**SOFTWARE DESIGN DOCUMENT(SDD)**

**Online Banking Dashboard (Mock Version)**

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

**1. Introduction**

**1.1 Purpose**

This document details the software design specification for the Online Banking Dashboard, a mock web application that simulates basic internet banking operations such as viewing account balances, transaction history, and fund transfers. The application is intended for educational and demonstration purposes, providing users with a realistic but fully simulated banking experience using mock data.

**1.2 System Overview**

**Key Features**

* Secure user login with predefined mock credentials
* Account summary displaying balance and account holder details
* Viewable transaction history with mock data
* Simulated fund transfer functionality between mock accounts
* User profile viewing and editing capabilities
* Responsive user interface accessible on desktop and mobile

**Architecture Design**

The system follows a client-server architecture consisting of:

* **Frontend:** Developed using React.js to provide a dynamic, responsive user interface
* **Backend:** Built with Spring Boot (Java), exposing RESTful APIs for business logic and data handling
* **Data Layer:** Uses mock data stored in-memory or in JSON files; no real banking data or external integrations
* Communication between frontend and backend occurs via REST APIs exchanging JSON data.

**2. Interface Design**

**2.1 API Interface Specification**

The backend exposes RESTful APIs to the frontend for key functionalities:

| **Operation** | **Method** | **Endpoint** | **Description** |
| --- | --- | --- | --- |
| User Login | POST | /api/auth/login | Authenticates user credentials |
| Get Account Summary | GET | /api/accounts/{id} | Fetches account summary |
| Get Transaction History | GET | /api/accounts/{id}/transactions | Retrieves transaction history |
| Fund Transfer | POST | /api/transfer | Performs a mock fund transfer |
| Get User Profile | GET | /api/profile/{userId} | Fetches user profile details |
| Update User Profile | PUT | /api/profile/{userId} | Updates user profile |

All API responses follow JSON format and appropriate HTTP status codes are returned to indicate success or error.

**2.2 Technology Stack**

* **Frontend:** React.js with Axios for API calls, CSS for styling, React Router for navigation
* **Backend:** Spring Boot (Java), Spring Security (mock authentication), Spring Data (mock repositories)
* **Development Tools:** VS Code (frontend), IntelliJ IDEA or Spring Tool Suite (backend)
* **Version Control:** Git and GitHub for source management
* **Testing:** Postman for API testing, Jest/React Testing Library for frontend unit tests

**2.3 Security Requirements**

* Mock authentication implemented to simulate secure login (no real credential storage)
* Session handling through JWT-like tokens for frontend session management
* API endpoints validate request payloads and reject malformed or unauthorized requests
* HTTPS usage recommended in deployment environments
* Security practices aim to mirror real banking app flows without actual sensitive data

**3. General Description**

**3.1 Problem Statement**

Current online banking applications are complex, making it difficult for learners and developers to explore their architecture or test features safely. There is a need for a simplified mock version of a banking dashboard that simulates core functionalities, enabling hands-on experience with frontend-backend integration in a risk-free environment.

**3.2 Functional Requirements**

* User login/logout with mock credentials
* Display of account summary and balance
* Viewing of transaction history (mock data)
* Simulated fund transfer between accounts
* Profile viewing and editing
* Display of error messages on invalid actions

