClient.CoreApi({
4.   isProduction: false,
5.   serverKey: 'SERVER-KEY-ANDA', // // sesuaikan dengan akun midtrans anda
6.   clientKey: 'CLIENT-KEY-ANDA' // sesuaikan dengan akun midtrans anda
7. });
```

Jangan lupa mengisi API key midtrans yang telah didapat dari Midtrans sebelumnya.

Membuat Route Transaksi di Node JS

Sekarang kita akan membuat “route”, tambahkan kode dibawah ini kedalam file “index.js” kita yang ada di root direktori *project*.

```
1. const express = require("express");
2. const app = express();
3. const PORT = 5000;
4.
5. app.use(express.json());
6.
7. //routes
8. app.get("/", (req, res) => {
9.   res.status(200).json({ message: "Welcome to Yeedev Shop API (midtrans)" });
10. });
11.
12. require("../routes/transaksi.routes.js") (app); // baris kode yang kita tambahkan
13.
14. app.use((req, res) => {
15.   return res.status(404).send('404 Not Found !');
16. });
17.
18. // Server
19. app.listen(PORT, () => {
20.   console.log(`Server is running on port ${PORT}.`);
21. });
```

Jika sudah, buatlah sebuah folder bernama “routes”, lalu didalamnya buat file dengan nama “transaksi.routes.js”, kemudian *copy* dan *paste* kode dibawah ini ke file tersebut:

```
1. module.exports = (app) => {
2.   const controller = require("../controllers/transaksi.controller.js");
3.
4.   app.use(function (req, res, next) {
5.     res.header(
6.       "Access-Control-Allow-Headers",
7.       "x-access-token, Origin, Content-Type, Accept, Authorization"
8.     );
9.     next();
10.  });
11.
12.   app.post('/midtrans-transaction/charge', controller.midtransChargeTransaction);
13.
14. };
```

Membuat Controller Charge Transaksi di Node JS

Sekarang buatlah sebuah folder bernama “controller” pada *root* direktori project, lalu buat sebuah file dengan nama “transaksi.controller.js”, kemudian *copy* dan *paste* kode di bawah ini ke file tersebut:

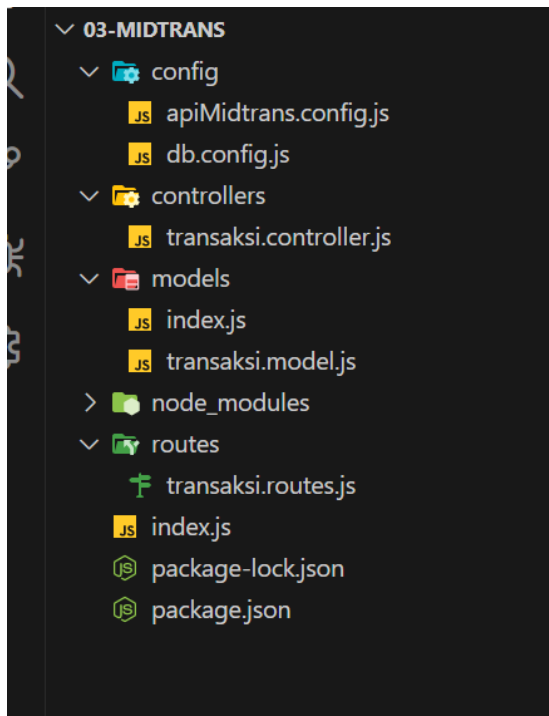
```
1.  const db = require('../models/index.js');
2.  const Transaksi = db.transaksi;
3.  const { coreApi } = require('../config/apiMidtrans.config.js');
4.
5.  exports.midtransChargeTransaction = async (req, res) => {
6.
7.      if (Object.keys(req.body).length === 0) {
8.          return res.status(400).send({
9.              success: false,
10.             message: "Content can not be empty!",
11.         });
12.     }
13.
14.     try {
15.         // lakukan charge transaksi ke server midtrans sesuai request
16.         coreApi.charge(req.body).then((chargeResponse) => {
17.             // console.log(chargeResponse);
18.             Transaksi.create({
19.                 order_id: chargeResponse.order_id,
20.                 nama: req.body.nama,
21.                 transaction_status: chargeResponse.transaction_status,
22.                 response_midtrans: JSON.stringify(chargeResponse),
23.             }).then(data => {
24.                 return res.status(201).json({ success: true, message: "Berhasil
melakukan charge transaction!", data: data });
25.             }).catch(error => {
26.                 return res.status(400).json({ success: false, message: error.message,
27.             });
28.
29.             }).catch((error) => {
