

LAPORAN UJIAN AKHIR SEMESTER
Kas Organisasi - Aplikasi Manajemen Keuangan

Matakuliah: Pemrograman WEB Lanjut

Dosen Pengampu: Pahrul Irfan, S.Kom., M.Kom.



Anggota Kelompok:

1. **Muhammad Aryan Fathurrahman (F1D022069) (Semester 7)**
2. **Muhammad Zidan Azzaki (F1D022080) (Semester 7)**

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS MATARAM

2025

BAB I

PENDAHULUAN

1.1 Latar Belakang

Transparansi dan akuntabilitas adalah fondasi utama dalam pengelolaan keuangan organisasi. Namun, banyak organisasi yang masih bergantung pada pencatatan manual atau spreadsheet sederhana. Metode ini memiliki risiko tinggi, mulai dari kehilangan data, kesulitan pelacakan transaksi, hingga celah manipulasi data, seperti penghapusan catatan keuangan tanpa jejak audit yang jelas.

Sebagai solusi atas permasalahan tersebut, kami mengembangkan sebuah aplikasi “Sistem Kas Organisasi” berbasis web. Sisi backend dibangun dengan Node.js dan Express.js untuk performa tinggi, didukung oleh MySQL untuk integritas relasi data. Sementara itu, sisi frontend menggunakan Vue 3 untuk menyajikan *interface* yang responsif dan interaktif.

Aplikasi ini dirancang dengan fokus utama pada keamanan dan validitas data. Selain menerapkan autentikasi berbasis JWT (JSON Web Token), sistem ini dilengkapi fitur khusus Deletion Request System. Fitur ini mencegah penghapusan data transaksi secara sepihak dengan mewajibkan mekanisme persetujuan, sehingga menjamin keamanan dan validitas laporan keuangan organisasi dari tindakan manipulasi.

1.2 Tujuan

Berdasarkan latar belakang serta rumusan masalah di atas, maka tujuan pembangunan sistem ini adalah:

1. Membangun aplikasi fullstack manajemen kas organisasi.
2. Mempermudah pengurus memantau realisasi anggaran dibandingkan nilai perencanaan.
3. Menerapkan autentikasi berbasis JWT agar akses data keuangan lebih aman.
4. Menyajikan ringkasan keuangan (dashboard) secara otomatis melalui query agregasi.

1.3 Teknolog yang Digunakan

1. Backend Sisi server dibangun menggunakan Node.js 18 dan Express.js untuk performa yang efisien, didukung mysql2/promise untuk interaksi database asinkron. Keamanan dan konfigurasi sistem dikelola menggunakan bcrypt untuk enkripsi, jsonwebtoken untuk otentikasi, serta dotenv untuk variabel lingkungan.
2. Frontend Antarmuka pengguna dikembangkan dengan Vue 3 dan Vite yang cepat, menggunakan Vue Router untuk navigasi halaman dan Axios untuk komunikasi API. TailwindCSS dapat digunakan sebagai opsi styling, sementara LocalStorage menangani penyimpanan data di sisi klien.

3. Database Sistem menggunakan MySQL 8 (InnoDB) yang menjamin integritas data melalui dukungan Foreign Key dan CHECK constraint. Database ini juga telah disiapkan dengan seed data awal untuk kebutuhan inisialisasi aplikasi.

BAB II

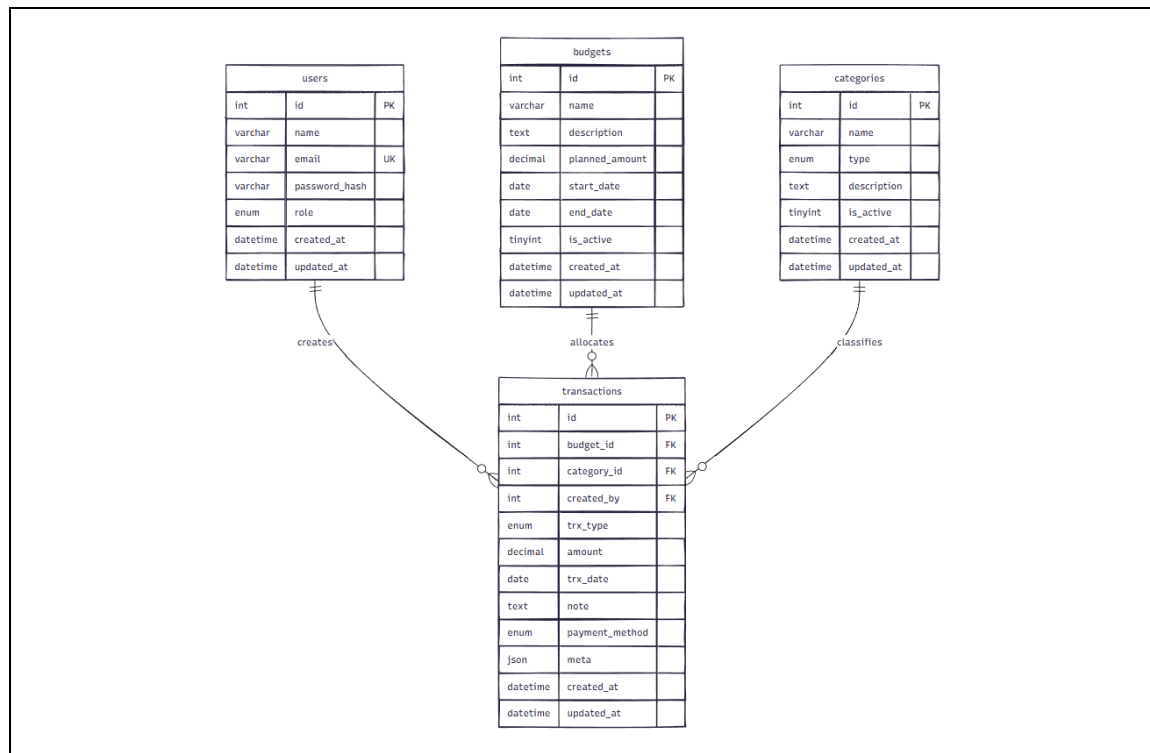
PERANCANGAN SISTEM

2.1 Deskripsi Sistem

Aplikasi SKO dirancang menggunakan arsitektur modern yang memisahkan backend sebagai penyedia layanan REST API dan frontend sebagai Single Page Application (SPA). Pada sisi server, sistem bertugas menangani logika bisnis krusial yang mencakup proses autentikasi pengguna, validasi input data yang ketat, serta manajemen data inti seperti anggaran (budgets), kategori, riwayat transaksi, dan kalkulasi ringkasan keuangan (summary).

Dari sisi antarmuka pengguna, aplikasi menyajikan visualisasi data yang dinamis dengan mengonsumsi API melalui pustaka Axios. Pengguna difasilitasi dengan halaman dashboard yang informatif, tabel daftar data, serta formulir interaktif untuk input dan melihat detail transaksi. Untuk menjamin keamanan akses, sistem menerapkan mekanisme JSON Web Token (JWT) untuk memelihara sesi pengguna, yang terintegrasi langsung dengan navigation guards pada Vue Router guna memastikan hanya pengguna yang telah terautentikasi yang dapat mengakses fitur dan halaman internal aplikasi.

2.2 Desain ERD



Struktur basis data aplikasi SKO dirancang dengan skema relasional yang berpusat pada empat entitas utama untuk mengelola aliran data keuangan secara komprehensif. Tabel Users bertindak sebagai gerbang keamanan dan manajemen identitas, menyimpan kredensial vital seperti email unik, password terenkripsi, serta hak akses (role) pengguna. Untuk kebutuhan

perencanaan, tabel Budgets mendefinisikan batasan alokasi dana yang mencakup target nominal, periode waktu mulai dan berakhir, serta status keaktifan anggaran tersebut. Di sisi lain, tabel Categories berfungsi sebagai standar klasifikasi yang mengelompokkan jenis transaksi ke dalam pemasukan atau pengeluaran agar pelaporan data menjadi lebih terstruktur.

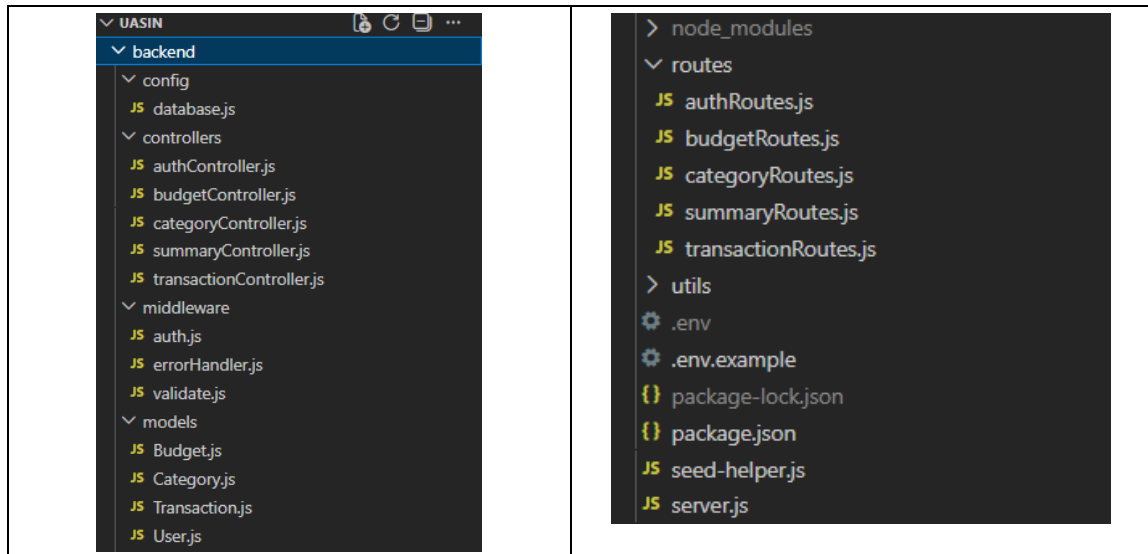
Pusat integrasi data terletak pada tabel Transactions, yang berperan sebagai entitas transaksional utama. Tabel ini memiliki relasi One-to-Many terhadap ketiga tabel referensi lainnya, di mana setiap pencatatan transaksi diikat melalui Foreign Key ke pengguna pembuat, kategori yang relevan, dan anggaran yang dialokasikan. Selain menyimpan atribut inti seperti nominal uang, tanggal, dan metode pembayaran, tabel transaksi ini juga didesain fleksibel dengan adanya kolom meta bertipe JSON untuk menampung data tambahan yang dinamis tanpa mengubah struktur tabel utama.

