PRAKTIKUM
SISTEM BASIS DATA

Nama	Jesaya David Gamalael N P	No. Modul	6
NPM	2306161965	Tipe	Tugas Tambahan

1. Implementasi Get Transactions

   Repository

```
exports.getAllTransactions = async () => {
    try {
        const result = await db.query(`SELECT
                t.*,
                json_build_object(
                    'id', u.id,
                    'name', u.name,
                    'email', u.email,
                    'password', u.password,
                    'balance', u.balance,
                    'created_at', u.created_at
                ) as user,
                json_build_object(
                    'id', i.id,
                    'name', i.name,
                    'price', i.price,
                    'store_id', i.store_id,
                    'image_url', i.image_url,
                    'stock', i.stock,
                    'created_at', i.created_at
                ) as item
            FROM transactions t
            JOIN users u ON t.user_id = u.id
            JOIN items i ON t.item_id = i.id
            ORDER BY t.created_at DESC`);

        return result.rows;
    } catch (error) {
        console.error('Failed to get transactions', error);
```

```
            throw error;
        }
    };
```

Controller

```
    exports.getAllTransactions = async (req, res) => {
        try {
            const transactions = await
transactionRepository.getAllTransactions();
            return baseResponse(res, true, 200, 'Transactions found',
transactions);
        } catch (error) {
            console.error('Transactions not found:', error);
            return baseResponse(res, false, 500, 'Failed to retrieve
transactions', null);
        }
    };
```

Routes

```
    router.get('/', transactionController.getAllTransactions);
```

2. Uji Endpoint

Method	Endpoint	Success	Failed
GET	/transaction		

3. Modul 5 dan 6 - Improvisasi Kode (Tambahan Baru)

Tipe	Implementasi	Deskripsi

<table>
<tr>
<td rowspan="3">Error Handling (Rollback & Atomicity)</td>
<td>

```
exports.updateItem = async (id, item) => {
  try {
    await db.query('BEGIN');

    const existingItem = await db.query(
      'SELECT * FROM items WHERE id = $1',
      [id]
    );

    if (existingItem.rows.length === 0) {
      await db.query('ROLLBACK');
      throw new Error('Item not found');
    }

    if (item.store_id && item.store_id !== existingItem.rows[0].store_id) {
      const storeExists = await db.query(
        'SELECT id FROM stores WHERE id = $1',
        [item.store_id]
      );

      if (storeExists.rows.length === 0) {
        await db.query('ROLLBACK');
        throw new Error('Store not found');
      }
    }

    const updatedItem = {
      name: item.name || existingItem.rows[0].name,
      price: item.price !== undefined ? parseFloat(item.price) : existingItem.rows[0].price,
      stock: item.stock !== undefined ? parseInt(item.stock, 10) : existingItem.rows[0].stock,
      store_id: item.store_id || existingItem.rows[0].store_id,
      image_url: item.image_url || existingItem.rows[0].image_url
    };

    if (updatedItem.price < 0) {
      await db.query('ROLLBACK');
      throw new Error('Price negatif!');
    }

    if (updatedItem.stock < 0) {
      await db.query('ROLLBACK');
      throw new Error('Stock negatif!');
    }
```

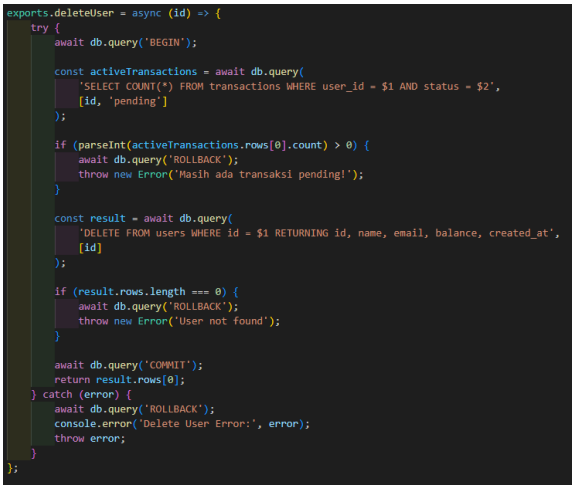
</td>
<td>Rollback diterapkan ke fungsi updateitem, sehingga penambahan dan perubahan data item jadi atomik dan tidak rentan kerusakan tengah operasi</td>
</tr>
<tr>
<td>

