

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	<pre>1 { 2 "success": true, 3 "message": "Transactions found", 4 "payload": [5 { 6 "id": "2ac49d2-1388-4834-b125-acc75f8512d0", 7 "user_id": "5a6c372c-6695-43cd-baaf-ef52b2272ab", 8 "time_id": "50f93385-1823-4811-94d8-dc0b0603d18", 9 "quantity": 1, 10 "total": 300000, 11 "status": "pending", 12 "created_at": "2025-03-18T21:17:53.511Z", 13 "user": { 14 "id": "5a6c372c-6695-43cd-baaf-ef52b2272ab", 15 "name": "netlabfinal2", 16 "email": "netlabfinal@mail.com", 17 "password": "51916930Xir116p00939u7Hx21.9Jhc10909ych2205v080yub0tM.ju", 18 "balance": 990000, 19 "created_at": "2025-03-19T04:21:00.440078" 20 } 21 }, 22 { 23 "id": "50f93385-1823-4811-94d8-dc0b0603d18", 24 "name": "Lampard A", 25 "price": 180000, 26 "store_id": "4014347b-9677-4642-9600-1026484518c4", 27 "image_url": "https://res.cloudinary.com/dhuf03kx7/image/upload/v1742139361/items/cycleppgqglu17y0867.jpg", 28 "stock": 9, 29 "created_at": "2025-03-16T15:36:01.712619" 30 } 31] 32 }</pre>	<pre>1 <!DOCTYPE html> 2 <html lang="en"> 3 4 <head> 5 <meta charset="utf-8"> 6 <title>Error</title> 7 </head> 8 9 <body> 10 <pre>Cannot POST /transaction/</pre> 11 </body> 12 13 </html></pre>

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

Tipe	Implementasi	Deskripsi
------	--------------	-----------

Error Handling (Rollback & Atomicity)

```
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 !== 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!');
    }
  }
}
```

Rollback diterapkan ke fungsi updateitem, sehingga penambahan dan perubahan data item jadi atomik dan tidak rentan kerusakan tengah operasi

```
exports.topUpBalance = 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;
  }
}
```

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

```
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;
  }
}
```

Register user menggunakan tambahan sistem rollback yang sama dengan lainnya

	<pre> exports.updateUser = async (id, userData) => { try { await db.query('BEGIN'); if (userData.email) { const existingUser = await db.query('SELECT id FROM users WHERE email = \$1 AND id != \$2', [userData.email, id]); if (existingUser.rows.length > 0) { await db.query('ROLLBACK'); throw new Error('Email already used!'); } } const result = await db.query('UPDATE users SET name = \$1, email = \$2, password = \$3 WHERE id = \$4 RETURNING id, name, email, balance, created_at', [userData.name, userData.email, userData.password, id]); if (result.rows.length === 0) { await db.query('ROLLBACK'); throw new Error('User not found!'); } await db.query('COMMIT'); return result.rows[0]; } catch (error) { await db.query('ROLLBACK'); console.error('Update User Error:', error); throw error; } }; </pre>	UpdateUser
	<pre> exports.deleteUser = async (id) => { try { await db.query('BEGIN'); const activeTransactions = await db.query('SELECT COUNT(*) FROM transactions WHERE user_id = \$1 AND status = \$2', [id, 'pending']); if (parseInt(activeTransactions.rows[0].count) > 0) { await db.query('ROLLBACK'); throw new Error('Masih ada transaksi pending!'); } const result = await db.query('DELETE FROM users WHERE id = \$1 RETURNING id, name, email, balance, created_at', [id]); if (result.rows.length === 0) { await db.query('ROLLBACK'); throw new Error('User not found!'); } await db.query('COMMIT'); return result.rows[0]; } catch (error) { await db.query('ROLLBACK'); console.error('Delete User Error:', error); throw error; } }; </pre>	DeleteUser (+ mengecek apabila ada transaksi yang masih pending sebelum delete)
Keamanan (Password Hide)	<pre> exports.updateUser = async (req, res) => { const userData = req.body; console.log('Update User - Request Body:', userData); if (!userData !userData.id !userData.name !userData.email !userData.password) { return baseResponse(res, false, 400, 'ID, name, email, and password are required'); } if (!EMAIL_REGEX.test(userData.email)) { return baseResponse(res, false, 400, 'Invalid email format', null); } if (!PASSWORD_REGEX.test(userData.password)) { return baseResponse(res, false, 400, 'Password min 8 char, ada 1 angka, dan 1 karakter khusus', null); } try { const existingUser = await userRepository.getUserById(userData.id); if (!existingUser) { return baseResponse(res, false, 404, 'User not found', null); } } const hashedPass = await bcrypt.hash(userData.password, SALT_ROUNDS); const updatedUser = await userRepository.updateUser(userData.id, { name: userData.name, email: userData.email, password: hashedPass }); baseResponse(res, true, 200, 'User updated', updatedUser); } catch (error) { console.error('Error updating user:', error); baseResponse(res, false, 500, error.message 'Server Error', null); } }; </pre>	UpdateUser (repository) tidak akan lagi mengembalikan password dalam responsnya agar lebih terjaga