BAB III

IMPLEMENTASI

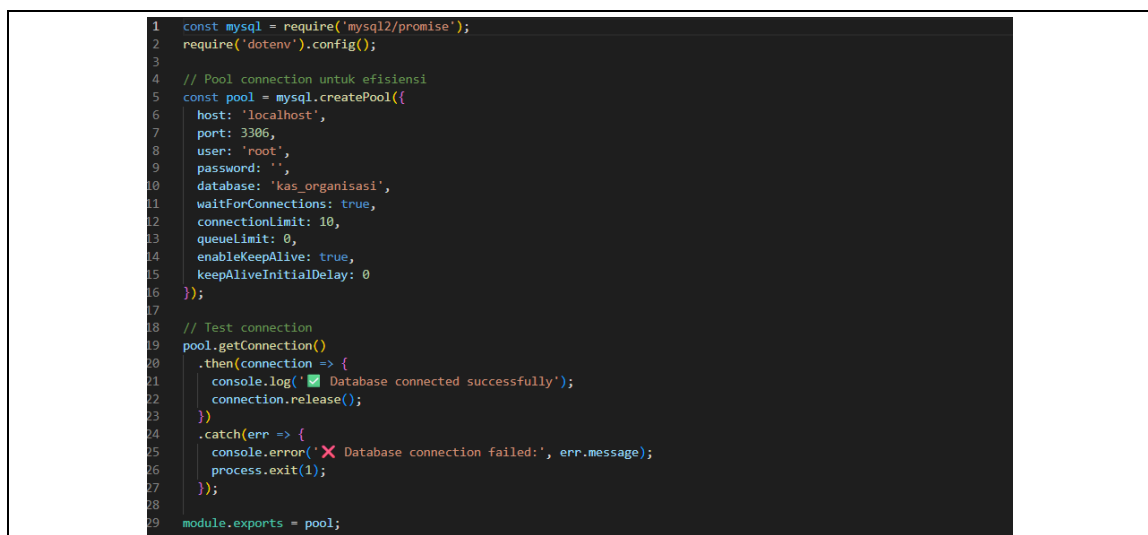
3.1 BACKEND

1. Struktur Folder



Gambar di atas merupakan struktur folder yang terorganisir dengan rapi. Di dalamnya terdapat folder config yang menyimpan konfigurasi vital seperti koneksi basis data, controllers sebagai pusat logika bisnis yang menangani permintaan dan respons, serta models yang berfungsi sebagai representasi data untuk berinteraksi langsung dengan tabel database. Selain itu, terdapat folder routes untuk mendefinisikan seluruh jalur endpoint API, middleware untuk menangani fungsi perantara seperti validasi dan autentikasi, serta utils yang berisi fungsi-fungsi bantuan umum agar kode tetap modular dan rapi.

2. Database Connection



Gambar di atas berfungsi membangun jembatan koneksi antara aplikasi backend dan database MySQL menggunakan library mysql2 yang mendukung fitur Promise untuk

penulisan kode asinkron yang modern. Sistem menerapkan teknik Connection Pooling dengan batasan maksimal 10 koneksi aktif secara bersamaan, yang bertujuan menjaga efisiensi memori server agar tidak terbebani saat banyak pengguna mengakses aplikasi. Selain itu, terdapat mekanisme validasi otomatis saat server pertama kali dinyalakan; jika koneksi ke database gagal, sistem akan langsung menghentikan proses aplikasi (`process.exit(1)`) untuk mencegah operasi berjalan tanpa penyimpanan data yang valid.

3. API

a. authRoutes.js

```
backend > routes > JS authRoutes.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const AuthController = require('../controllers/authController');
4  const validate = require('../middleware/validate');
5  const { auth } = require('../middleware/auth');
6
7  // Validation schemas
8  const registerSchema = {
9    name: { required: true, type: 'string', minLength: 3, maxLength: 100 },
10   email: { required: true, type: 'string', email: true, maxLength: 120 },
11   password: { required: true, type: 'string', minLength: 6 },
12   role: { required: false, type: 'string', enum: ['admin', 'member'] }
13 };
14
15 const loginSchema = {
16   email: { required: true, type: 'string', email: true },
17   password: { required: true, type: 'string' }
18 };
19
20 // Routes
21 router.post('/register', validate(registerSchema), AuthController.register);
22 router.post('/login', validate(loginSchema), AuthController.login);
23 router.get('/me', auth, AuthController.getProfile);
24
25 module.exports = router;
26
```

Gambar di atas merupakan file `authRoutes.js` yang berfungsi sebagai pengatur lalu lintas untuk fitur autentikasi pengguna. Di sini, `express.Router()` digunakan untuk membuat grup rute khusus, yang kemudian mendefinisikan tiga titik akses (endpoint) utama yaitu `POST /register` untuk pendaftaran akun baru, `POST /login` untuk proses masuk pengguna, dan `GET /me` untuk mengambil data profil pengguna yang sedang login. Setiap rute ini dihubungkan ke fungsi pengontrol (controller) yang sesuai (`register`, `login`, `getMe`) yang berisi logika bisnis sebenarnya, dan khusus untuk rute `/me`, diterapkan middleware `authMiddleware` di tengahnya untuk memastikan hanya pengguna yang memiliki token valid yang bisa mengakses data tersebut.

b. budgetRoutes.js

```
const express = require('express');
const router = express.Router();
const BudgetController = require('../controllers/budgetController');
const validate = require('../middleware/validate');
const { auth } = require('../middleware/auth');

// Validation schemas
const createBudgetSchema = {
  name: { required: true, type: 'string', minLength: 3, maxLength: 120 },
  description: { required: false, type: 'string' },
  planned_amount: { required: true, type: 'number', min: 0 },
  start_date: { required: true, type: 'string', date: true },
  end_date: { required: true, type: 'string', date: true },
  is_active: { required: false, type: 'number', enum: [0, 1] }
};

const updateBudgetSchema = {
  name: { required: true, type: 'string', minLength: 3, maxLength: 120 },
  description: { required: false, type: 'string' },
  planned_amount: { required: true, type: 'number', min: 0 },
  start_date: { required: true, type: 'string', date: true },
  end_date: { required: true, type: 'string', date: true },
  is_active: { required: false, type: 'number', enum: [0, 1] }
};

const idSchema = {
  id: { required: true, type: 'number', min: 1 }
};

// All routes require authentication
router.use(auth);

// Routes
router.get('/', BudgetController.getAll);
router.get('/:id', validate(idSchema), BudgetController.getById);
router.post('/', validate(createBudgetSchema), BudgetController.create);
router.put('/:id', validate({ ...idSchema, ...updateBudgetSchema }), BudgetController.update);
router.delete('/:id', validate(idSchema), BudgetController.delete);

module.exports = router;
```

Gambar di atas merupakan file `budgetRoutes.js` berfungsi sebagai pengelola rute untuk modul anggaran (budget), di mana setiap permintaan API yang masuk diarahkan ke fungsi pengontrol yang relevan dengan bantuan middleware autentikasi (`authMiddleware`) untuk memastikan keamanan akses. Rute-rute yang didefinisikan mencakup operasi CRUD lengkap: GET / untuk mengambil daftar semua anggaran (tersedia untuk semua pengguna login), POST / untuk membuat anggaran baru (khusus admin), PUT /:id untuk memperbarui data anggaran berdasarkan ID (khusus admin), dan DELETE /:id untuk menghapus anggaran (khusus admin). Struktur ini menegaskan pembagian hak akses yang ketat, di mana hanya pengguna dengan peran tertentu yang diizinkan memodifikasi data keuangan organisasi.

c. categoryRoutes.js

```
jackend > routes > # categoryRoutes.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const CategoryController = require('../controllers/categoryController');
4  const validate = require('../middleware/validate');
5  const { auth } = require('../middleware/auth');
6
7  // Validation schemas
8  const createCategorySchema = {
9    name: { required: true, type: 'string', minLength: 3, maxLength: 120 },
10   type: { required: true, type: 'string', enum: ['income', 'expense', 'both'] },
11   description: { required: false, type: 'string' },
12   is_active: { required: false, type: 'number', enum: [0, 1] }
13 };
14
15 const updateCategorySchema = {
16   name: { required: true, type: 'string', minLength: 3, maxLength: 120 },
17   type: { required: true, type: 'string', enum: ['income', 'expense', 'both'] },
18   description: { required: false, type: 'string' },
19   is_active: { required: false, type: 'number', enum: [0, 1] }
20 };
21
22 const idSchema = {
23   id: { required: true, type: 'number', min: 1 }
24 };
25
26 // All routes require authentication
27 router.use(auth);
28
29 // Routes
30 router.get('/', CategoryController.getAll);
31 router.get('/:id', validate(idSchema), CategoryController.getById);
32 router.post('/', validate(createCategorySchema), CategoryController.create);
33 router.put('/:id', validate({ ...idSchema, ...updateCategorySchema }), CategoryController.update);
34 router.delete('/:id', validate(idSchema), CategoryController.delete);
35
36 module.exports = router;
37
```

Gambar di atas merupakan file `categoryRoutes.js` yang berfungsi mengelola rute untuk manajemen kategori transaksi seperti pemasukan atau pengeluaran dengan

menerapkan kontrol akses berbasis peran (role-based access control). Di dalamnya, rute GET / dapat diakses oleh semua pengguna terautentikasi untuk melihat daftar kategori, sementara operasi modifikasi data seperti POST / (tambah kategori baru), PUT /:id (perbarui kategori), dan DELETE /:id (hapus kategori) dilindungi secara ganda menggunakan authMiddleware dan logika di dalam controller untuk memastikan hanya admin yang dapat mengeksekusinya. Mekanisme ini menjamin standarisasi data keuangan organisasi dengan mencegah pengguna biasa mengubah klasifikasi transaksi secara sembarangan.

d. summaryRoutes.js

```
backend > routes > JS summaryRoutes.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const SummaryController = require('../controllers/summaryController');
4  const { auth } = require('../middleware/auth');
5
6  // All routes require authentication
7  router.use(auth);
8
9  // Routes
10 router.get('/', SummaryController.getOverallSummary);
11 router.get('/by-budget', SummaryController.getSummaryByBudget);
12 router.get('/by-category', SummaryController.getSummaryByCategory);
13 router.get('/by-date', SummaryController.getSummaryByDate);
14
15 module.exports = router;
16
```

Gambar di atas merupakan file summaryRoutes.js yang berperan khusus dalam menyediakan data analitik dan ringkasan keuangan untuk ditampilkan pada dashboard aplikasi. Rute ini mendefinisikan endpoint GET /dashboard-stats yang terhubung ke summaryController.getDashboardStats dan dilindungi oleh authMiddleware, yang berarti hanya pengguna yang sudah login yang dapat mengaksesnya. Fungsi ini tidak digunakan untuk input data, melainkan fokus pada pengambilan data agregat (seperti total pemasukan, total pengeluaran, sisa saldo anggaran, dan grafik tren transaksi) dari database untuk disajikan dalam bentuk visualisasi yang memudahkan pengguna memantau kesehatan keuangan organisasi secara real-time.

