Personal Finance Management System

1.Project Overview

This project is a Personal Finance Management System designed to empower users in taking control of their financial lives by providing an all-encompassing platform for managing their financial activities. The system offers a robust and intuitive solution that allows users to seamlessly track expenses, manage multiple accounts and balances, set and achieve financial goals, and efficiently handle recurring bills.

2.Scope

This document presents the design and architecture of the Personal Finance Management System. The application aims to empower users to take control of their finances by providing tools for expense tracking, balance monitoring, transaction management, bills handling, and goal setting. The focus is on delivering both functional and non-functional requirements, supported by a thorough overview of the technologies used and the system's core components and interactions.

3. Functional Requirements

3.1 Overview:

Financial Snapshot: Provides a summary of key information from all tabs:

- **Balance**: Total balance across accounts.
- **Transactions**: Recent transactions.
- **Bills**: Upcoming bills.
- **Expenses**: 12-month credit and debit summary with expense breakdown.
- **Goals**: Users goals with planned spending amounts

3.2 Balance

- Manages different accounts of the user

3.3 Expense

- view expense records with categorization.

3.4 Transactions

- Lists recent transactions with account name, payment type and amount

3.5 Bills

- List all Upcoming bills

3.6 Goals

- Enable users can set an expense goal and the amount to be spent in a particular category

4. Non-Functional Requirements

4.1 Performance

- The application should load within 3 seconds for the Overview tab and display updated data in real-time.
- Transactions and balance updates should reflect changes within 5 seconds.

4.2 Scalability

- The system should be able to handle an increasing number of users and transactions without significant degradation in performance.
- It should support adding new features or scaling up to accommodate additional data or functionalities in the future.

4.3 Reliability

- It should handle user data and transactions reliably without data loss or corruption.

4.4 Usability

- The user interface should be intuitive and easy to navigate, with clear labels and instructions.

4.5 Security

- The application must implement robust security measures to protect sensitive financial data from unauthorized access and breaches.
- User authentication and authorization should be handled securely, with encrypted passwords and secure session management.

4.6 Compatibility

- The system should be compatible with major web browsers (e.g., Chrome, Firefox, Safari).

5. Technology Used

5.1 Frontend

- **React:** JavaScript library for building user interfaces.
- **HTML:** Standard markup language for structuring web content.
- **CSS:** Stylesheet language for designing and layout of web pages.
- **JavaScript:** Programming language for dynamic content and interactivity.

5.2 Backend

- **Java Spring Boot:** Framework for building production-ready applications with Java.
- **MySQL:** Relational database management system for storing and managing data.

5.3 DevOps and Deployment

- **Docker** for containerization.
- **Kubernetes** for orchestration and scaling.
- **AWS** for hosting, database management, and other cloud services.

6. High-Level Design

6.1Architecture Overview

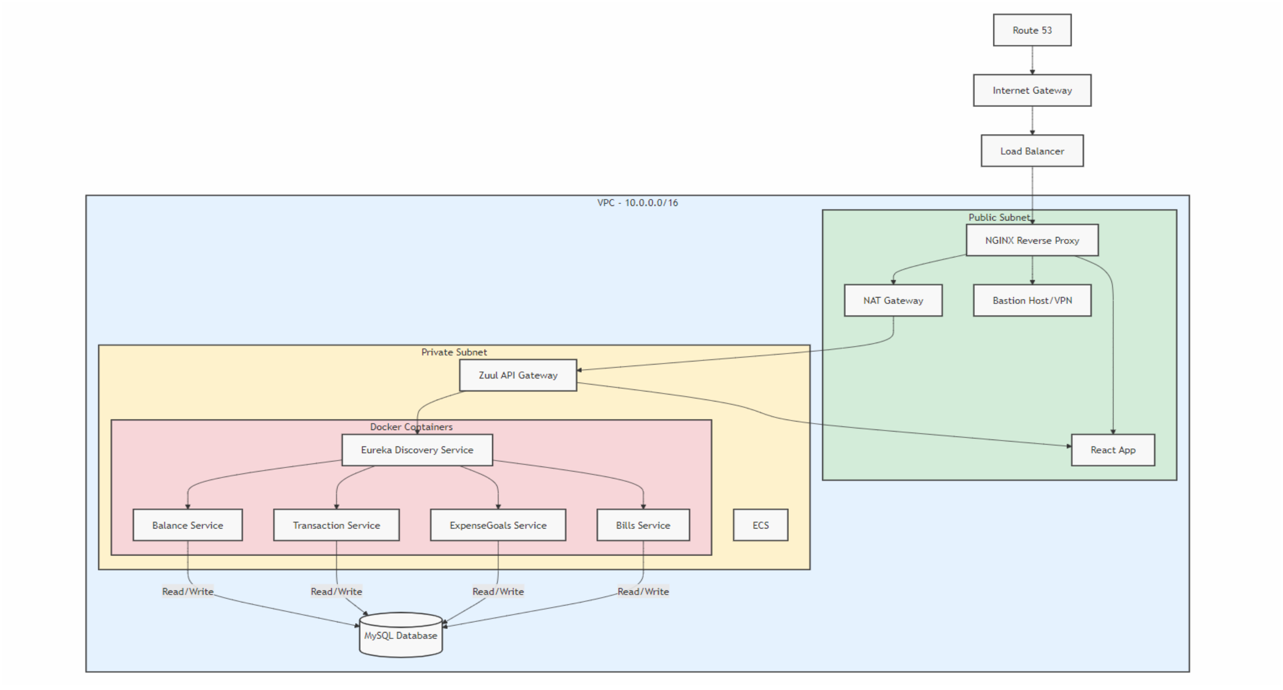
The Personal Finance Management System follows a multi-tier architecture with the following layers:

1. **Presentation Layer:** React and Bootstrap for a responsive UI with tabs for Overview, Balance, Expense, Goals, Bills, and Transactions.

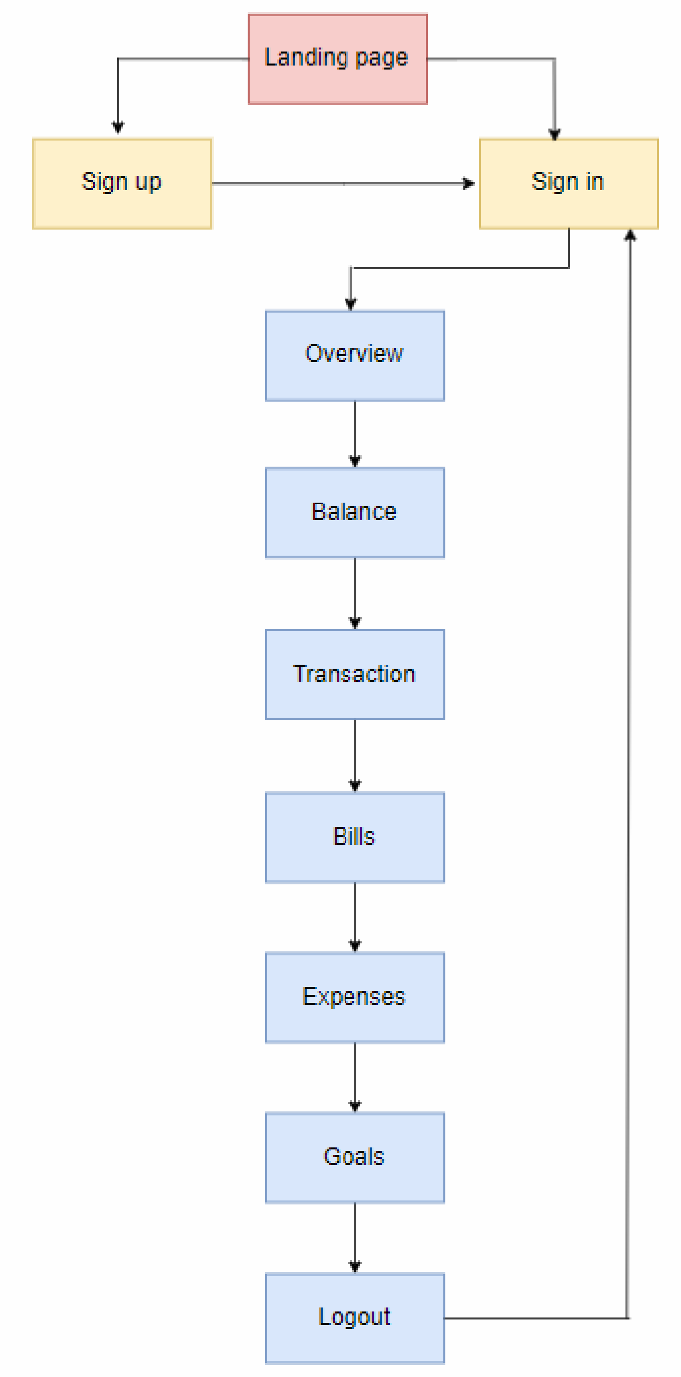
2. **Application Layer:** Java Spring Boot handles business logic, API requests, and user authentication.