Skalabilitas (Pagination)

```
exports.getAllTransactions = async (page = 1, limit = 20) => {
  try {
    const offset = (page - 1) * limit;

    const result = await db.query`
      SELECT
        t.*,
        json_build_object(
          'id', u.id,
          'name', u.name,
          'email', u.email,
          '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
      LIMIT $1 OFFSET $2
    `, [limit, offset];

    const countResult = await db.query('SELECT COUNT(*) FROM transactions');

    return {
      transactions: result.rows,
      totalCount: parseInt(countResult.rows[0].count),
      currentPage: page,
      totalPages: Math.ceil(parseInt(countResult.rows[0].count) / limit)
    };
  } catch (error) {
    console.error('Get All Transactions Error:', error);
    throw error;
  }
};
```

Respons all transactions dijadikan multiple pages (banyak halaman) agar lebih mudah diolah dan dicek saat skala sudah besar)

Keamanan (Login Limit)

```
exports.loginUser = async (req, res) => {
  const email = req.query.email;
  const password = req.query.password;
  const clientIp = req.ip || req.connection.remoteAddress;
  const rateKey = `${clientIp}:${email}`;

  if (rateLimit[rateKey] && rateLimit[rateKey].attempts >= MAX_LOGIN_ATTEMPTS) {
    if (Date.now() - rateLimit[rateKey].timestamp < RATE_LIMIT_RESET_MS) {
      return baseResponse(res, false, 429, '5x Gagal, Try Again Later.');
    } else {
      delete rateLimit[rateKey];
    }
  }

  if (!email || !password) {
    return baseResponse(res, false, 400, 'Email and password required');
  }

  try {
    const user = await userRepository.loginUser(email);

    if (!user) {
      if (!rateLimit[rateKey]) {
        rateLimit[rateKey] = { attempts: 0, timestamp: Date.now() };
      }
      rateLimit[rateKey].attempts++;
      rateLimit[rateKey].timestamp = Date.now();

      return baseResponse(res, false, 401, 'Invalid email or password', null);
    }

    const passwordMatch = await bcrypt.compare(password, user.password);
    if (!passwordMatch) {
      if (!rateLimit[rateKey]) {
        rateLimit[rateKey] = { attempts: 0, timestamp: Date.now() };
      }
      rateLimit[rateKey].attempts++;
      rateLimit[rateKey].timestamp = Date.now();

      return baseResponse(res, false, 401, 'Invalid email or password', null);
    }

    delete rateLimit[rateKey];
  }
};
```

Login dibatasi ke 5x kegagalan sebelum dilock 15 menit untuk mencegah serangan brute force menggunakan date dan rate key.

Error Handling (Transaksi Overall)

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

const transaction = transactionResult.rows[0];

const createdAt = new Date(transaction.created_at);
const now = new Date();
const ExpiredTime = 24;

if ((now - createdAt) / (1000 * 60 * 60) > ExpiredTime && transaction.status === 'pending') {
  await db.query(
    'UPDATE transactions SET status = $1 WHERE id = $2',
    ['expired', transactionId]
  );
  await db.query('ROLLBACK');
  throw new Error('Transaction has expired');
}

if (transaction.status === 'paid') {
  await db.query('ROLLBACK');
  throw new Error('Transaction is already paid');
}

if (transaction.status === 'expired') {
  await db.query('ROLLBACK');
  throw new Error('Cannot pay an expired transaction');
}

if (parseInt(transaction.balance, 10) < parseInt(transaction.total, 10)) {
  await db.query('ROLLBACK');
  throw new Error('Insufficient balance');
}

if (parseInt(transaction.stock, 10) < parseInt(transaction.quantity, 10)) {
  await db.query('ROLLBACK');
  throw new Error('Insufficient stock');
}