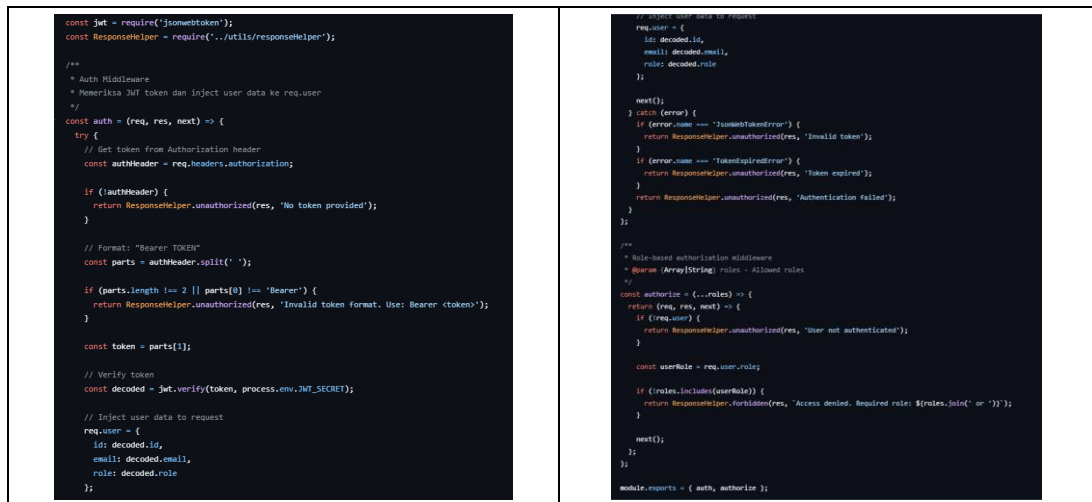
e. transactionRoutes.js

```
backend > routes > JS transactionRoutes.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const TransactionController = require('../controllers/transactionController');
4  const validate = require('../middleware/validate');
5  const { auth } = require('../middleware/auth');
6
7  // Validation schemas
8  const createTransactionSchema = {
9    budget_id: { required: true, type: 'number', min: 1 },
10   category_id: { required: true, type: 'number', min: 1 },
11   trx_type: { required: true, type: 'string', enum: ['income', 'expense'] },
12   amount: { required: true, type: 'number', min: 0.01 },
13   trx_date: { required: true, type: 'string', date: true },
14   note: { required: false, type: 'string' },
15   payment_method: { required: false, type: 'string', enum: ['cash', 'transfer', 'ewallet'] },
16   meta: { required: false, json: true }
17 };
18
19 const updateTransactionSchema = {
20   budget_id: { required: true, type: 'number', min: 1 },
21   category_id: { required: true, type: 'number', min: 1 },
22   trx_type: { required: true, type: 'string', enum: ['income', 'expense'] },
23   amount: { required: true, type: 'number', min: 0.01 },
24   trx_date: { required: true, type: 'string', date: true },
25   note: { required: false, type: 'string' },
26   payment_method: { required: false, type: 'string', enum: ['cash', 'transfer', 'ewallet'] },
27   meta: { required: false, json: true }
28 };
29
30 const idSchema = {
31   id: { required: true, type: 'number', min: 1 }
32 };
33
34 // All routes require authentication
35 router.use(auth);
36
37 // Routes
38 router.get('/', TransactionController.getAll);
39 router.get('/:id', validate(idSchema), TransactionController.getById);
40 router.post('/', validate(createTransactionSchema), TransactionController.create);
41 router.put('/:id', validate({ ...idSchema, ...updateTransactionSchema }), TransactionController.update);
42 router.delete('/:id', validate(idSchema), TransactionController.delete);
43
44 module.exports = router;
45
```

Gambar di atas merupakan file transactionRoutes.js yang memiliki peran utama dalam menangani arus data transaksi keuangan dengan mendefinisikan rute-rute API yang lengkap dan aman. Pengguna terautentikasi dapat mengakses GET / untuk melihat riwayat transaksi, GET /:id untuk detail spesifik, dan POST / untuk mencatat transaksi baru yang divalidasi oleh middleware khusus validateTransaction guna mencegah kesalahan input data. Untuk operasi sensitif, sistem menerapkan pembatasan ketat: PUT /:id memungkinkan pengguna mengedit transaksi mereka sendiri, sementara DELETE /:id mewajibkan pengguna biasa untuk mengajukan request penghapusan terlebih dahulu (yang ditangani oleh rute terpisah /request-delete/:id) dan hanya Admin yang memiliki hak istimewa untuk menyetujui atau menolak penghapusan tersebut melalui rute /approve-delete/:id, menjaga akuntabilitas dan keamanan data keuangan organisasi.

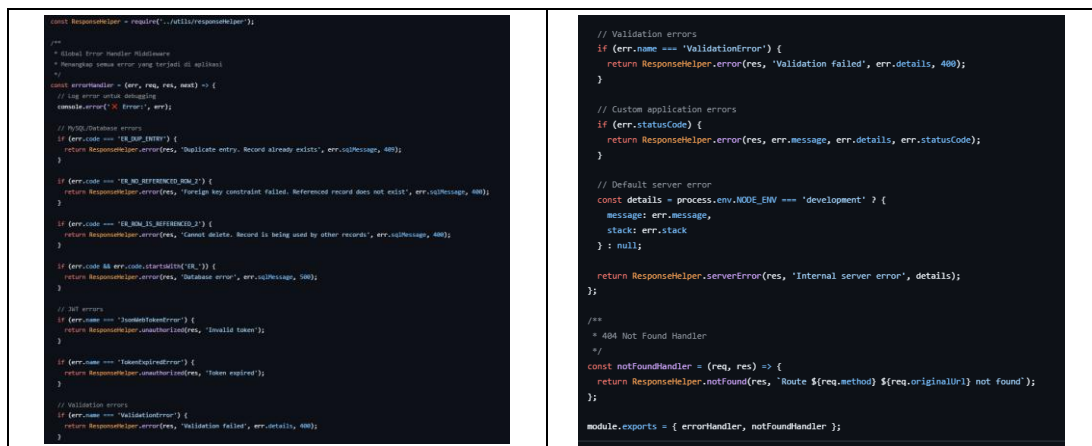
4. Middleware

a. Auth.js



Gambar di atas merupakan salah satu middleware yaitu auth.js yang berfungsi sebagai lapisan keamanan utama aplikasi yang bertugas memverifikasi identitas pengguna sebelum mereka diizinkan mengakses rute-rute privat. Sistem ini bekerja dengan memeriksa keberadaan header otorisasi (Authorization) pada setiap permintaan masuk; jika ditemukan, token JWT yang disertakan akan diekstrak dan divalidasi keasliannya menggunakan kunci rahasia (JWT_SECRET). Apabila token valid, data pengguna yang tersimpan di dalam token (seperti ID dan role) akan didekodekan dan dilampirkan ke objek req.user, sehingga rute selanjutnya dapat mengenali siapa yang sedang melakukan permintaan, namun jika token tidak ada atau tidak valid, akses akan langsung ditolak dengan pesan error.

b. ErrorHandler.js



Gambar di atas merupakan salah satu middleware yaitu ErrorHandler.js yang berfungsi sebagai jaring pengaman terakhir untuk seluruh aplikasi backend yang

menangkap setiap error yang terjadi selama proses request berlangsung. Middleware ini secara otomatis mencegah error baik itu kesalahan validasi, kegagalan database, atau error tak terduga dan mengolahnya menjadi respons JSON yang rapi dan seragam, mencegah aplikasi crash atau membocorkan detail teknis sensitif kepada pengguna.