30.                 return res.status(400).json({ success: false, message: error.message, });
31.             });
32.
33.         } catch (error) {
34.             return res.status(500).json({ success: false, message: error.message, });
35.         }
36.     }
```

Pada bagian “controller” ini terdapat kode untuk melakukan akses API Midtrans.

Keterangan:

- `coreApi.charge()` adalah method untuk melakukan charge transaction ke API Midtrans, di dalamnya terdapat *parameter* yang berisi *request body* yang dikirimkan dari *endpoint* sesuai dengan format pada dokumentasi Midtrans.
- Jika berhasil, akan dikembalikan *response* dari midtrans yang selanjutnya *data response* akan disimpan kedalam tabel transaksi pada *database*.

Jika semua kode telah ada, maka akan tampak struktur folder pada proyek kita seperti pada gambar berikut:



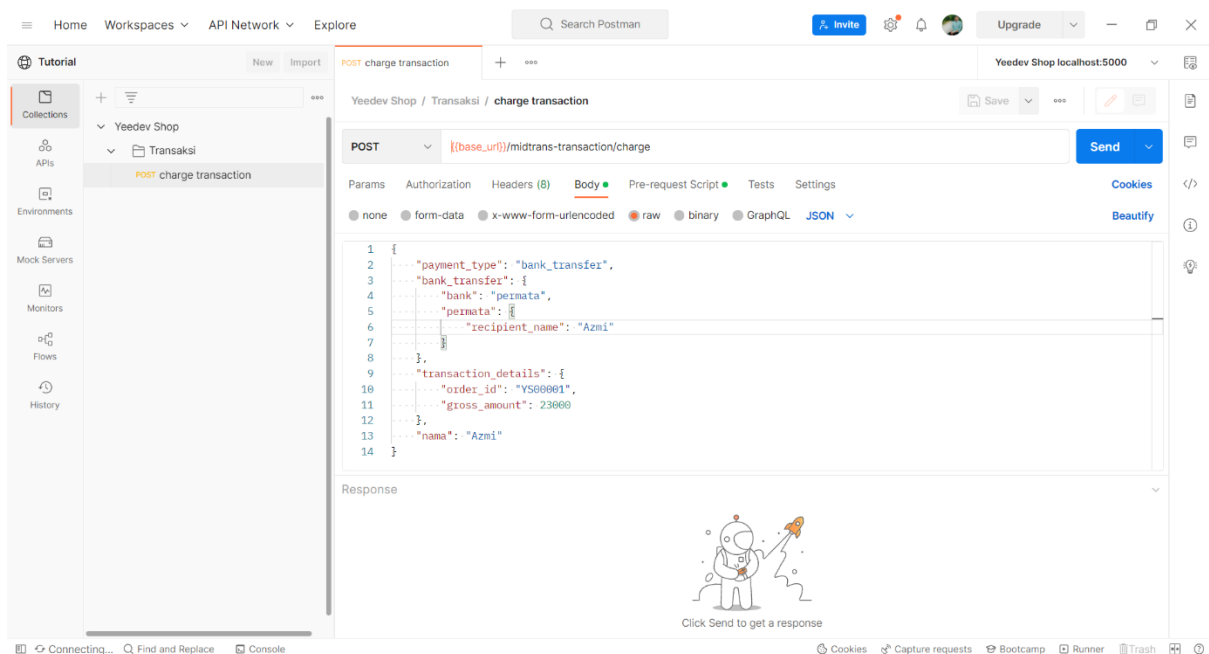
Hasil dan Uji Coba Integrasi API Node JS dengan Midtrans

Sekarang kita akan melihat hasil dan melakukan pengujian API yang telah selesai kita buat dengan menggunakan aplikasi Postman. Jalankan proyek kita dengan perintah “node index.js” pada Terminal dan tekan *enter*.

Kemudian, buka Postman.

Kemudian *fork* pada *collection* Postman agar dapat dijalankan, jangan lupa pilih *environment* dipojok kanan atas sebelum menjalankan API.

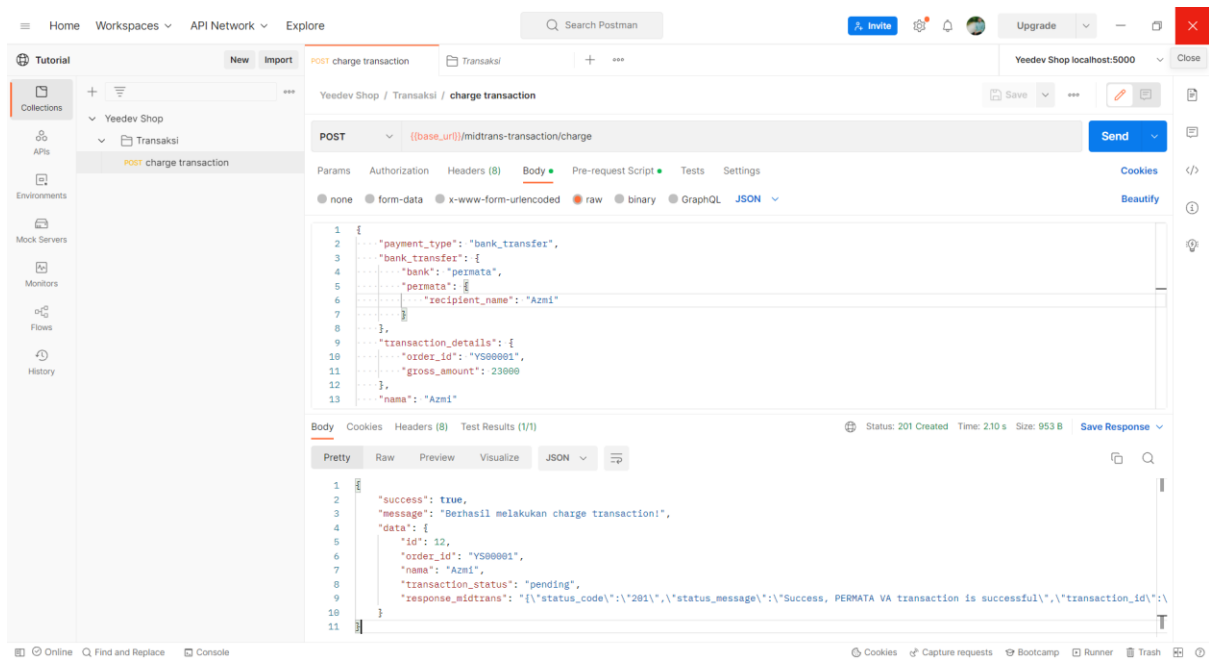
Jika sudah akan tampil seperti gambar berikut.



Keterangan:

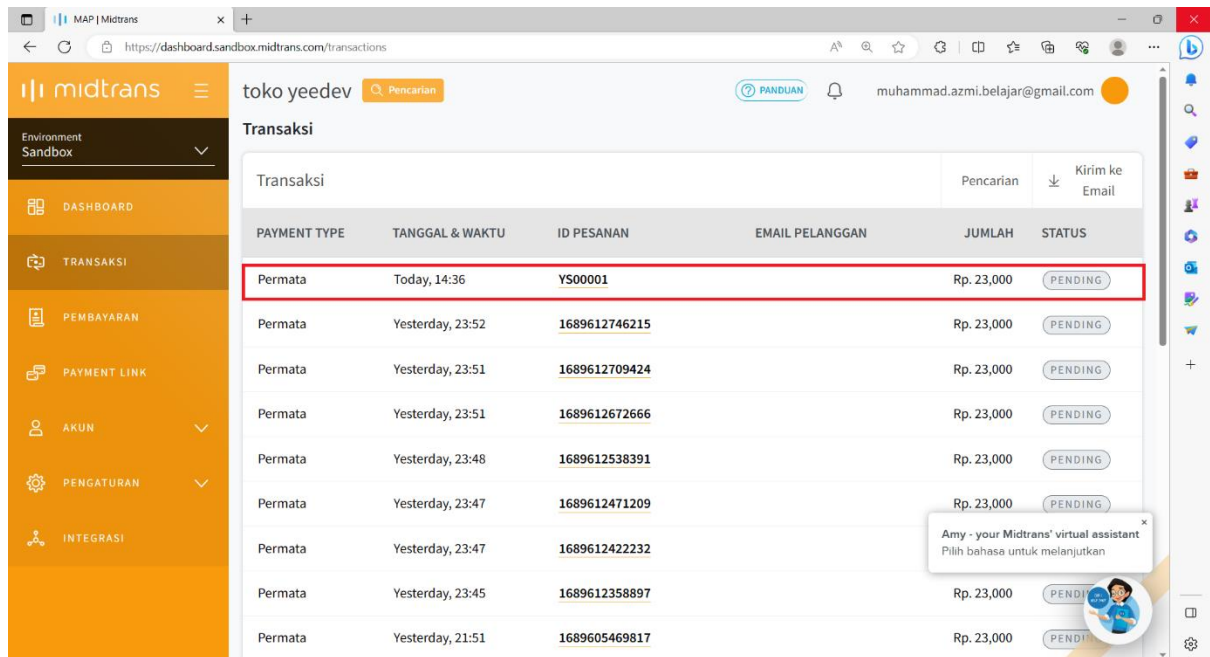
- `payment_type`: adalah tipe pembayaran yang dipilih, pada contoh ini adalah `bank_transfer`, dan dibawahnya terdapat detail terkait `bank_transfer`.
- `transaction_details`: berisi detail transaksi meliputi `order_id`, `order_id` tidak boleh sama dengan transaksi sebelumnya, karena `order_id` ini bertindak seperti primary key ketika disimpan dalam database. Lalu `gross_amount`: berisi total biaya yang dikenakan pada pelanggan.
- `nama`: nama pelanggan (optional), karena merupakan costum field.