await db.query(
  'UPDATE transactions SET status = $1 WHERE id = $2',
  ['paid', transactionId]
);
```

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)

Tip	Implementasi	Deskripsi
Error Handling (Rollback & Atomicity)	<pre>exports.payTransaction = async (transactionId) => { try { await db.query('BEGIN'); const transactionResult = await db.query(`SELECT t.*, i.price, i.stock, u.balance FROM transactions t JOIN items i ON t.item_id = i.id JOIN users u ON t.user_id = u.id WHERE t.id = \$1`, [transactionId]); } if (transactionResult.rows.length === 0) { await db.query('ROLLBACK'); throw new Error('Transaction not found'); } const transaction = transactionResult.rows[0]; console.log('Transaction data:', transaction); if (transaction.status === 'paid') { await db.query('ROLLBACK'); throw new Error('Transaction is already paid'); } }</pre>	Sistem rollback diterapkan supaya atomik, dan jika ada kesalahan tengah transaksi, operasi dibatalkan tanpa menyebabkan kerugian bagi pihak manapun

<p>Error Handling (Nested Checking + Error Message)</p>	<pre>exports.getStoreById = async (req, res) => { try { const storeId = req.params.id; if(!storeId) { return baseResponse(res, false, 400, 'Store ID is required'); } const store = await storeRepository.getStoreById(storeId); if(!store) { return baseResponse(res, false, 404, 'Store not found', null); } baseResponse(res, true, 200, 'Store retrieved successfully', store); } catch (error) { baseResponse(res, false, 500, 'Failed to retrieve store', error); } }</pre>	<p>Pengecekan error dilakukan dengan berlapis supaya mengcover semua kemungkinan error (misalnya store tidak ada, store ID kosong, dan store ID invalid, sekaligus)</p>
<p>Error Handling (Validasi Store)</p>	<pre>exports.getItemsByStoreId = async (req, res) => { try { const { store_id } = req.params; // Cek jika store ada const storeExists = await storeRepository.getStoreById(store_id); if (!storeExists) { return baseResponse(res, false, 404, "Store doesn't exist", null); } const items = await itemRepository.getItemsByStoreId(store_id); if (items && items.length > 0) { baseResponse(res, true, 200, "Items found", items); } else { baseResponse(res, true, 200, "No items found for this store", []); } } catch (error) { console.error("Error fetching items by store ID:", error); baseResponse(res, false, 500, "Failed to fetch items", error.message); } };</pre>	<p>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</p>
<p>Skalabilitas (Async/Wait)</p>	<pre>exports.getStoreById = async (id) => { try { const result = await db.query('SELECT * FROM stores WHERE id = \$1', [id]); return result.rows[0]; } catch (error) { throw error; } };</pre>	<p>Menggunakan async dan await (di kebanyakan fungsi) agar proses tidak saling memblokir dan dapat berjalan dengan lancar bersama</p>
<p>Skalabilitas (Tipe Objek)</p>	<pre>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,</pre>	<p>Menggunakan jenis object json_build_object dalam mendapatkan data relasi query agar request ke database lebih cepat dan efisien</p>

Skalabilitas (Hapus Image)	<pre>// Hapus gambar dari Cloudinary if (itemExists.image_url) { try { // Format standar: https://res.cloudinary.com/cloud_name/image/upload/v1234567890 const urlParts = itemExists.image_url.split('/'); const publicIdWithExtension = urlParts[urlParts.length - 1]; const folder = urlParts[urlParts.length - 2]; // Remove file extension const publicId = publicIdWithExtension.split('.')[0]; // Hapus image dari Cloudinary await cloudinary.uploader.destroy(`\${folder}/\${publicId}`); } catch (cloudinaryError) { console.error("Image delete error:", cloudinaryError); } }</pre>	Saat item dihapus, gambar dengan URLnya dalam storage Cloudinary juga dihapus, agar ruang jadi lebih ringan dan tidak memenuhi DB