c. Validate.js

<pre>const ResponseHelper = require('../utils/responseHelper'); /** * Validation Middleware Factory * Membuat middleware validasi berdasarkan schema yang diberikan * * @param {Object} schema - Schema validasi * @returns {Function} Express middleware */ const validate = (schema) => { return (req, res, next) => { const errors = []; const dataToValidate = { ...req.body, ...req.query, ...req.params }; // Validasi setiap field dalam schema for (const [field, rules] of Object.entries(schema)) { const value = dataToValidate[field]; // Check required if (rules.required && (value === undefined value === null value === '')) { errors.push(`Field '\${field}' is required`); continue; } // Skip validasi lain jika field tidak required dan kosong if (rules.required && (value === undefined value === null value === '')) { continue; } // Check type if (rules.type) { const actualType = Array.isArray(value) ? 'array' : typeof value; if (rules.type === 'number') { const num = Number(value); if (!isNaN(num)) { errors.push(`Field '\${field}' must be a number`); continue; } } // Convert string number to actual number dataToValidate[field] = num; } // Convert string number to actual number dataToValidate[field] = num; } // Check min/max untuk number if (rules.type === 'number') { const num = Number(value); if (rules.min !== undefined && num < rules.min) { errors.push(`Field '\${field}' must be at least \${rules.min}`); } if (rules.max !== undefined && num > rules.max) { errors.push(`Field '\${field}' must be at most \${rules.max}`); } } // Check minLength/maxLength untuk string if (rules.type === 'string') { if (rules.minLength && value.length < rules.minLength) { errors.push(`Field '\${field}' must be at least \${rules.minLength} characters`); } if (rules.maxLength && value.length > rules.maxLength) { errors.push(`Field '\${field}' must be at most \${rules.maxLength} characters`); } } // Check enum if (rules.enum && !rules.enum.includes(value)) { errors.push(`Field '\${field}' must be one of: \${rules.enum.join(', ')}`); } // Check email format if (rules.email) { const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+\$/; if (!emailRegex.test(value)) { errors.push(`Field '\${field}' must be a valid email address`); } } // Check date format (YYYY-MM-DD) if (rules.date) { const dateRegex = /^\d{4}-\d{2}-\d{2}\$/; if (!dateRegex.test(value)) { errors.push(`Field '\${field}' must be a valid date (YYYY-MM-DD)`); } else { // Validate actual date const dateObj = new Date(value); if (!isNaN(dateObj.getTime())) { errors.push(`Field '\${field}' must be a valid date`); } } } // Check JSON format if (rules.json && value) { try { if (typeof value === 'string') { JSON.parse(value); } else if (typeof value !== 'object') { errors.push(`Field '\${field}' must be valid JSON`); } } catch (e) { errors.push(`Field '\${field}' must be valid JSON`); } } // Custom validator if (rules.custom && typeof rules.custom === 'function') { const customError = rules.custom(value, dataToValidate); if (customError) { errors.push(customError); } } // Jika ada error, return error response if (errors.length > 0) { return ResponseHelper.error(res, 'Validation failed', errors, 400); } // Update req.body dengan data yang sudah divalidasi (termasuk konversi tipe) Object.assign(req.body, dataToValidate); next(); }; }; module.exports = validate;</pre>	<pre>const ResponseHelper = require('../utils/responseHelper'); /** * Validation Middleware Factory * Membuat middleware validasi berdasarkan schema yang diberikan * * @param {Object} schema - Schema validasi * @returns {Function} Express middleware */ const validate = (schema) => { return (req, res, next) => { const errors = []; const dataToValidate = { ...req.body, ...req.query, ...req.params }; // Validasi setiap field dalam schema for (const [field, rules] of Object.entries(schema)) { const value = dataToValidate[field]; // Check required if (rules.required && (value === undefined value === null value === '')) { errors.push(`Field '\${field}' is required`); continue; } // Skip validasi lain jika field tidak required dan kosong if (rules.required && (value === undefined value === null value === '')) { continue; } // Check type if (rules.type) { const actualType = Array.isArray(value) ? 'array' : typeof value; if (rules.type === 'number') { const num = Number(value); if (!isNaN(num)) { errors.push(`Field '\${field}' must be a number`); continue; } } // Convert string number to actual number dataToValidate[field] = num; } // Convert string number to actual number dataToValidate[field] = num; } // Check min/max untuk number if (rules.type === 'number') { const num = Number(value); if (rules.min !== undefined && num < rules.min) { errors.push(`Field '\${field}' must be at least \${rules.min}`); } if (rules.max !== undefined && num > rules.max) { errors.push(`Field '\${field}' must be at most \${rules.max}`); } } // Check minLength/maxLength untuk string if (rules.type === 'string') { if (rules.minLength && value.length < rules.minLength) { errors.push(`Field '\${field}' must be at least \${rules.minLength} characters`); } if (rules.maxLength && value.length > rules.maxLength) { errors.push(`Field '\${field}' must be at most \${rules.maxLength} characters`); } } // Check enum if (rules.enum && !rules.enum.includes(value)) { errors.push(`Field '\${field}' must be one of: \${rules.enum.join(', ')}`); } // Check email format if (rules.email) { const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+\$/; if (!emailRegex.test(value)) { errors.push(`Field '\${field}' must be a valid email address`); } } // Check date format (YYYY-MM-DD) if (rules.date) { const dateRegex = /^\d{4}-\d{2}-\d{2}\$/; if (!dateRegex.test(value)) { errors.push(`Field '\${field}' must be a valid date (YYYY-MM-DD)`); } else { // Validate actual date const dateObj = new Date(value); if (!isNaN(dateObj.getTime())) { errors.push(`Field '\${field}' must be a valid date`); } } } // Check JSON format if (rules.json && value) { try { if (typeof value === 'string') { JSON.parse(value); } else if (typeof value !== 'object') { errors.push(`Field '\${field}' must be valid JSON`); } } catch (e) { errors.push(`Field '\${field}' must be valid JSON`); } } // Custom validator if (rules.custom && typeof rules.custom === 'function') { const customError = rules.custom(value, dataToValidate); if (customError) { errors.push(customError); } } // Jika ada error, return error response if (errors.length > 0) { return ResponseHelper.error(res, 'Validation failed', errors, 400); } // Update req.body dengan data yang sudah divalidasi (termasuk konversi tipe) Object.assign(req.body, dataToValidate); next(); }; }; module.exports = validate;</pre>
--	--

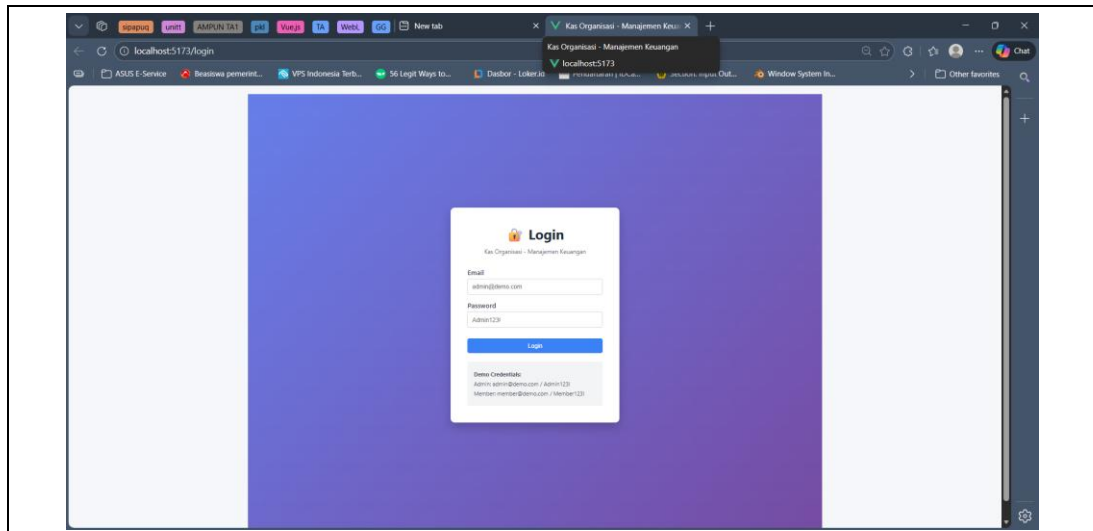
Gambar di atas merupakan salah satu middleware yaitu validate.js yang berfungsi sebagai gerbang verifikasi data yang memastikan setiap input dari pengguna memenuhi standar sebelum diproses lebih lanjut oleh sistem. Menggunakan pustaka express-validator, kode ini mendefinisikan serangkaian aturan ketat untuk pendaftaran (validateRegister) yang mewajibkan nama, email valid, dan password minimal 6 karakter; login (validateLogin) yang mengharuskan email dan password terisi; serta transaksi (validateTransaction) yang memvalidasi tipe transaksi, jumlah uang (harus angka positif), dan tanggal. Jika ada aturan yang dilanggar, fungsi handleValidationErrors akan langsung

menghentikan proses dan mengirimkan respons error detail kepada pengguna, sehingga menjaga database tetap bersih dari data sampah atau tidak valid.

3.2 FRONTEND

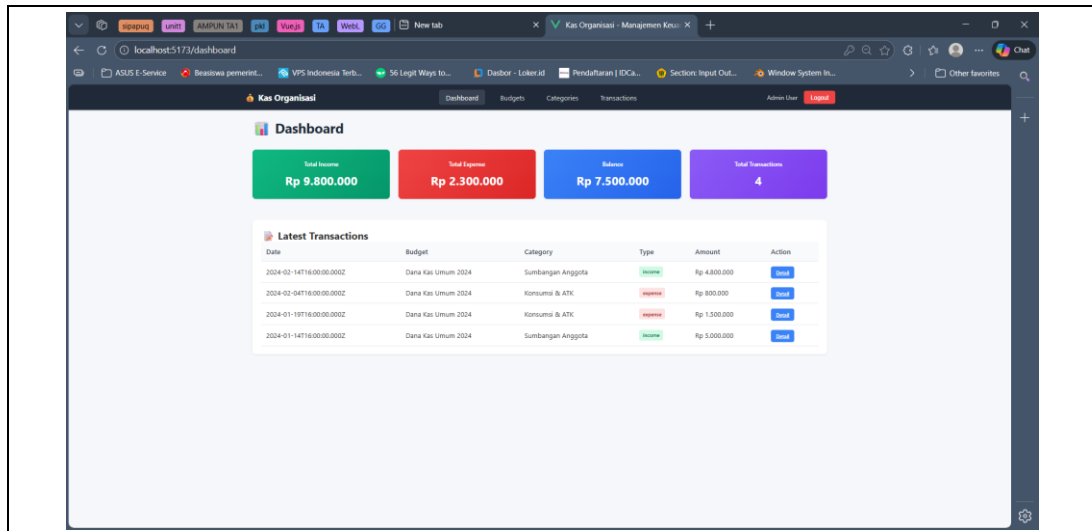
1. Halaman

a. Login page



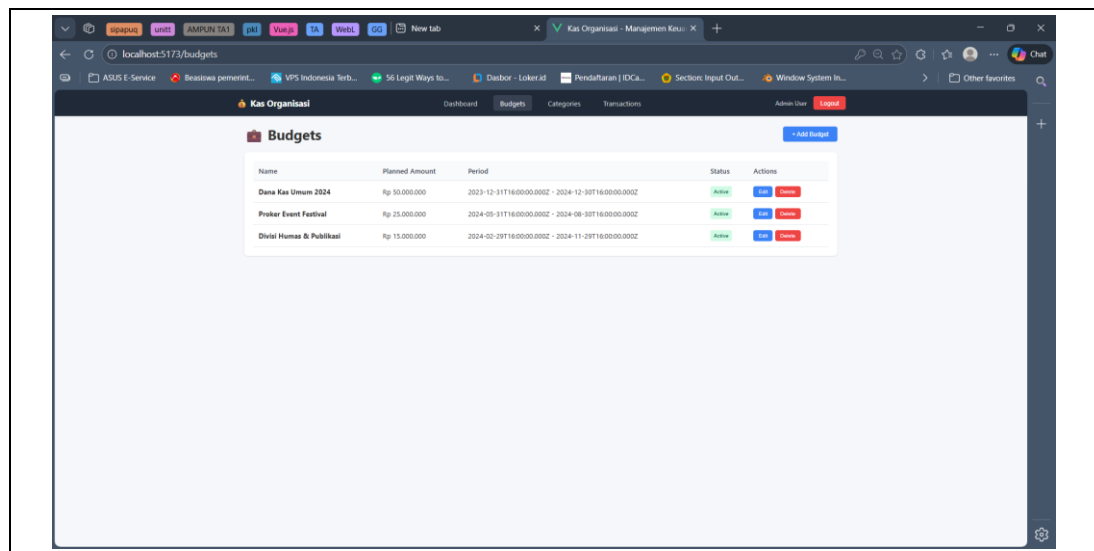
Gambar frontend di atas merupakan tampilan halaman login yang berfungsi sebagai gerbang autentikasi utama, di mana terdapat form sederhana yang mewajibkan pengguna memasukkan kredensial berupa alamat email dan kata sandi untuk masuk ke dalam sistem. Halaman ini dibangun menggunakan kerangka kerja Vue.js yang menawarkan arsitektur berbasis komponen dan konsep Single Page Application (SPA), membuat antarmuka sangat responsif tanpa perlu memuat ulang halaman saat berpindah menu. Secara teknis, proses pengiriman data login ditangani oleh pustaka HTTP client (seperti Axios yang tersimpan di folder src/api) yang menghubungkan antarmuka ini dengan endpoint backend untuk memvalidasi akun secara aman dan efisien.