- Disini kita hanya mencoba pembayaran dengan bank transfer, sebenarnya masih ada pembayaran lain. Format json akan menyesuaikan dengan pembayaran yang kita pilih, jadi untuk lengkapnya dapat dilihat pada dokumentasi resmi midtrans.

Jalankan API dengan cara klik tombol “Send”. Jika berhasil akan tampil seperti gambar berikut.



Sekarang kita telah berhasil melakukan *charge* transaksi, hasil *response* ini akan tersimpan pada *database* yang telah kita buat sebelumnya.

Riwayat dari transaksi ini bisa kita lihat pada *web dashboard* Midtrans di bagian Menu “Transaksi” seperti pada gambar di bawah ini:



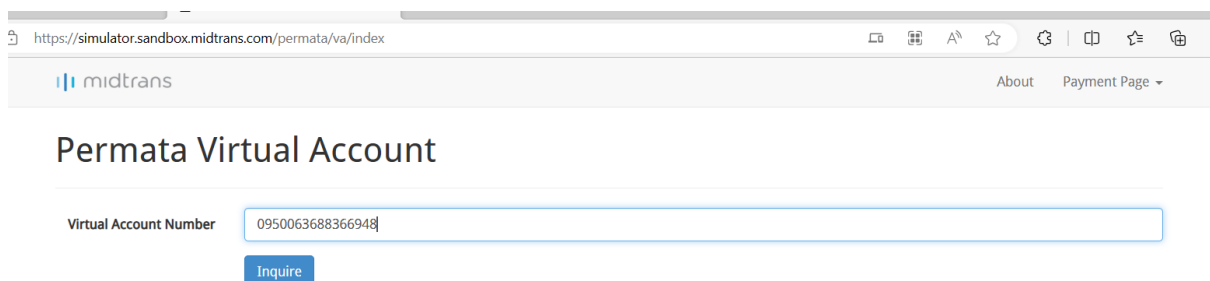
Terlihat status transaksi pada riwayat transaksi “pending”.

Sampai disini charge transaksi yang telah berhasil, tinggal menunggu pembayaran dari pelanggan, yaitu pelanggan harus transfer ke nomor virtual akun di *response* Midtrans yang disimpan tadi.

Dikarenakan ini masih pengembangan ataupun Midtrans dalam environment sanbox, kita bisa melakukan simulasi pembayaran dari simulator yang disediakan oleh Midtrans pada link berikut.

[Simulator pembayaran Midtrans \(VA Permata\)](https://simulator.sandbox.midtrans.com/permata/va/index)

Kemudian akan terbuka halaman untuk simulasi pembayaran dengan Pertama Virtual Account seperti pada gambar berikut:



Masukkan nomor virtual akun pada halaman simulator, dan klik “Inquire”. Kemudian akan muncul halaman konfirmasi seperti pada gambar berikut:

https://simulator.sandbox.midtrans.com/permata/va/inquiry



Permata Virtual Account

Total Transaksi

23000.00

(23000.00)

Atas Nama

azmi

Bayar

Kemudian klik tombol bayar, jika berhasil akan muncul gambar seperti berikut:

https://simulator.sandbox.midtrans.com/permata/va/payment



Permata Virtual Account

Transaksi Sukses

Lakukan Reversal hanya jika anda ingin membatalkan transaksi ini!

Reversal

Sekarang transaksi telah berhasil dibayar, status transaksi "Pending" sekarang berubah menjadi "Settlement" yang berarti pembayaran sudah selesai dilakukan.