3.Data Layer: Relational Database stores user details, account details, transactions, expensegoals, and bills ensuring data integrity and efficient querying.

Below is the Architecture Diagram of our application:



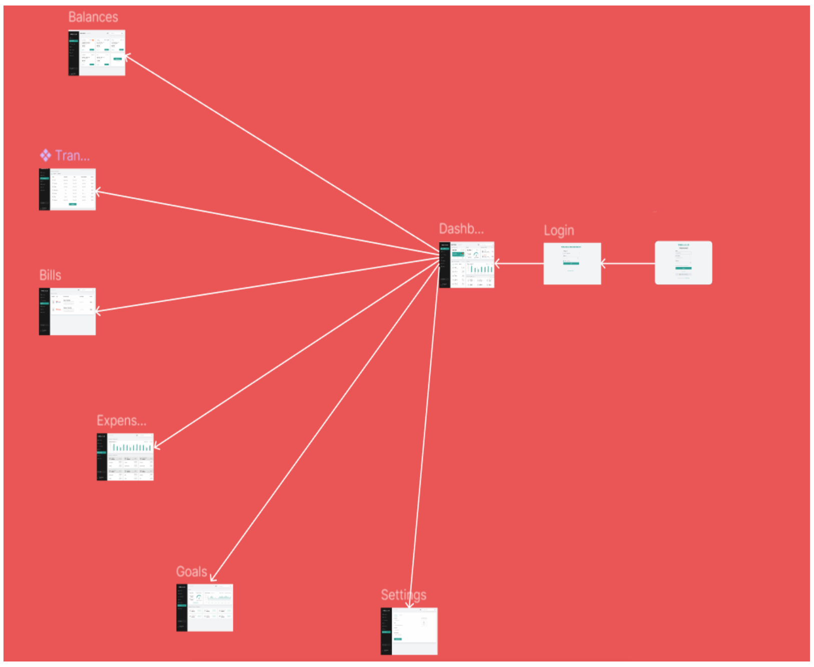
Below is the flowchart diagram of our application:



6.2 Functional Specifications

- **Overview:** Shows total balance, transactions, bills, expenses, goals, and a monthly comparison graph.
- **Balance:** Lists account cards with balance details and an option to add new cards.
- **Transactions:** Lists recent transactions with details
- **Bills:** Manages recurring bills with add, edit, delete options, reminders, and history.
- **Expense:** Displays expense records with categories and spending insights.
- **Goals:** Manages saving and expense goals, tracking progress and targets.

6.2 System Interactions (Wireframe Diagrams)

- **Figma Wireframes:**
    - Design and demonstrate user interfaces, including balance display, expense management, goal tracking, bill handling, and transaction details.
    - 

6.4 Performance Considerations

- **Performance Benchmarks:**
  - Average response times for loading tabs and updating data.
  - Maximum concurrent users supported by the system.
  - Latency thresholds for real-time updates and transactions.
- **Load Testing Results and Optimization Strategies:**
  - Results from load testing to evaluate system performance under various conditions.
  - Optimization strategies including caching mechanisms, database indexing, and efficient query design to enhance performance and scalability.