b. Dashboard page



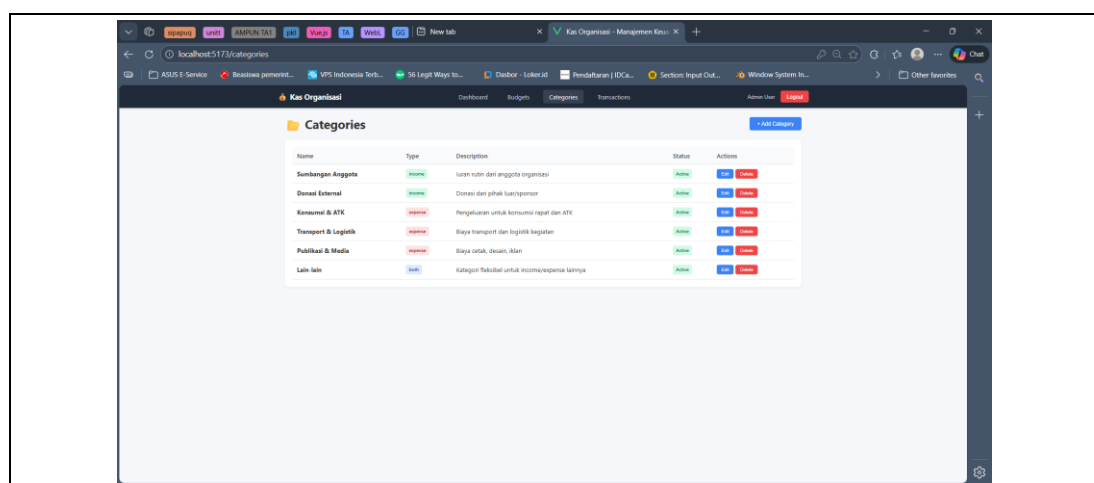
Gambar frontend di atas merupakan tampilan halaman Dashboard yang berfungsi sebagai pusat pemantauan utama, menyajikan ringkasan visual kondisi keuangan organisasi melalui kartu statistik interaktif (pemasukan, pengeluaran, saldo, dan total transaksi) serta daftar transaksi terbaru. Halaman ini dibangun menggunakan kerangka kerja Vue.js yang menerapkan konsep Single Page Application (SPA), sehingga memungkinkan pembaruan data statistik secara dinamis dan responsif tanpa perlu melakukan reload halaman penuh. Secara teknis, seluruh data agregat yang ditampilkan diperoleh melalui mekanisme fetching data menggunakan pustaka HTTP client (Axios) yang memanggil endpoint API khusus di sisi backend (seperti `/api/summary`), memastikan akurasi perhitungan keuangan tersaji secara real-time kepada pengguna.

c. Budget page



Gambar frontend di atas menampilkan halaman Manajemen Anggaran (Budgets) yang berfungsi sebagai pusat kendali perencanaan keuangan, di mana pengguna dapat mengelola alokasi dana organisasi secara terstruktur dengan menetapkan target nominal dan periode waktu tertentu. Antarmuka ini menyajikan tabel data informatif yang memuat rincian nama anggaran, jumlah rencana (planned amount), durasi pelaksanaan, serta status keaktifan, yang dilengkapi dengan tombol aksi (Actions) untuk memperbarui atau menghapus entitas anggaran sesuai kebutuhan. Dari sisi teknis, halaman ini dikembangkan menggunakan komponen Vue.js yang dinamis, di mana data anggaran diambil secara real-time dari server melalui permintaan HTTP GET ke endpoint API terkait saat halaman dimuat, memastikan pengurus organisasi selalu mendapatkan informasi alokasi dana yang akurat dan terkini.

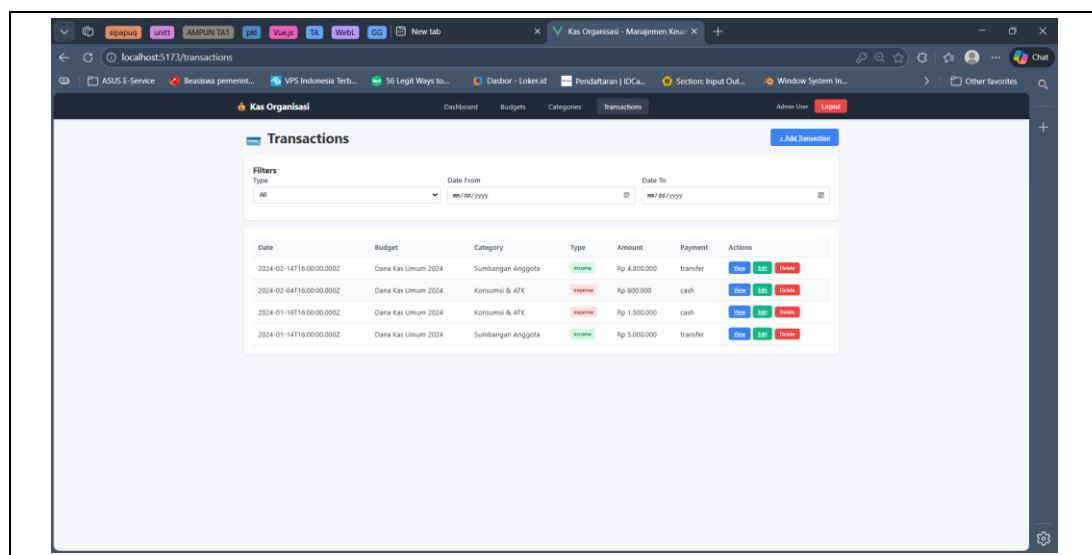
d. Categories page



Gambar frontend di atas merupakan tampilan halaman Manajemen Kategori (Categories) yang berfungsi sebagai referensi utama untuk mengklasifikasikan jenis

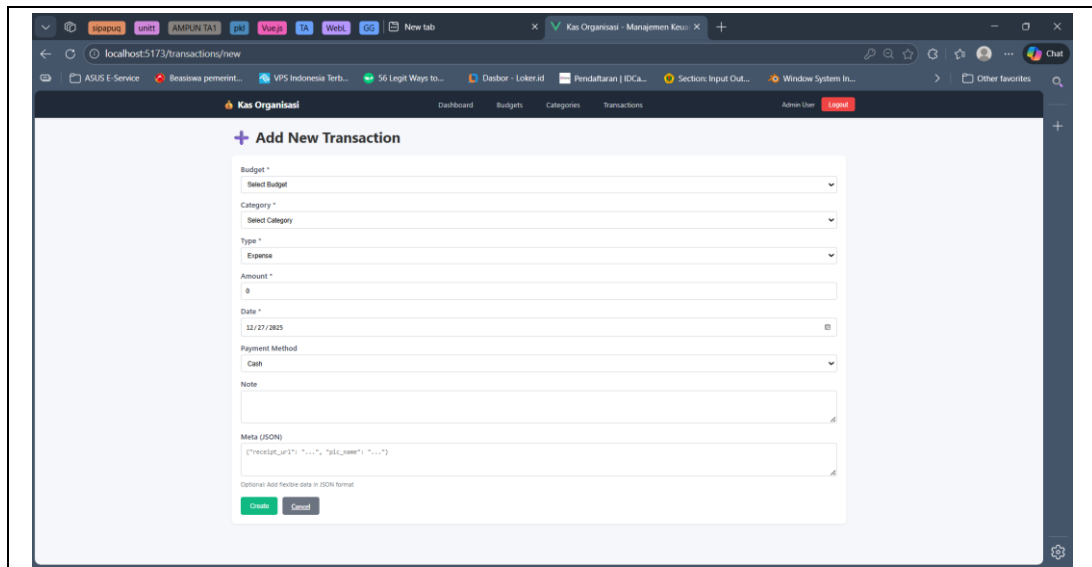
transaksi, baik itu pemasukan (income) maupun pengeluaran (expense), guna menjamin kerapian dan standarisasi laporan keuangan organisasi. Antarmuka ini menyajikan tabel data yang memuat informasi krusial seperti nama kategori, tipe aliran dana, deskripsi peruntukan, serta status keaktifan, yang dilengkapi dengan fasilitas tombol aksi untuk pengelolaan data secara fleksibel (Edit/Delete). Halaman ini dibangun menggunakan kerangka kerja Vue.js dengan konsep Single Page Application (SPA) yang menjamin perpindahan antar menu tanpa reload, sementara di sisi teknis, data kategori tersebut diambil dan disinkronisasi dari database server melalui permintaan HTTP asinkron (Axios) yang menghubungkan antarmuka dengan endpoint API secara aman.

e. Transactions page



Gambar frontend di atas menampilkan halaman Manajemen Transaksi (Transactions) yang berperan sebagai jurnal keuangan utama, tempat seluruh aktivitas pemasukan dan pengeluaran dicatat secara terperinci untuk memastikan transparansi arus kas organisasi. Antarmuka ini didesain sangat informatif dengan menyertakan fitur filtering data berdasarkan tipe transaksi dan rentang tanggal, serta menyajikan tabel lengkap yang memuat informasi tanggal, anggaran terkait, kategori, hingga metode pembayaran. Secara teknis, halaman ini memanfaatkan reaktivitas Vue.js untuk menangani interaksi pengguna, di mana setiap perubahan pada filter akan memicu permintaan data baru ke backend melalui API dengan parameter query yang dinamis, memastikan laporan transaksi yang ditampilkan selalu akurat dan relevan sesuai kebutuhan audit pengguna.

f. Add New Transaction page

The image shows a web browser window displaying the 'Add New Transaction' form. The browser's address bar shows 'localhost:5173/transactions/new'. The application's header includes a navigation menu with 'Dashboard', 'Budgets', 'Categories', and 'Transactions', along with a 'Logout' button. The form itself is titled '+ Add New Transaction' and contains several input fields: 'Budget' (a dropdown menu), 'Category' (a dropdown menu), 'Type' (a dropdown menu with 'Expense' selected), 'Amount' (a text input field), 'Date' (a date picker showing '12/27/2025'), 'Payment Method' (a dropdown menu with 'Cash' selected), and 'Note' (a text area). Below these fields is a 'Meta (JSON)' field containing a JSON string: '{"transaction": "...", "date": "..."}'. At the bottom of the form are two buttons: 'Create' and 'Cancel'.