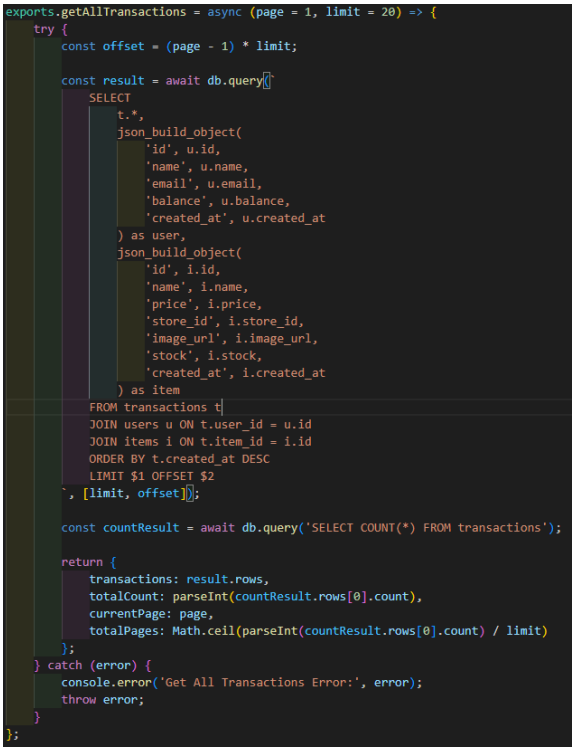
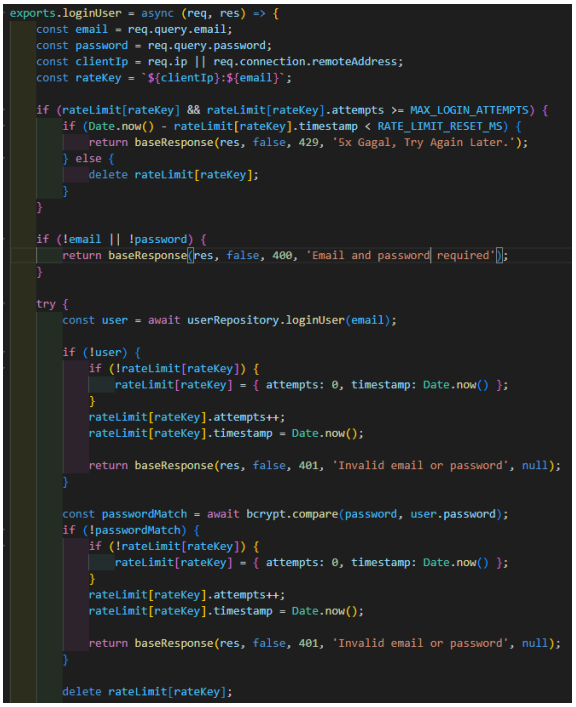
```
exports.topUpUserBalance = async (id, amount) => {
  try {
    await db.query('BEGIN');

    const currentUser = await db.query(
      'SELECT * FROM users WHERE id = $1 FOR UPDATE', // Locking
      [id]
    );

    if (currentUser.rows.length === 0) {
      await db.query('ROLLBACK');
      throw new Error('User not found');
    }

    const currentBalance = currentUser.rows[0].balance || 0;
    const newBalance = currentBalance + parseInt(amount, 10);

    // Log
    await db.query(
      'INSERT INTO balance_logs (user_id, previous_balance, amount, new_balance, action) VALUES ($1, $2, $3, $4, $5)',
      [id, currentBalance, amount, newBalance, 'top_up']
    );

    const result = await db.query(
      'UPDATE users SET balance = $1 WHERE id = $2 RETURNING id, name, email, balance, created_at',
      [newBalance, id]
    );

    await db.query('COMMIT');
    return result.rows[0];
  } catch (error) {
    await db.query('ROLLBACK');
    console.error('Top Up Error:', error);
    throw error;
  }
};
```

</td>
<td>Sistem rollback juga diterapkan ke proses top up balance. Proses transaksi juga dilock agar hanya bisa menjalankan 1 transaksi per waktu dan tiap transaksi dilog dalam database</td>
</tr>
<tr>
<td>

```
exports.registerUser = async (user) => {
  try {
    await db.query('BEGIN'); // Cek jika email ada
    const emailCheck = await db.query('SELECT id FROM users WHERE email = $1', [user.email]);
    if (emailCheck.rows.length > 0) {
      await db.query('ROLLBACK');
      throw new Error('Email already exists');
    }

    const res = await db.query(
      'INSERT INTO users (name, email, password, balance) VALUES ($1, $2, $3, $4) RETURNING id, name, email, balance, created_at',
      [user.name, user.email, user.password, 0]
    );
    await db.query('COMMIT');
    return res.rows[0];
  } catch (error) {
    await db.query('ROLLBACK');
    console.error('Registration Error:', error);
    throw error;
  }
};
```

