# SOFTWARE REQUIREMENTS SPECIFICATIONS (SRS)

ONLINE BANKING DASHBOARD (MOCK VERSION)

**Prepared by:** Poojaa D V  
**Date:** 30 July 2025

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to define the software requirements for an Online Banking Dashboard. This dashboard provides users with a mock experience of internet banking, allowing simulated operations such as viewing balances, transaction history, and mock fund transfers. It is intended for educational and demonstrative purposes only—no real financial data or transactions are involved.

**1.2 Scope**

This application enables users to:

* Log in securely with mock credentials
* View account summaries and balances
* Review mock transaction history
* Simulate fund transfers between mock accounts  
  The front-end is developed using React for a responsive and interactive UI, while the backend is built with Spring Boot to manage REST APIs and business logic.

**1.3 Intended Audience**

* Instructors and peers evaluating software projects
* Developers and testers involved in implementation
* System designers and UI/UX designers
* Students or individuals seeking to understand full-stack banking systems

**1.4 Definitions & Acronyms**

* **API:** Application Programming Interface
* **UI:** User Interface
* **DB:** Database
* **SRS:** Software Requirements Specification

**2. General Description:**

**Product Overview**

The Online Banking Dashboard is a stand-alone web app using a client-server model. The frontend is built with React.js, and the backend uses Spring Boot (Java). It uses mock data stored in-memory or JSON files, with no real bank connections.

**Users and Environment**

Designed for users familiar with basic internet banking, the app runs on modern browsers like Chrome and Firefox. It requires internet access to connect to backend APIs but does not link to actual banks or third parties.

**Technical Constraints**

The system relies on mock data and follows strict frontend-backend separation. It must use React.js and Spring Boot, communicate via RESTful JSON APIs, and run in compatible browser environments.

**3. Problem Statement**

**Current Scenario:**  
Traditional online banking platforms are complex and not beginner-friendly for software developers or students wishing to explore their architecture.

**Problem:**  
There is a need for a simplified, mock version of an online banking system that allows users to simulate key operations and explore the front-end/backend integration without security or data concerns.

**Proposed Solution:**  
Develop a lightweight, mock banking dashboard using React and Spring Boot to simulate the user experience and backend logic of a full-fledged banking system.

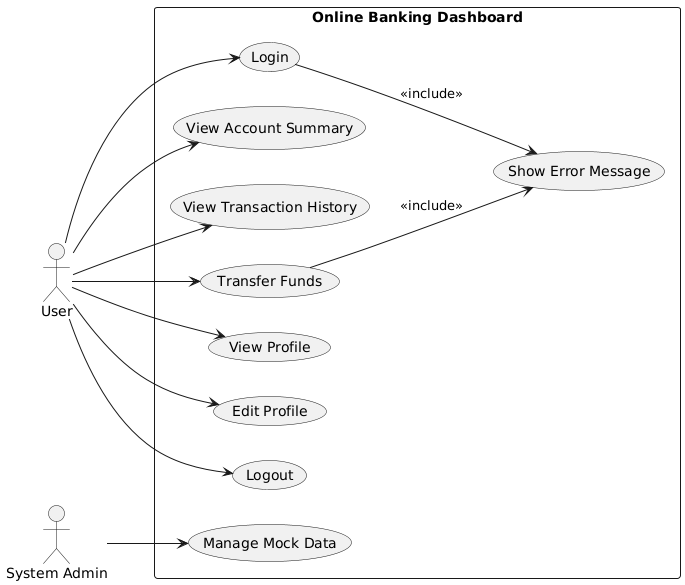
**4. Functional Requirements**

* **User Login:** The system should allow users to sign in using predefined mock credentials.
* **Account Overview:** After login, users should see their account balance and name on the dashboard.
* **Transaction History:** Users should be able to view a list of mock transactions associated with their account.
* **Fund Transfer Simulation:** The system should allow users to simulate money transfers between fake accounts.
* **User Logout:** A logout option should be available to end the session securely.
* **Error Handling:** Clear error messages should appear when login credentials are incorrect.

**5. Non-Functional Requirements**

* **Responsive Design:** The UI must adapt seamlessly to desktops, tablets, and mobile phones.
* **Fast Load Times:** The dashboard should load in under 3 seconds on a standard internet connection.
* **No Real Data:** All data in the system should be mock only — no integration with real banking systems.
* **Clean Code Standards:** The app should follow modular, readable code practices with proper documentation.
* **Security Awareness:** Even in mock mode, login mechanisms should simulate secure sessions and basic encryption.
* **RESTful APIs:** Backend should expose APIs that conform to REST principles and respond with JSON data.

**6. Use Case Diagram**





**7. Design Constraints**

* **Frontend Technology:**  
  The user interface must be developed using **React.js**, leveraging its component-based architecture and state management for building an interactive and responsive UI.
* **Backend Framework:**  
  The backend services must be implemented using **Spring Boot (Java)**, chosen for its RESTful API support, lightweight nature, and easy integration with frontend frameworks.
* **Database Setup:**  
  The application must use an **H2 in-memory database** or **mock JSON files** to simulate data storage. No real or persistent database should be used, as the system is for demonstration only.
* **System Architecture:**  
  The project must follow a **client-server model**, with clear separation between the frontend (React) and backend (Spring Boot APIs). Communication should occur over HTTP.
* **API Communication Format:**  
  The frontend must communicate with the backend using **HTTP requests** via **Axios** or **Fetch**, and the backend must return data in **JSON format**.
* **Version Control:**  
  The project must use **Git** for source control, with code hosted on **GitHub**. Proper commit messages, branching practices, and code organization are required.
* **Development Tools:**
  + **Frontend:** Developed using **Visual Studio Code (VS Code)**.
  + **Backend:** Developed using **IntelliJ IDEA** or **Spring Tool Suite (STS)**.
  + **API Testing:** Backend APIs should be tested using **Postman** or equivalent REST clients.