Gambar frontend di atas menampilkan halaman formulir “Add New Transaction” yang berfungsi sebagai antarmuka input utama bagi pengguna untuk mencatat aktivitas keuangan baru ke dalam sistem secara mendetail. Formulir ini menyediakan berbagai field input esensial mulai dari pemilihan anggaran dan kategori terkait, penetapan tipe transaksi (pemasukan/pengeluaran), hingga pengisian nominal dan metode pembayaran, dilengkapi dengan kolom khusus “Meta (JSON)” untuk mengakomodasi data tambahan yang bersifat fleksibel. Secara teknis, antarmuka ini memanfaatkan fitur reactive binding pada Vue.js untuk menangkap input pengguna secara real-time, yang kemudian akan dikemas menjadi objek JSON dan dikirimkan ke backend melalui metode HTTP POST setelah tombol “Create” ditekan, menjamin integritas data sebelum tersimpan di basis data.

2. Routing



Gambar 2di atas memperlihatkan implementasi konfigurasi routing menggunakan Vue Router yang bertugas mengatur navigasi antar halaman dalam aplikasi Single Page Application (SPA) tanpa perlu memuat ulang peramban. Kode ini mendefinisikan daftar rute secara terstruktur dalam sebuah array, di mana setiap objek memetakan path URL tertentu ke komponen Vue yang sesuai, sekaligus menyertakan properti meta sebagai mekanisme kontrol akses keamanan. Melalui konfigurasi ini, sistem menerapkan proteksi bertingkat di sisi klien (client-side), yaitu membatasi akses ke halaman privat (seperti Dashboard dan Transaksi) hanya bagi pengguna yang terautentikasi (`requiresAuth: true`) serta menerapkan validasi peran khusus (role-based) untuk fitur administratif—seperti mengedit transaksi atau mengakses kategori—agar hanya dapat dibuka oleh pengguna dengan status 'admin'.

3. Konsumsi API

Halaman/Komponen	Method	Endpoint	Keterangan
Login	POST	<code>/auth/login</code>	Kirim kredensial, simpan token + data user saat sukses
Dashboard	GET	<code>/summary</code>	Ambil ringkasan total pemasukan/pengeluaran, saldo, jumlah transaksi, dan 5 transaksi terbaru
Budgets	GET	<code>/budgets</code>	Daftar seluruh budget aktif untuk tabel dan form select
Budgets	POST	<code>/budgets</code>	Tambah budget baru (khusus admin)
Budgets	PUT	<code>/budgets/:id</code>	Perbarui budget (khusus admin)
Budgets	DELETE	<code>/budgets/:id</code>	Hapus budget (khusus admin)
Categories	GET	<code>/categories</code>	Daftar kategori untuk tabel dan form select
Categories	POST	<code>/categories</code>	Tambah kategori baru (admin)
Categories	PUT	<code>/categories/:id</code>	Edit kategori (admin)
Categories	DELETE	<code>/categories/:id</code>	Hapus kategori (admin)
Transactions list	GET	<code>/transactions</code> + query <code>page, limit, trx_type, date_from, date_to</code>	Ambil daftar transaksi dengan filter & pagination
Transactions list	DELETE	<code>/transactions/:id</code>	Hapus transaksi (admin)
Transactions list	POST	<code>/transactions/:id/request-delete</code>	Memberi minta admin menghapus transaksi
Transaction form	GET	<code>/transactions/:id</code>	Ambil detail transaksi saat mode edit
Transaction form	POST	<code>/transactions</code>	Buat transaksi baru
Transaction form	PUT	<code>/transactions/:id</code>	Update transaksi
Transaction detail	GET	<code>/transactions/:id</code>	Tampilkan detail lengkap transaksi

Gambar tabel di atas mempresentasikan spesifikasi teknis implementasi Konsumsi API (API Consumption) pada sisi frontend, yang berfungsi sebagai peta jalan integrasi data antara antarmuka pengguna dan server. Tabel ini mendetailkan daftar endpoint REST

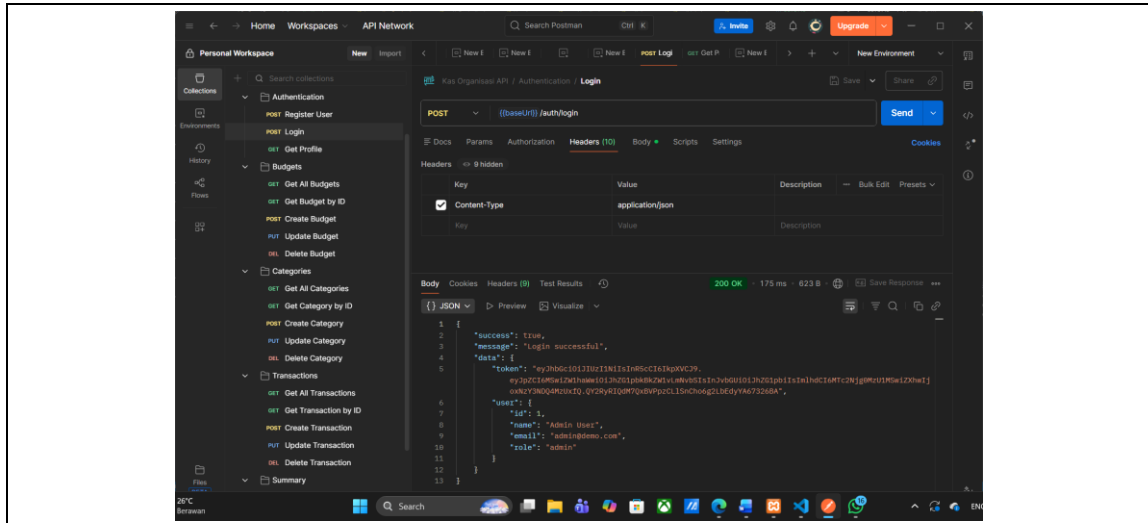
API yang dipanggil oleh setiap halaman atau komponen—seperti Login, Dashboard, dan Transaksi—beserta metode HTTP yang digunakan (GET, POST, PUT, DELETE) untuk menjalankan operasi CRUD (Create, Read, Update, Delete). Dokumentasi ini menjadi acuan vital bagi pengembang dalam menyusun kode request (menggunakan pustaka seperti Axios), memastikan bahwa setiap interaksi pengguna, mulai dari autentikasi hingga pengelolaan anggaran, dikirimkan ke alamat server yang tepat dengan parameter yang valid demi menjaga konsistensi dan keamanan pertukaran data.

BAB IV

PENGUJIAN

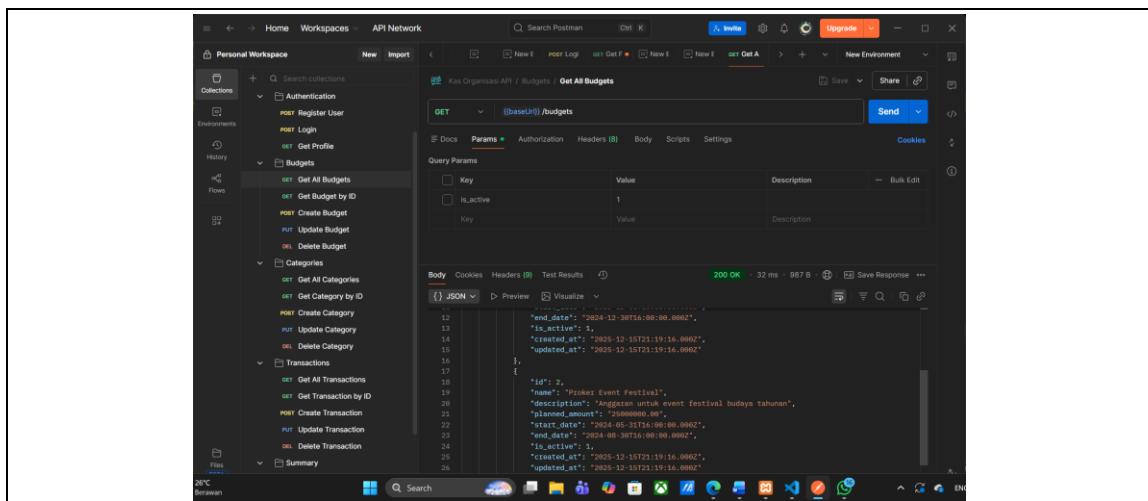
4.1 Pengujian API

1. POST



Gambar di atas menampilkan hasil pengujian API untuk fitur autentikasi pengguna (login) menggunakan software Postman, di mana permintaan dikirimkan ke endpoint POST /auth/login untuk memverifikasi kredensial akun. Hasil pengujian menunjukkan bahwa server merespons dengan status HTTP 200 OK dan mengembalikan objek JSON yang memuat pesan sukses serta data krusial berupa token akses (JWT) dan detail profil pengguna. Keberhasilan pengujian ini mengonfirmasi bahwa logika keamanan di sisi backend berfungsi dengan baik, di mana token yang dihasilkan tersebut nantinya akan digunakan sebagai kunci otorisasi untuk mengakses rute-rute privat lainnya dalam sistem.