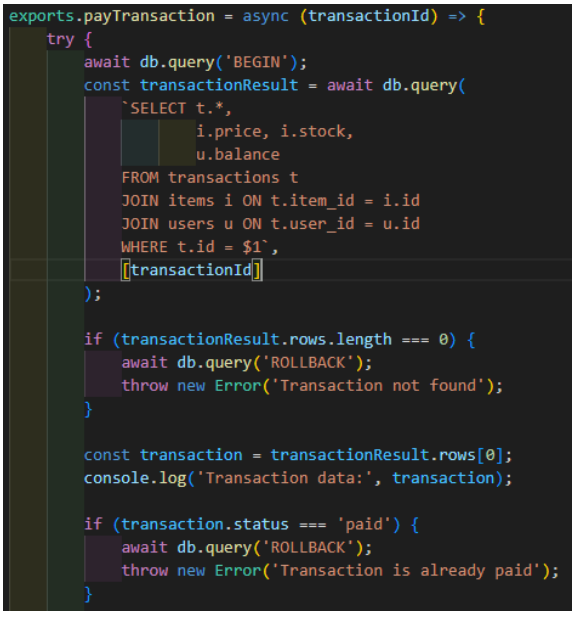
</td>
<td>Register user menggunakan tambahan sistem rollback yang sama dengan lainnya</td>
</tr>
</table>

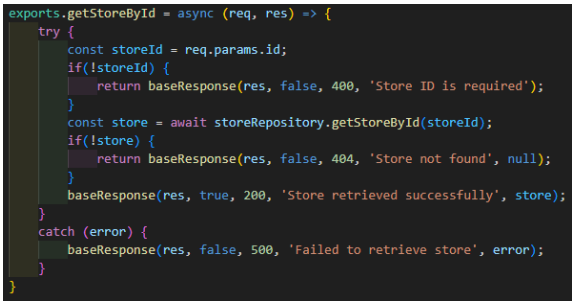
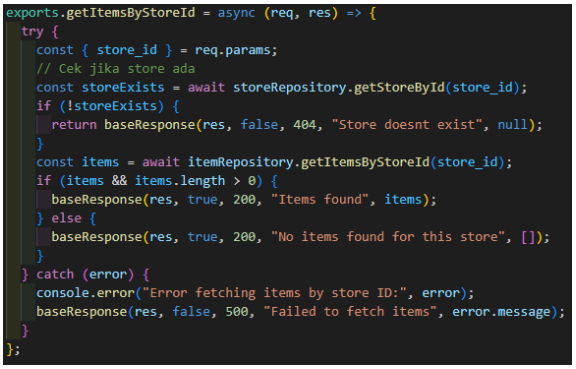
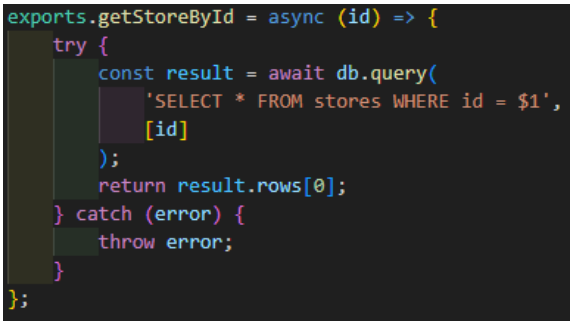
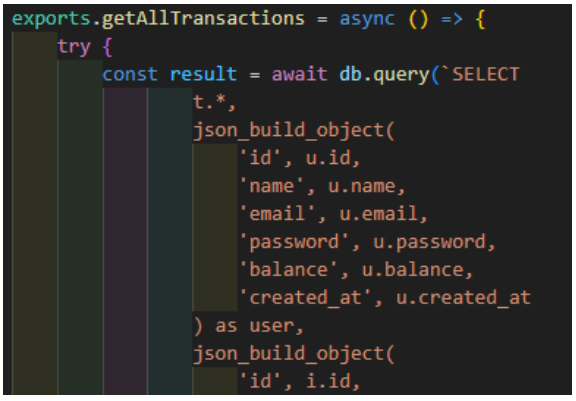
		UpdateUser
		DeleteUser (+ mengecek apabila ada transaksi yang masih pending sebelum delete)
Keamanan (Password Hide)		UpdateUser (repository) tidak akan lagi mengembalikan password dalam responsnya agar lebih terjaga

