System Design Document (SDD)

Project Title: BudgetIQ
Version: 1.0
Date: October 2025
Author: Krishna Bhatt

1. Introduction

1.1 Purpose

This System Design Document (SDD) describes the architecture, system components, data design, and interface details for the **BudgetIQ** application.
It serves as a guide for developers, testers, and future maintainers to understand the structure and design logic behind the system.

1.2 Scope

BudgetIQ is a **web-based personal finance management system** that enables users to track expenses, create budgets, manage recurring transactions, and set financial goals.
The system integrates analytics, smart alerts, and security mechanisms (JWT, 2FA).
Future versions (v2.0) will introduce **AI-driven insights** and **cloud synchronization**.

1.3 References

- Software Requirements Specification (SRS) — BudgetIQ v1.0
- Spring Boot, React.js, MySQL Documentation
- OWASP Security Guidelines

2. System Overview

BudgetIQ follows a **three-tier architecture** consisting of:

1. **Presentation Layer (Frontend – React.js)**
   Provides an interactive UI for users to perform operations like adding transactions, setting goals, and viewing analytics.
2. **Application Layer (Backend – Spring Boot)**
   Handles business logic, authentication, API endpoints, and communication with the database.
3. **Data Layer (Database – MySQL)**
   Stores all persistent data such as user details, budgets, transactions, goals, and recurring schedules.

(Planned Future Layers: Cloud Storage Integration, AI Microservice)

3. System Architecture

3.1 Architectural Design

Architecture Type: Layered MVC (Model-View-Controller)

Description:

- **View:** React.js components (Dashboard, Expense List, Goal Tracker, etc.)
- **Controller:** REST Controllers in Spring Boot that handle HTTP requests and responses.
- **Model:** Java entities mapped to database tables using JPA and Hibernate ORM.

High-Level Architecture Diagram:

```
+------------------------------+
|         Frontend             |
|  React.js (UI Components)    |
+-------------↓----------------+
|       RESTful APIs           |
| Spring Boot (Controllers)    |
+-------------↓----------------+
|      Service Layer            |
| Business Logic & Validation  |
+-------------↓----------------+
|      Data Access Layer        |
| JPA / Hibernate / MySQL     |
+-------------↓---------------+
|      Database (MySQL)        |
+-----------------------------+
          ↑
          |
      (Future)
  Cloud Storage / AI Service
```

4. System Components

4.1 Frontend (React.js)

- Developed using React.js and JavaScript.
- Uses **Axios** for API communication.
- Includes UI modules:
  - Dashboard
  - Add/Edit Transaction
  - Goal Tracker
  - Budget Planner
  - Analytics Charts (using Recharts or Chart.js)
  - Authentication Pages (Login/Signup with 2FA)

4.2 Backend (Spring Boot)

- Developed with **Spring Boot 3.x** and **Java 17**.
- Organized into standard layers:
  - **Controller Layer:** Handles REST APIs.
  - **Service Layer:** Contains business logic (e.g., goal progress calculation, recurring transaction scheduler).
  - **Repository Layer:** Uses Spring Data JPA for database operations.

Key Backend Modules:

Module	Description
User Module	Handles registration, authentication, JWT, and 2FA verification.

Transaction Module	Manages expense/income entries and categorization.
Budget Module	Enables users to set monthly or custom budgets and track usage.
Goal Module	Allows users to define goals, allocate savings, and view progress.
Recurring Module	Automates repetitive transactions based on user-defined frequency.
Notification Module	Sends alerts and reminders via email or dashboard.
Analytics Module	Provides spending insights and category-wise reports.
AI Module (Planned)	Uses predictive models for financial recommendations.
Cloud Sync (Planned)	Syncs user data across devices securely.

5. Database Design

5.1 Logical Data Model

The system uses a **relational database (MySQL)** with entity relationships as follows:

ER Diagram (simplified textual view):

```
User
 ├── user_id (PK)
 ├── name
 ├── email
 ├── password_hash
 ├── two_factor_enabled
 └── role

Transaction
 ├── transaction_id (PK)
 ├── user_id (FK)
 ├── category
 ├── amount
 ├── type (income/expense)
 ├── date
 ├── recurring_id (FK)
 └── notes
```

```
Budget
 ├── budget_id (PK)
 ├── user_id (FK)
 ├── month
 ├── limit_amount
 ├── spent_amount

Goal
 ├── goal_id (PK)
 ├── user_id (FK)
 ├── goal_name
 ├── target_amount
 ├── saved_amount
 ├── target_date

Recurring_Transaction
 ├── recurring_id (PK)
 ├── user_id (FK)
 ├── frequency
 ├── next_execution_date

Notification
 ├── notification_id (PK)
 ├── user_id (FK)
 ├── message
 ├── status (read/unread)
 ├── created_at
```

6. Data Flow Design

6.1 Data Flow Diagram (Level 0 – Context Diagram)

```
        +------------------------+
        |         User           |
        +-----------+------------+
                    |
                   v
        +-----------------------+
        |     BudgetIQ App      |
        | (React + Spring Boot) |
        +----------+------------+
                    |
                    v
        +-----------------------+
        |    Database (MySQL)   |
        +-----------------------+
```

6.2 Level 1 DFD (Expense Tracking Example)

```
User → [Add Transaction Form] → [Transaction Controller] → [Service
Layer] → [Database]
```

7. Security Design

- **Authentication:** JWT-based secure login.
- **Two-Factor Authentication (2FA):**
  Optional OTP-based verification during login.
- **Authorization:** Role-based access control (user/admin).
- **Data Security:**
  - HTTPS communication
  - Passwords encrypted using **BCrypt**
  - Sensitive data stored securely
- **Session Management:** Token expiration and refresh mechanisms.

8. Integration Design

- **Frontend–Backend Integration:** RESTful APIs (JSON format).
- **Email Service Integration:** For notifications and 2FA verification.
- **Planned External Integrations:**

    - OpenAI API (AI-based insights)
    - Cloud Storage Service (AWS S3 / Firebase)
    - Banking APIs (for automatic transaction import).

9. Error Handling & Logging

- All exceptions handled via a **Global Exception Handler**.
- Errors categorized as:
    - User Input Errors (validation issues)
    - Server Errors (database or logic failure)
    - External Service Errors (email/AI API)
- Logging using **Spring Boot's SLF4J + Logback**.

10. Deployment Design

10.1 Development Environment

- IDE: IntelliJ IDEA / VS Code
- Tools: Maven, Node.js, npm
- Local DB: MySQL

10.2 Deployment Environment

- Hosting: AWS EC2 (planned)
- Build Automation: GitHub Actions or Jenkins
- Containerization (Optional): Docker for microservice setup

11. Future Enhancements

- **AI Financial Advisor:** Generate personalized tips and forecasts.
- **Cloud Sync:** Multi-device data synchronization.

- **Mobile App:** Native Android/iOS versions.
- **Dark Mode and Custom Themes:** Enhanced UI flexibility.
- **Multi-language Support:** Expand accessibility.

12. Appendix

- **Version:** 1.0 (Web-based MVP)
- **Next Version:** 2.0 (AI + Cloud Integration)
- **Author:** Krishna Bhatt