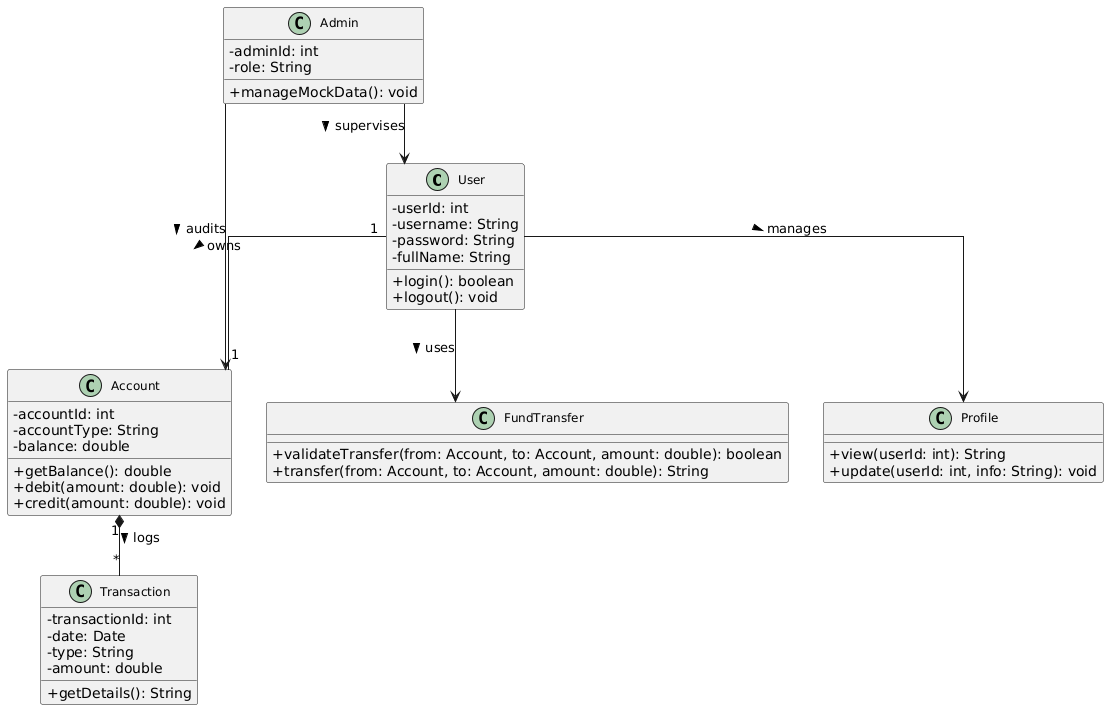
**3.3 Non-Functional Requirements**

* Responsive UI supporting desktop and mobile devices
* Application loads within 3 seconds on standard internet
* Use of mock data exclusively, no real banking integration
* Clean, modular, and well-documented codebase
* RESTful API design returning JSON responses
* Basic simulated security practices implemented