The screenshot shows the Midtrans dashboard with a sidebar on the left containing navigation links: DASHBOARD, TRANSAKSI, PEMBAYARAN, PAYMENT LINK, AKUN, PENGATURAN, and INTEGRASI. The main content area is titled 'Transaksi' and displays a table of transactions. The first transaction is highlighted with a green border and a green 'SETTLEMENT' status. The other transactions are marked as 'FAILURE'.

| PAYMENT TYPE | TANGGAL & WAKTU | ID PESANAN | EMAIL PELANGGAN | JUMLAH | STATUS |
|--------------|------------------|---------------|-----------------|------------|------------|
| Permata | 18/07/2023 14:36 | YS00001 | | Rp. 23,000 | SETTLEMENT |
| Permata | 17/07/2023 23:52 | 1689612746215 | | Rp. 23,000 | FAILURE |
| Permata | 17/07/2023 23:51 | 1689612709424 | | Rp. 23,000 | FAILURE |
| Permata | 17/07/2023 23:51 | 1689612672666 | | Rp. 23,000 | FAILURE |
| Permata | 17/07/2023 23:48 | 1689612538391 | | Rp. 23,000 | FAILURE |
| Permata | 17/07/2023 23:47 | 1689612471209 | | Rp. 23,000 | FAILURE |
| Permata | 17/07/2023 23:47 | 1689612422232 | | Rp. 23,000 | FAILURE |
| Permata | 17/07/2023 23:45 | 1689612358897 | | Rp. 23,000 | FAILURE |

Sampai disini transaksi berhasil dan selesai, namun pada aplikasi kita masih belum terupdate datanya dikarenakan perubahan di Midtrans belum terkirim ke aplikasi Node JS yang telah kita buat.

5. Middleware & Validasi (JWT)

Menggunakan JWT untuk autentikasi pada endpoint yang sensitif (CRUD User, CRUD Orders, CRUD Payments). Middleware akan memverifikasi token pada setiap request yang memerlukan autentikasi.

Menggunakan JSON Web Tokens (JWT) untuk autentikasi pada endpoint yang sensitif dalam aplikasi e-commerce adalah cara yang efektif untuk melindungi data pengguna dan transaksi. Berikut adalah langkah-langkah untuk mengimplementasikan middleware dan validasi JWT dalam sistem Anda.

1. Instalasi Dependensi

Pastikan Anda memiliki Node.js dan Express terinstal. Selanjutnya, instal paket yang diperlukan:

1. Instalasi Dependensi

Pastikan Anda memiliki Node.js dan Express terinstal. Selanjutnya, instal paket yang diperlukan:

```
bash
npm install express jsonwebtoken bcrypt dotenv
```

2. Konfigurasi Lingkungan

Buat file `.env` untuk menyimpan variabel lingkungan, termasuk kunci rahasia untuk JWT:

```
text
JWT_SECRET=your_secret_key
```

3. Middleware untuk Verifikasi JWT

Buat file middleware, misalnya `middleware/authenticate.js`, yang akan memverifikasi token JWT:

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

dotenv.config();

const authenticate = (req, res, next) => {
  const token = req.header('Authorization');

  if (!token) {
    return res.status(401).json({ message: 'Authorization token is required.' });
  }

  try {
    const decoded = jwt.verify(token.replace('Bearer ', ''), process.env.JWT_SECRET);
    req.user = decoded.user;
    next();
  } catch (err) {
    return res.status(401).json({ message: 'Invalid token.' });
  }
};

module.exports = authenticate;
```

4. Menggunakan Middleware pada Endpoint Sensitif

Terapkan middleware authenticate pada endpoint yang memerlukan autentikasi, seperti CRUD untuk Users, Orders, dan Payments:

```
javascript
const express = require('express');
const authenticate = require('./middleware/authenticate');

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

// Contoh endpoint CRUD User
app.post('/users', authenticate, (req, res) => {
  // Logika untuk membuat pengguna baru
});

app.get('/users/:id', authenticate, (req, res) => {
  // Logika untuk mendapatkan detail pengguna berdasarkan ID
});

// Endpoint lain untuk Orders dan Payments juga menggunakan middleware
yang sama

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

5. Proses Login dan Pembuatan Token

Saat pengguna berhasil login, buat token JWT yang akan digunakan untuk autentikasi di permintaan berikutnya:

```
javascript
app.post('/login', async (req, res) => {
  const { username, password } = req.body;

  // Verifikasi kredensial pengguna di database
  // Jika valid:
  const token = jwt.sign({ user: { id: user.id } },
    process.env.JWT_SECRET, { expiresIn: '1h' });
  res.json({ token });
});
```

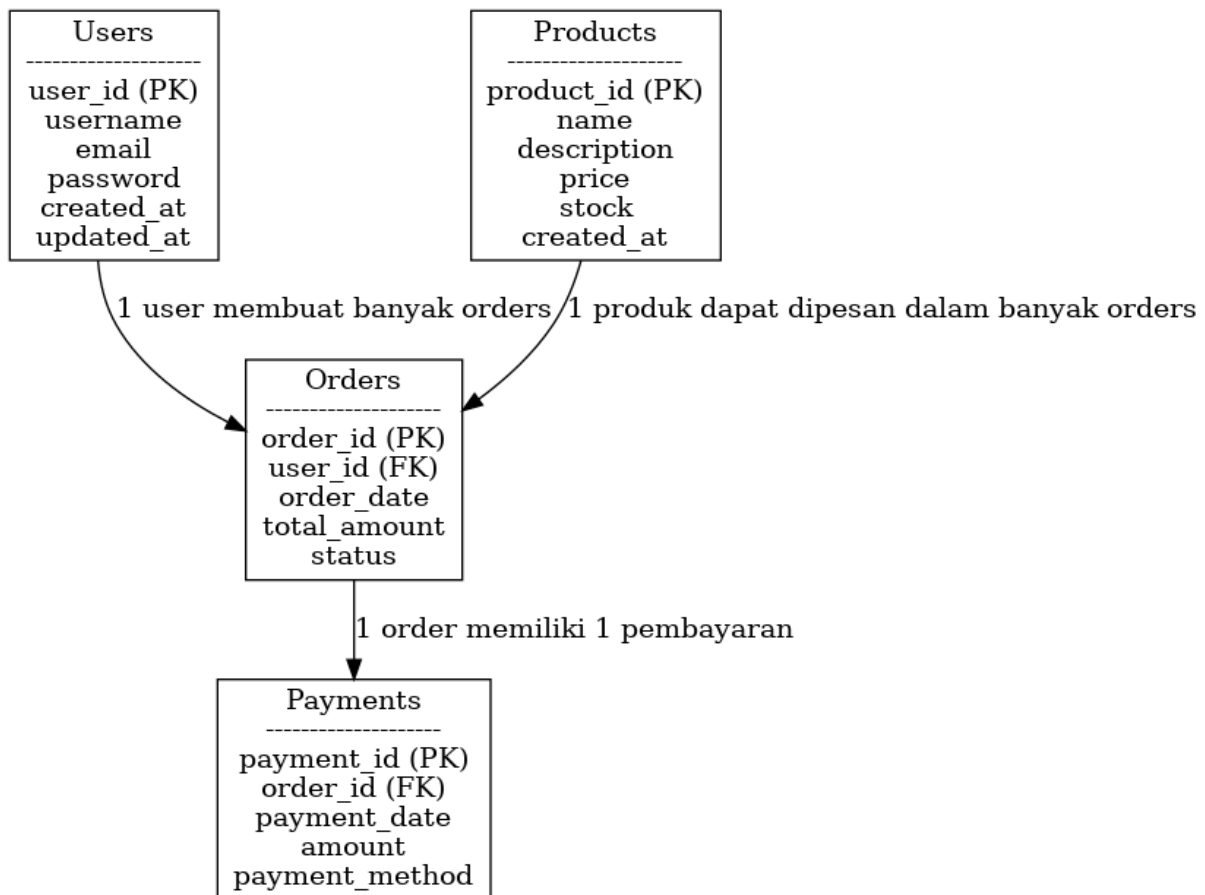
6. Pengujian Endpoint

Setelah implementasi selesai, Anda dapat menggunakan alat seperti Postman untuk menguji endpoint. Pastikan untuk mengirimkan token JWT dalam header `Authorization` saat mengakses endpoint yang dilindungi:

- **Login:** Kirim permintaan POST ke `/login` dengan kredensial.
- **Akses Endpoint Terkait:** Kirim permintaan GET ke `/users/:id` dengan header `Authorization: Bearer <token>`.

7. Tampilan Laporan

Laporan ini akan dihiasi dengan diagram ER, diagram alur API, dan contoh response dari API agar lebih menarik dan informatif.



Keterangan:

- **PK:** Primary Key
- **FK:** Foreign Key
- Relasi:
 - Satu pengguna dapat memiliki banyak pesanan.
 - Satu pesanan dapat memiliki satu atau lebih pembayaran.

2. Diagram Alur API

Diagram alur API menunjukkan bagaimana permintaan dan respons terjadi antara klien dan server. Berikut adalah contoh sederhana dari alur API untuk proses login dan pembuatan pesanan:

Contoh Diagram Alur API

text