Keamanan (Regex)	<pre>const EMAIL_REGEX = /^[a-zA-Z0-9-]+@[a-zA-Z0-9-]+\.[a-zA-Z]{2,}\$/; const PASSWORD_REGEX = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[!@#\$%^&*~:;'\-+=(){} ~\[\]{} \\\/<>~])[A-Za-z0-9!@#\$%^&*~:;'\-+=(){} ~\[\]{} \\\/<>~]{8,}\$/;</pre>	Menggunakan regex untuk validasi email yang digunakan pengguna (harus valid) dan mencegah serangan via input string di query
Keamanan (Validasi Input)	<pre>exports.registerUser = async (req, res) => { if (!req.query.email !req.query.password !req.query.name) { return baseResponse(res, false, 400, 'Email, password, and name are required', null); } if (!EMAIL_REGEX.test(req.query.email)) { return baseResponse(res, false, 400, 'Invalid email format', null); } if (!PASSWORD_REGEX.test(req.query.password)) { return baseResponse(res, false, 400, 'Password min 8 char, ada 1 angka, dan 1 karakter khusus', null); } }</pre>	Kecocokan password dan email dengan regex + data dalam database dicek untuk memastikan keamanan data akun
Keamanan (Password Hashing)	<pre>try { const existingUser = await userRepository.getUserByEmail(req.query.email); if (existingUser) { return baseResponse(res, false, 409, 'Email already in use', null); } const hashedPass = await bcrypt.hash(req.query.password, SALT_ROUNDS); const userData = { name: req.query.name, email: req.query.email, password: hashedPass }; const newUser = await userRepository.registerUser(userData); baseResponse(res, true, 201, 'User created successfully', newUser); } catch (error) { console.error('Registration error:', error); if (error.code === '23505' && error.constraint === 'users_email_key') { return baseResponse(res, false, 409, 'Email already in use', null); } baseResponse(res, false, 500, 'Server error occurred during registration', null); }</pre>	Password dihash saat akan dikirim ke database, dan dihash saat pencocokan dengan bcrypt agar keamanan data pengguna terpastikan (tidak berformat mentah)
Keamanan (Logika Transaksi)	<pre>if (transaction.status === 'paid') { await db.query('ROLLBACK'); throw new Error('Transaction is already paid'); } if (parseInt(transaction.balance, 10) < parseInt(transaction.total, 10)) { await db.query('ROLLBACK'); throw new Error('Insufficient balance'); } if (parseInt(transaction.stock, 10) < parseInt(transaction.quantity, 10)) { await db.query('ROLLBACK'); throw new Error('Insufficient stock'); }</pre>	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

sbd / express_jesayadavidgnp APP + Service Templates III

Deploy ■ ↺ ↻ 📄 🔗 🔍 📦 🔧

CPU 0.0% Memory 127.6 MB Network I/O 7.4 KB / 3.7 KB

Source

Upload **Github** **Docker Image** **Git** **Dockerfile**

Owner * noobplatinum **Repository *** PraktikumSBD

Github username or organization name Github repository name

[github.com / noobplatinum / PraktikumSBD](#)

Branch * master **Build Path *** /

This has to be a valid branch in your repo This is useful if you have a monorepo

Save

Build

☒ **Dockerfile**
Uses the "docker build" command ([docs](#))

☐ **Buildpacks**
Choose your desired buildpacks

☐ **Nixpacks**
New way of building apps from Railway ([docs](#))

File

☒ **Dockerfile**

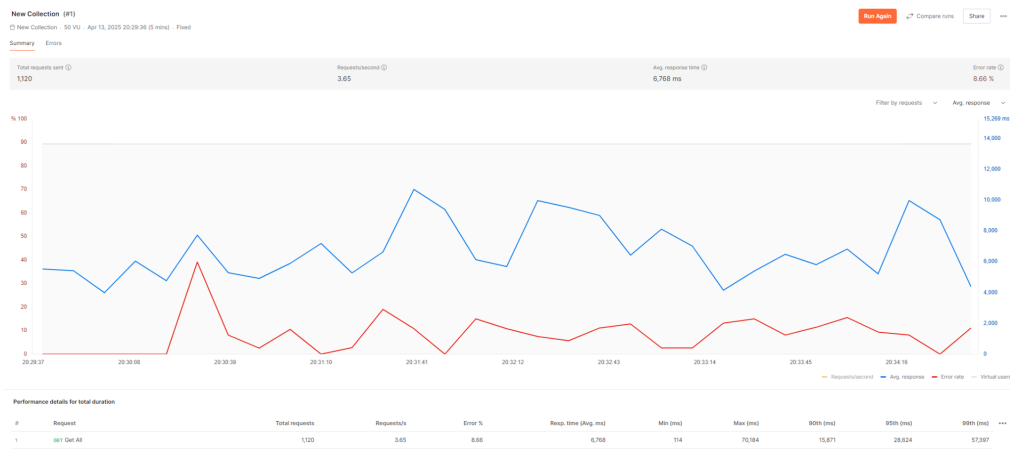
The name and path of your Dockerfile

Save

2. Dummy (NeonDB)

1 -- QUERY TRUNCATED																												
2	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('c8371e1-e1a-43c5-886f-60861d8876e6', '3e26ab0a-df41-4299-ae35-57d5fd255a77', '8dac7861-4d6f-4e69-0f46-1e8d43bb3db', 1, 700);																											
3	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('7b3eae04-287f-432d-8ce2-22e79081d2bf', '7125d96a-5957-4490-926b-6473946c59c', '071d62aa-9b2b-490b-9c9b-a8e29465f272', 1, 120);																											
4	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('db9171e-4ee9-4571-9344-dc96961caec5', '31ac2218-8895-4b16-9077-ba87b91d045d', '364b4311-6645-474f-a905-c487b9c39646', 1, 40);																											