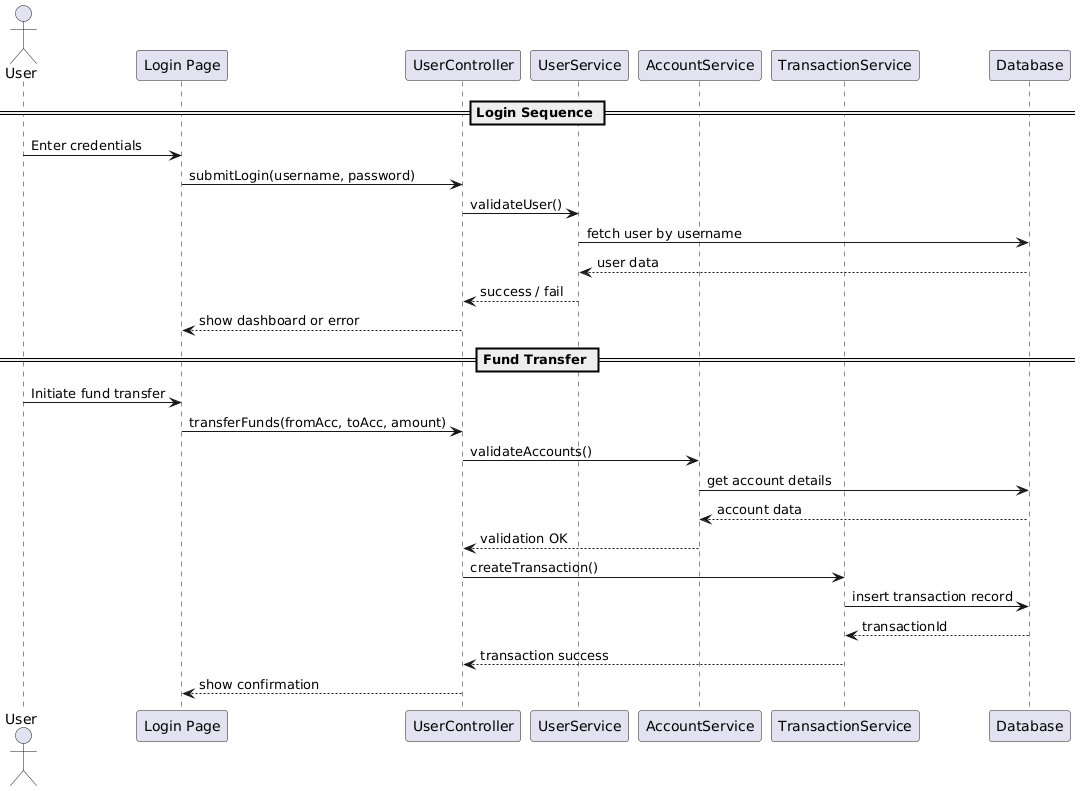
**3.4 Design Constraints**

* Frontend strictly implemented in React.js
* Backend developed with Spring Boot and Java
* Communication over HTTP using RESTful APIs
* Mock data stored in-memory or JSON files, no persistent database
* Version control maintained with GitHub
* API testing performed with Postman or equivalent tools

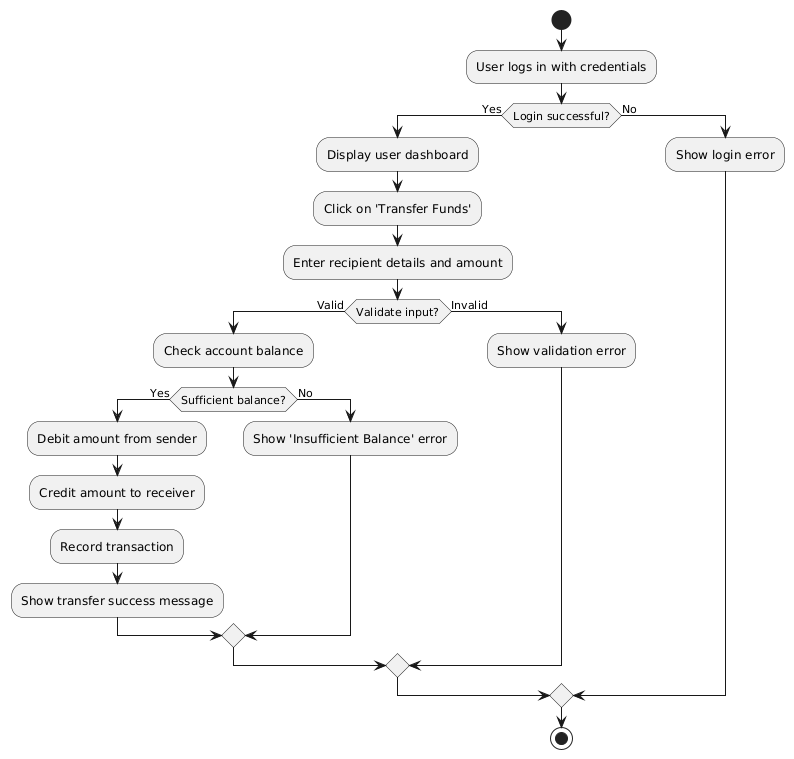
**4. Class Diagram**



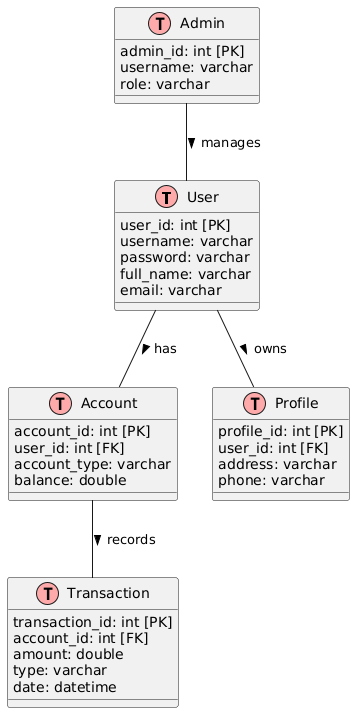
**5. Sequential Diagram**



**5. Activity Diagram**



**6. E-R Diagram**



**7.Use Case Diagram**