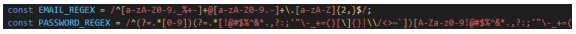
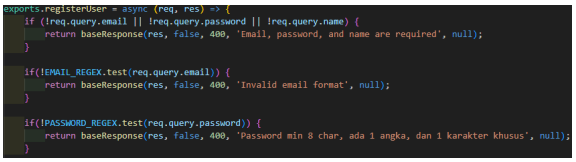
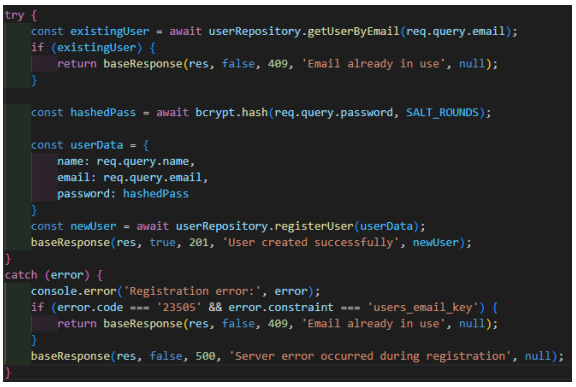
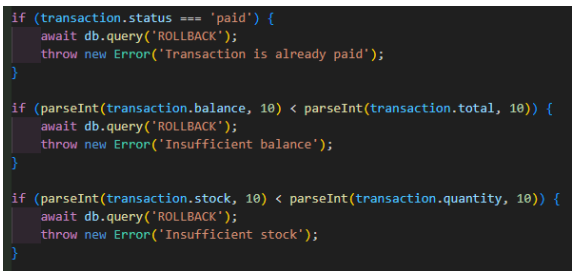
Skalabilitas (Pagination)		Respons all transactions dijadikan multiple pages (banyak halaman) agar lebih mudah diolah dan dicek saat skala sudah besar)
Keamanan (Login Limit)		Login dibatasi ke 5x kegagalan sebelum dilock 15 menit untuk mencegah serangan brute force menggunakan date dan rate key.

Error Handling (Transaksi Overall)		Transaksi diberikan sistem rollback seperti user yang lain,dan juga diberikan check agar balance dan stock tidak negatif.

4. Modul 5 dan 6 - Improvisasi Kode (List Lama)

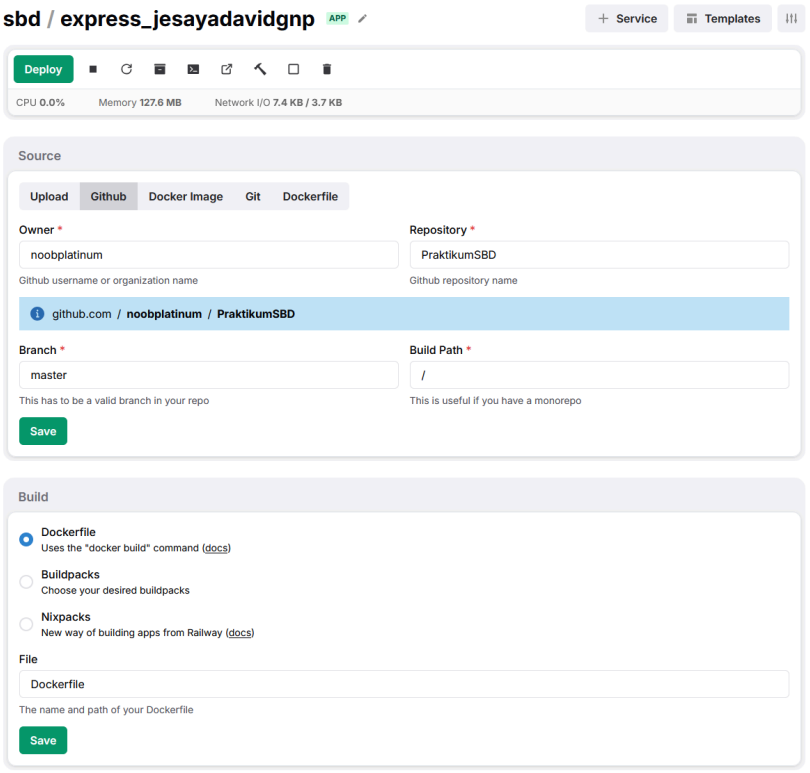
Tipe	Implementasi	Deskripsi
Error Handling (Rollback & Atomicity)		Sistem rollback diterapkan supaya atomik, dan jika ada kesalahan tengah transaksi, operasi dibatalkan tanpa menyebabkan kerugian bagi pihak manapun