5	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('0d594702-d4c9-4a5c-99c7-93a409636833', '8d83d070-25ae-4072-ac50-6bacfd047109', '0f0792c8-18e8-409e-9920-1348b1020ffe', 1, 25);																											
6	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('3919a4a5-8d6f-4e27-b80c-5d9d7a2ff9b3', '978b7ef1-a-beaf-4871-960a-21a46d5574ab', '320d6036-8b29-45c8-9a5c-0461a4f5112b', 1, 150);																											
7	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('b5133f6c-2aac-439e-8b10-71ba63f152aa', 'ea2e0474-90ae-4485-bf07-d3e52aa32c2c', '6e3df909-f2ce-4f6a-9a99-d07812e6ec65', 1, 130);																											
8	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('3a722084-2ebf-4708-8cdc-7406521915f6', '94b0d443-76cf-4469-a136-c311854c153', '04701b0f-b11c-4b74-8c3d-5cb3e0b8c7a', 1, 500);																											
9	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('d7ab082a-3040-4cdc-8274-9378bacc18b', 'f080472f-4008-4897-a79a-ba8e31c93f3', '9214ee17-b2c8-4158-8c0d-917b0fc13ad2', 1, 500);																											
10	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('68d28a51-8169-41b2-ucc3-87e909d22780', '8f24a404-ba16-48f6-a5ea-1873991c2224', 'c1547c39-23e1-4c68-050c-acc2758bc0a8', 1, 750);																											
11	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('b2d3279e-7a99-4c14-afa3-64d93faac082', '624ce358-94b7-45f3-b1a1-ba1e11db0694', 'c2223701-344a-401e-8dce-e079b7012d6e', 1, 30);																											
12	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('2c2e0997-9d16-49dc-a1a1-6d0689b3ad30', '59568fb6-fa64-48d0-b915-c786a507a027', '6f4c16f-8499-4995-ac59-2d21271b62dd', 1, 45);																											
13	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('0e9ffff1f-132b-4fac-9036-f02b0bc8859', '08065315-334d-409d-a084-1ba0b5fd0204', 'fef421ce-c12a-40f5-b04c-61c36172027b', 1, 20);																											
14	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('e97a64f8-f85a-47ae-ad51-27adfd23171', '730a4ea4-23a6-4ded-971f-690db09c88ce', 'e6ff3b87-ae77-4a8f-8799-a69779083525', 1, 90);																											
15	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('8d023228-2955-4b02-9371-2cdc8d821e95', '6ea951b0-853c-4ee5-988d-7aff8cf50eeb', 'cebe5c27-baac-4f07-834d-ad140b90685d', 1, 25);																											
16	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('4db09331-2206-4e9b-88dc-c2d2c0ee0e19', '6071a684-b347-4993-8708-1d0e7221231b', '08b0c60e-224f-430e-b758-43084f1ba48b', 1, 300);																											
17	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('1306578a-e05b-4455-9808-ba0e951e73f1', 'ac1a5d31-e0ee-4122-896b-9c96a0baa550', '99e0307c-af98-45d5-059f-d078a65cab1b', 1, 450);																											
18	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('15ea8357-89ec-47cb-223f-1cb79d7c9e0b', 'a744979d-cfb2-4238-8e89-572f2d42c625', 'b735e080-ba40-491e-b72a-144d221d1ef5', 1, 10);																											
19	INSERT INTO transactions (id, user_id, item_id, quantity, total) VALUES ('31aa39e5-a858-4716-801e-856dc79a1cfb', '4c3b3222-188c-4767-9562-a4cabeef095c', '63faa5e0-8717-4d28-9e0b-77806f74dbff', 1, 50);																											
20 -- QUERY TRUNCATED																												
Connected (45 queries)																												
Run Explain Analyze 53ms No result																												
1: INSERT	2: INSERT	3: INSERT	4: INSERT	5: INSERT	6: INSERT	7: INSERT	8: INSERT	9: INSERT	10: INSERT	11: INSERT	12: INSERT	13: INSERT	14: INSERT	15: INSERT	16: INSERT	17: INSERT	18: INSERT	19: INSERT	20: INSERT	21: INSERT	22: INSERT	23: INSERT	24: INSERT	25: INSERT	26: INSERT	27: INSERT	28: INSERT	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	

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)