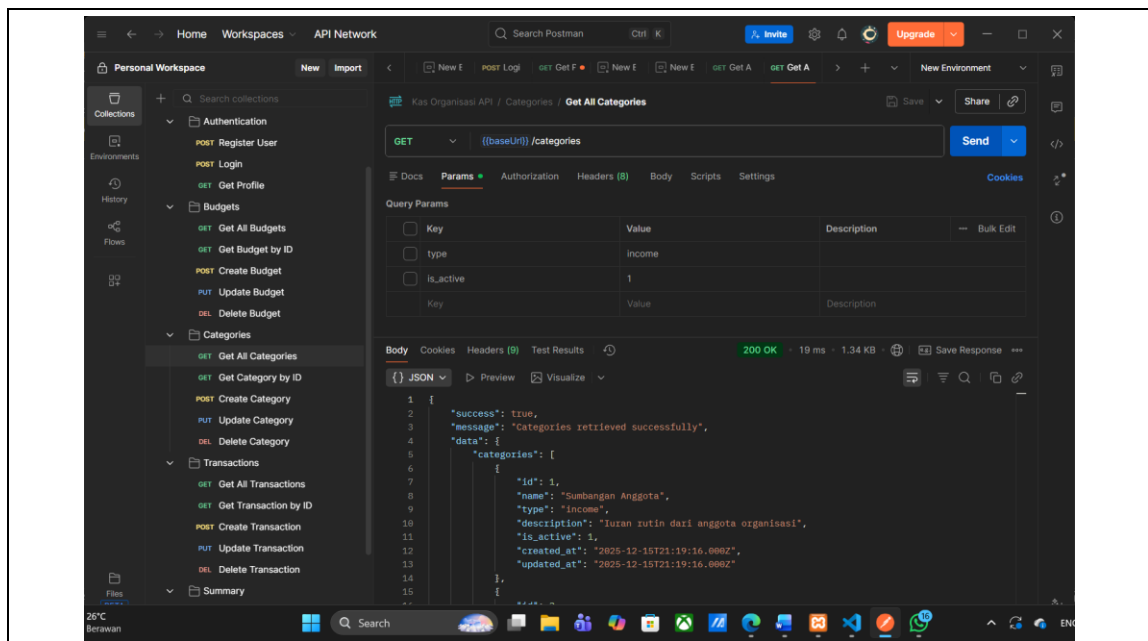
2. GET ALL BUDGETS



Gambar di atas menampilkan hasil pengujian API untuk fitur pengambilan data anggaran (Get All Budgets) menggunakan Postman, di mana permintaan HTTP dikirimkan ke endpoint

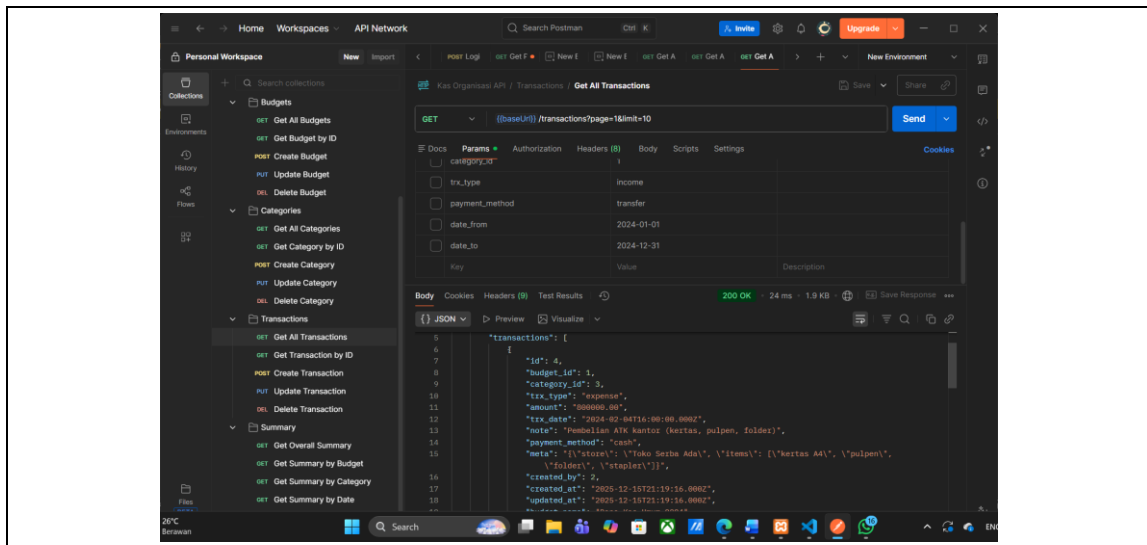
GET /budgets tanpa memerlukan parameter body tambahan. Hasil eksekusi menunjukkan bahwa server merespons dengan status kode 200 OK dan mengembalikan payload JSON berisi daftar objek anggaran yang lengkap, mencakup informasi vital seperti nama kegiatan, nominal rencana (planned amount), periode waktu, serta status keaktifan. Keberhasilan pengujian ini memvalidasi bahwa fungsi controller di sisi backend telah berhasil melakukan query ke basis data dan menyajikan data tersebut dalam format terstruktur yang siap dikonsumsi oleh tabel pada antarmuka frontend.

3. GET ALL CATEGORIES



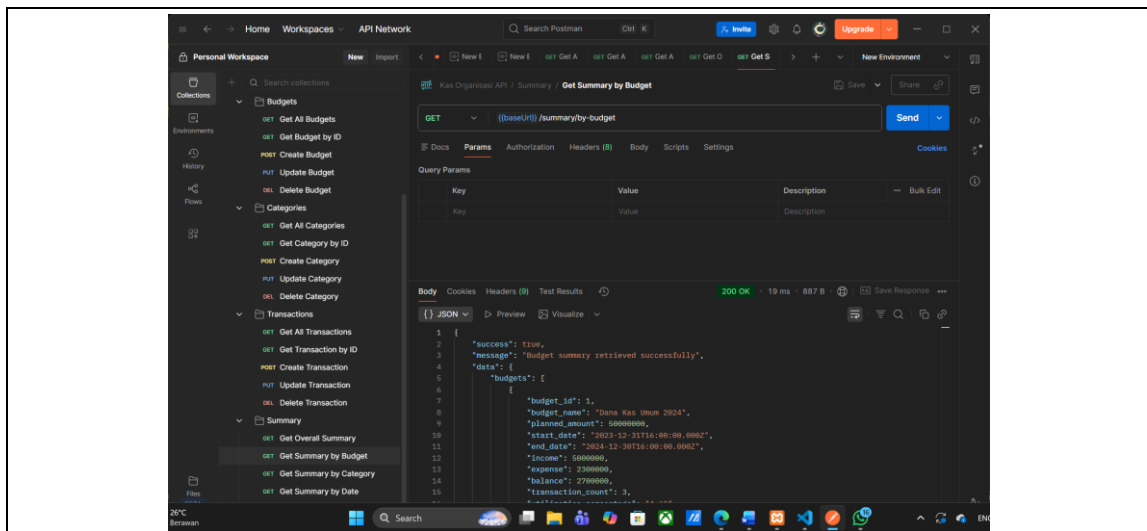
Gambar di atas menampilkan hasil pengujian API untuk fitur pengambilan daftar kategori transaksi (Get All Categories) menggunakan Postman, di mana permintaan HTTP dikirimkan ke endpoint GET /categories untuk memperoleh seluruh data klasifikasi keuangan yang tersedia. Hasil pengujian memperlihatkan bahwa server memberikan respons sukses dengan status 200 OK, diikuti oleh lampiran data (payload) dalam format JSON yang berisi himpunan objek kategori lengkap dengan atribut identitas seperti nama, jenis aliran dana (income/expense), serta deskripsi. Keberhasilan permintaan ini memverifikasi bahwa sistem backend mampu menyediakan data referensi yang valid, yang nantinya krusial bagi frontend untuk menyajikan opsi kategori yang tepat saat pengguna melakukan pencatatan transaksi.

4. GET ALL TRANSACTIONS



Gambar di atas menampilkan hasil pengujian API untuk fitur pengambilan riwayat transaksi (Get All Transactions) menggunakan Postman, di mana permintaan HTTP dikirimkan ke endpoint GET /transactions yang dilengkapi dengan parameter query seperti page dan limit untuk kebutuhan paginasi data. Respons server menunjukkan status sukses 200 OK dengan muatan data (payload) JSON yang berisi daftar transaksi secara terperinci, meliputi atribut vital seperti ID anggaran, kategori, nominal, tipe transaksi, hingga data fleksibel dalam kolom meta. Pengujian ini memvalidasi bahwa controller transaksi di sisi backend mampu menangani permintaan data kompleks dengan efisien, memastikan antarmuka pengguna nantinya dapat menampilkan riwayat keuangan yang akurat dan terpaginasi.

5. GET SUMMARY BY BUDGET

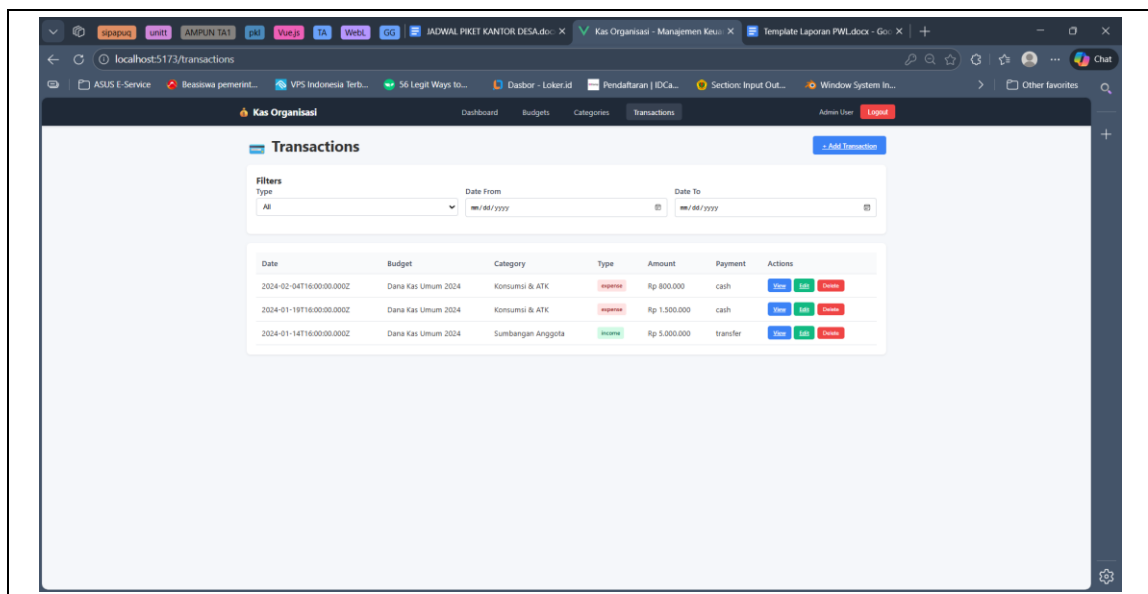


Gambar di atas menampilkan hasil pengujian API untuk fitur ringkasan keuangan berbasis anggaran (Get Summary by Budget) menggunakan Postman, di mana permintaan

HTTP dikirimkan ke endpoint GET /summary/by-budget untuk memperoleh analisis performa setiap pos anggaran. Server merespons dengan status 200 OK dan mengembalikan data JSON yang berisi kalkulasi otomatis mengenai total pemasukan, total pengeluaran, sisa saldo (balance), serta frekuensi transaksi untuk masing-masing anggaran aktif. Keberhasilan pengujian ini mengonfirmasi bahwa logika agregasi data di backend berfungsi optimal, menyediakan metrik evaluasi yang esensial bagi antarmuka dashboard untuk memvisualisasikan efektivitas pengelolaan dana organisasi secara real-time.