Error Handling (Nested Checking + Error Message)		Pengecekan error dilakukan dengan berlapis supaya mengcover semua kemungkinan error (misalnya store tidak ada, store ID kosong, dan store ID invalid, sekaligus)
Error Handling (Validasi Store)		Mengecek jika store memang ada dalam database dengan ID yang tepat sebelum membuat item baru di dalamnya, agar tidak ada item yang terdaftar tanpa store asal
Skalabilitas (Async/Wait)		Menggunakan async dan await (di kebanyakan fungsi) agar proses tidak saling memblokir dan dapat berjalan dengan lancar bersama
Skalabilitas (Tipe Objek)		Menggunakan jenis object json_build_object dalam mendapatkan data relasi query agar request ke database lebih cepat dan efisien

Skalabilitas (Hapus Image)		Saat item dihapus, gambar dengan URLnya dalam storage Cloudinary juga dihapus, agar ruang jadi lebih ringan dan tidak memenuhi DB
Keamanan (Regex)		Menggunakan regex untuk validasi email yang digunakan pengguna (harus valid) dan mencegah serangan via input string di query
Keamanan (Validasi Input)		Kecocokan password dan email dengan regex + data dalam database dicek untuk memastikan keamanan data akun
Keamanan (Password Hashing)		Password dihash saat akan dikirim ke database, dan dihash saat pencocokan dengan bcrypt agar keamanan data pengguna terpastikan (tidak berformat mentah)
Keamanan (Logika Transaksi)		Pembayaran mengecek status pembayaran, stok, dan uang pembayar dahulu untuk mencegah stok negatif, pembayaran ganda, atau pembayaran tanpa saldo memadai

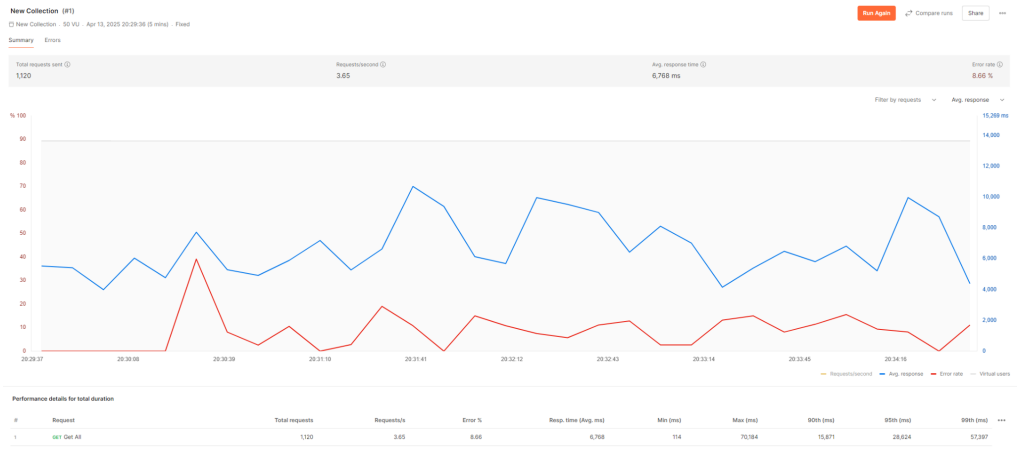
Bonus - Stress Testing

1. Service Easypanel



2. Dummy (NeonDB)

3. Stress Test (Postman)



4. Kenapa Berhasil?

   Penyebab utama keberhasilan test disini adalah **penggunaan async + wait**, serta pengecekan error yang dilakukan di awal operasi. Jika API call mengalami kegagalan, maka **operasi akan langsung diabort**, sehingga tiap request tidak perlu diproses hingga akhir dan melambatkan prosesor server. Selain itu, sistem async dan wait **memungkinkan banyak operasi berjalan bersamaan** tanpa saling tabrakan. Tanpa ini, proses harus menunggu jauh lebih lama untuk sinkronisasi, atau beresiko mengalami error saat berjalan karena race conditions dan sebagainya. Wait memastikan seluruh proses yang sedang berjalan mandiri (asinkronus) tetap aman dari hal seperti ini. Jadi, **banyak API call bisa terjadi tanpa tingkat error yang tinggi.**

5. PDF (Dalam ZIP)