6.5 Technology Stack

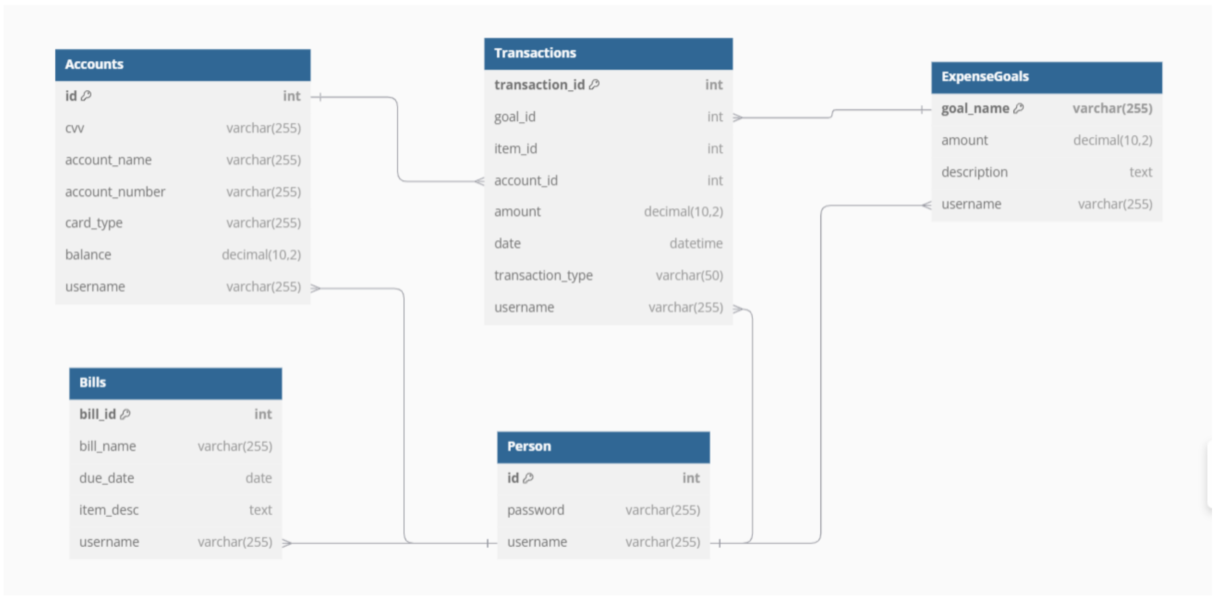
- **Front-End:** ReactJS, HTML5, CSS3
- **Back-End:** Java Spring Boot
- **Database:** MYSQL
- **Deployment:** Docker, Kubernetes, AWS

7. Low-Level Design

7.1 Database Diagrams:

The databases for the Personal Finance Management System includes:

1. **Accounts:** Stores account details such as account name, number, CVV, and balance.
2. **Transaction:** Logs all financial transactions, capturing details like date, amount, type, and description.
3. **Expense Goals:** Stores the amount the user wants to spend in specific categories, helping users manage and limit their spending in each category.
4. **Bills:** Keeps records of recurring or one-time bills, including due dates, amounts, and payment status.
5. **Person:** Contains the username and password for authentication



7.2 Detailed Component Specification

Overview Module:

- **Classes:** Overview, Balance, Transaction, Goal, Bill
- **Functions:** getTotalBalance(), getRecentTransactions(), getUpcomingBills(), getCurrentExpenses(), getFinancialGoals(), generateComparisonGraph()

Account Module:

- **Classes:** Balance
- **Functions:** viewCardDetails(), addNewCard(),editCard()

Expense Module:

- **Classes:** ExpenseGoals
- **Functions:** addExpense(), getExpenseByCategory()

Bills Module:

- **Classes:** Bill
- **Functions:** addBill(), editBill(), deleteBill()

Transactions Module:

- **Classes:** Transaction
- **Functions:** recordTransaction(), getTransactionHistory(), filterTransactions(),

7.3 Interface Definitions

- **Overview Interface**: Displays total balance, recent transactions, upcoming bills, past 12 months expenses and credits with a comparison graph and a expense breakdown in each category
- **Balance Interface**: Enable users to add, view and edit their card details
- **Expense Interface**: Displays the Past twelve months debit vs credit graph and show the expense breakdown in each category
- **Goals Interface**: Enable Users to set goal and amount
- **Bills Interface**: Displays the upcoming bills and their due dates
- **Transactions Interface**: Displays the transactions of a user.

7.4 Resource Allocation

- **Memory and Storage Allocation**: Estimate memory needs and allocate sufficient RAM to handle peak loads. Ensure ample storage for user data, with provisions for growth and backups.
- **Processing Power**: Define CPU requirements based on user load and data complexity, ensuring enough processing power for real-time updates and calculations.

7.5 Error Handling and Recovery

- **Error Logging:** Implement a centralized system to capture and track errors effectively.
- **Automatic Recovery:** Ensure system resilience with failover mechanisms and automated recovery procedures.
- **User Feedback:** Provide clear error messages and recovery options, along with support channels to assist users.

7.6 Security Considerations

- **Authentication:** Use JWTs (JSON Web Tokens) or OAuth for secure session management. Ensure robust user authentication and session handling to protect user accounts.
- **Data Protection**: Encrypt sensitive data at rest and in transit.
- **Security Audits:** Schedule regular security audits and penetration testing