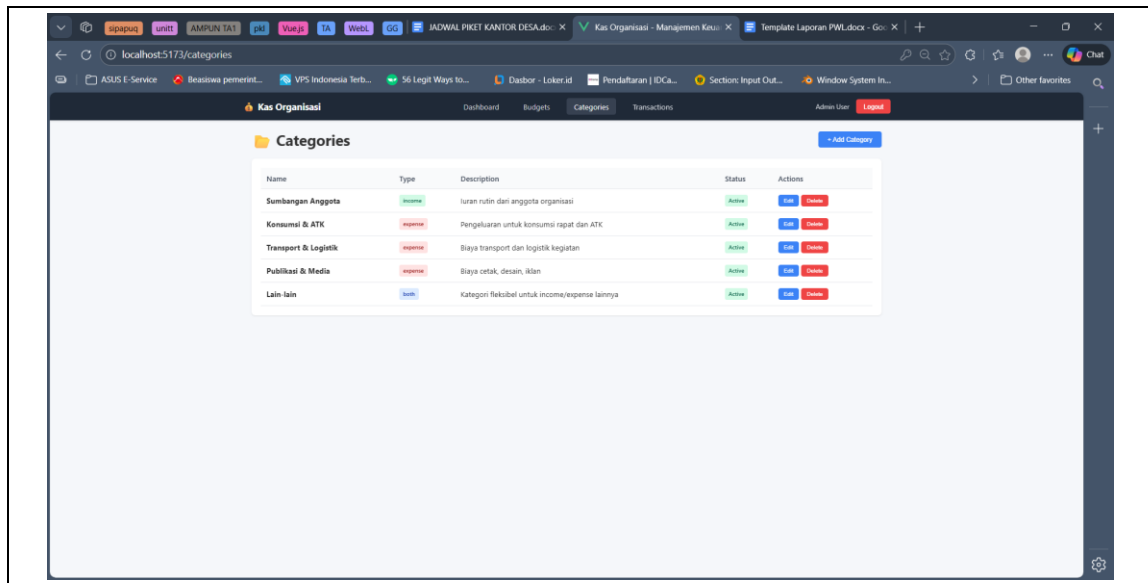
4.2 Pengujian Aplikasi

1. TRANSACTIONS



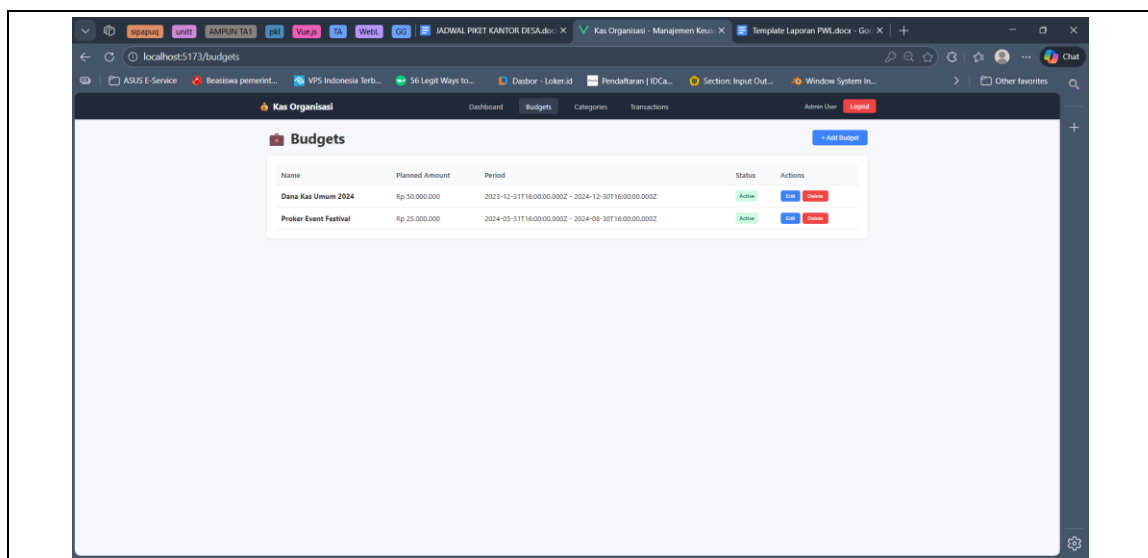
Gambar di atas menampilkan hasil pengujian antarmuka pengguna (User Interface Testing) pada halaman Transaksi, yang menunjukkan keberhasilan sistem dalam memuat dan menyajikan daftar riwayat keuangan secara terstruktur dari basis data. Dalam pengujian ini, tabel data terlihat berfungsi dengan baik menampilkan atribut krusial seperti tanggal, kategori, dan nominal, sementara fitur interaktif seperti filter pencarian dan tombol aksi (Edit/Delete) tersedia untuk digunakan, memvalidasi bahwa integrasi antara komponen frontend Vue.js dan endpoint API transaksi telah berjalan stabil dalam menangani permintaan data pengguna.

2. CATEGORIES



Gambar di atas menampilkan hasil pengujian antarmuka pengguna pada halaman Manajemen Kategori (Categories), yang membuktikan kemampuan sistem dalam memuat dan menampilkan data referensi klasifikasi transaksi secara akurat dari basis data. Pada tampilan tersebut, tabel data berhasil menyajikan informasi detail mencakup nama kategori, jenis aliran dana (income/expense), serta deskripsi peruntukan, lengkap dengan status keaktifan yang divisualisasikan dengan indikator warna hijau. Keberadaan tombol aksi interaktif (Edit/Delete) dan tombol Add Category juga mengonfirmasi bahwa fitur pengelolaan CRUD kategori telah terimplementasi dengan baik dan siap digunakan oleh administrator untuk menstandarisasi pelaporan keuangan organisasi.

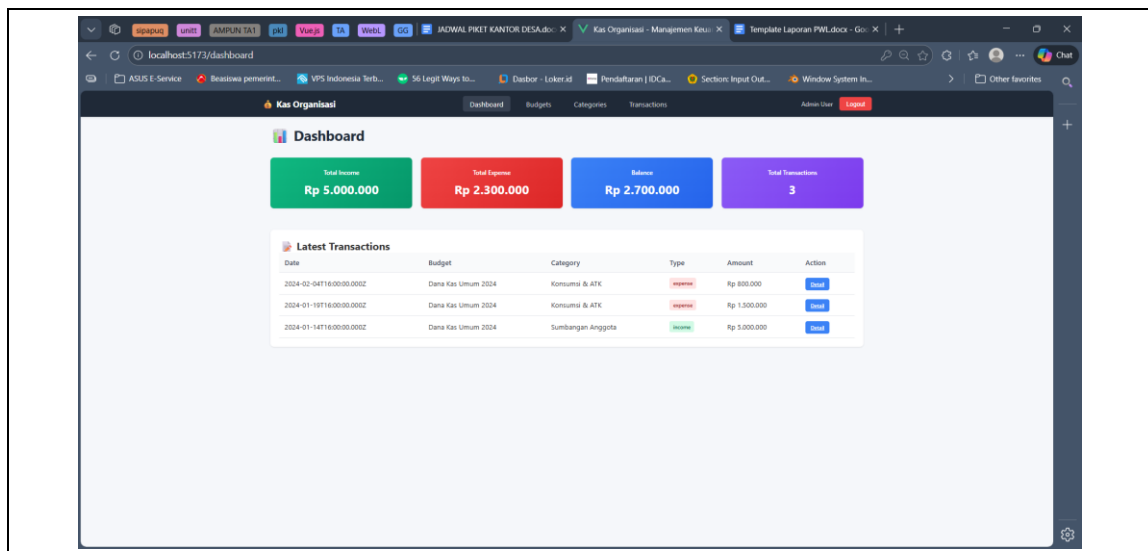
3. BUDGETS



Gambar di atas menampilkan hasil pengujian antarmuka pengguna pada halaman Manajemen Anggaran (Budgets), yang memvalidasi kemampuan aplikasi dalam memuat dan

menyajikan daftar perencanaan keuangan secara dinamis dari server. Pada tampilan tersebut, tabel data secara akurat menampilkan rincian alokasi dana yang mencakup nama anggaran, target nominal (planned amount), rentang periode waktu, serta status keaktifan yang ditandai dengan label visual. Keberhasilan pemuatan data ini, beserta tersedianya tombol aksi operasional (Edit/Delete) dan tombol Add Budget, mengonfirmasi bahwa integrasi antara komponen view Vue.js dan endpoint API pengelolaan anggaran telah berjalan sesuai spesifikasi fungsional yang dirancang.

4. DASHBOARD



Gambar di atas menampilkan halaman Dashboard yang berfungsi sebagai panel kontrol utama untuk memantau kesehatan finansial organisasi secara sekilas dan menyeluruh. Antarmuka ini menyajikan visualisasi data makro melalui kartu statistik berwarna yang memuat informasi krusial seperti Total Pemasukan, Total Pengeluaran, Saldo Akhir, dan Jumlah Transaksi, serta dilengkapi dengan tabel Latest Transactions untuk meninjau aktivitas keuangan terkini. Dikembangkan dengan basis komponen Vue.js, halaman ini menerapkan mekanisme reactive data fetching yang secara otomatis mengirimkan permintaan HTTP GET ke endpoint statistik di backend saat dimuat, memastikan pengguna selalu mendapatkan sajian data keuangan yang akurat dan real-time tanpa perlu memuat ulang halaman secara manual.

LAMPIRAN

Link Repository Github:

<https://github.com/danJidan/Web-Kas-Organisasi.git>