```
[Client] --POST /login--> [Server]
      ↓                      ↓
[Token JWT] <--- 200 OK --- [Response]

[Client] --POST /orders--> [Server]
      ↓                      ↓
[Order Details] <--- 201 Created --- [Response]
```

Keterangan:

- Klien mengirimkan permintaan login ke server.
- Server memverifikasi kredensial dan mengembalikan token JWT.
- Klien menggunakan token tersebut untuk membuat pesanan baru.
- Server memproses pesanan dan mengembalikan detail pesanan yang baru dibuat.

3. Contoh Respons dari API

Berikut adalah beberapa contoh respons dari endpoint yang telah dirancang:

a. Respons untuk Login

Endpoint: `POST /login`

Contoh Permintaan:

json

```
{
  "username": "user123",
  "password": "password123"
}
```

Contoh Respons:

json

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

b. Respons untuk Mendapatkan Daftar Pengguna

Endpoint: `GET /users`

Contoh Respons:

json

```
[
  {
    "user_id": 1,
    "username": "user123",
    "email": "user123@example.com",
    "created_at": "2023-01-01T12:00:00Z"
  },
  {
    "user_id": 2,
    "username": "user456",
    "email": "user456@example.com",
    "created_at": "2023-01-02T12:00:00Z"
  }
]
```

c. Respons untuk Membuat Pesanan

Endpoint: `POST /orders`

Contoh Permintaan:

json

```
{
  "user_id": 1,
  "total_amount": 150000,
  "status": "pending"
}
```

Contoh Respons:

json

```
{
  "order_id": 101,
  "user_id": 1,
  "order_date": "2023-02-02T10:00:00Z",
  "total_amount": 150000,
  "status": "pending"
}
```