DevBeast – Expense & Income Management Platform

Enterprise-Grade Backend Project Documentation

1. Executive Summary

DevBeast is a backend-first, scalable Expense & Income Management Platform designed to help users track personal finances, analyze spending patterns, and generate financial insights. The system is built using Java 21, Spring Boot, PostgreSQL, and Docker, following clean architecture principles and modern backend best practices.

2. Business Problem Statement

Individuals often lack a centralized, reliable system to track expenses and income, understand spending behavior, and securely access financial data. DevBeast addresses this problem with a clean REST API, secure data ownership, and extensible design.

3. Scope of the Application

In Scope: User management, expense management, income management, categorization, reporting APIs, authentication, and persistence.
Out of Scope: Frontend UI, mobile applications, third-party bank integrations.

4. System Architecture

The application follows a layered monolithic architecture with clear separation of concerns: Controller → Service → Repository → Database. The system is container-ready and cloud-friendly.

5. Technology Stack

Backend: Java 21, Spring Boot, Spring Data JPA, Hibernate, Maven, JUnit 5.
Database: PostgreSQL 16, Flyway, HikariCP.
Infrastructure: Docker, Linux.

6. Application Modules

User Management: Handles user identity and ownership.
Expense Management: Core module for tracking expenses.
Income Management: Tracks income sources.
Category Management: Organizes expenses.
Security: JWT-based authentication (planned).

7. Expense Management Module

Expenses are recorded with amount, category, date, and description. Each expense is owned by a user and supports full CRUD operations and filtering.

8. Security Design

The platform will use stateless JWT-based authentication with role-based authorization. All data access will be user-scoped to ensure isolation and privacy.

9. Database Design

The database schema is normalized with foreign key relationships, indexed columns for performance, and strict referential integrity.

10. Validation & Exception Handling

Input validation is enforced using Jakarta Validation annotations. All errors are handled centrally using @ControllerAdvice with consistent API responses.

11. Testing Strategy

The project follows a layered testing approach including unit tests, repository tests, and integration tests. Mockito and JUnit 5 are used for testing.

12. Deployment Strategy

The application is containerized using Docker and designed to be deployed across multiple environments with externalized configuration.

13. Future Enhancements

Angular frontend, reporting dashboards, Redis caching, CI/CD pipelines, and advanced analytics.

14. Interview Value

This project demonstrates real-world backend engineering skills including system design, persistence, security, and scalability.