Banking SRS

Software Requirements Specification

Revision History

Date	Revision	Description	Author
mm/dd/yyyy	1.0	Initial Version	Your Name

Table of Contents

1. Purpose

This document outlines the requirements for the Banking System.

1.1. Scope

This document will describe the requirements and overall structure of the Banking System. It will define the system's functionality, user types, and constraints, and serve as a reference for the design and implementation of the system.

1.2. Definitions, Acronyms, Abbreviations

- **ATM (Automated Teller Machine)** – A system that allows users to perform basic banking transactions.
- **Authentication** – The process of verifying a user's identity.
- **Authorization** – The process of determining whether a user is permitted to perform an action.
- **Balance** – The amount of money currently available in a bank account.
- **Customer** – A user who owns one or more bank accounts.
- **Employee** – A bank staff member with administrative privileges.
- **PIN (Personal Identification Number)** – A numeric code used for secure user authentication.
- **Transaction** – Any operation that accesses or modifies account data.
- **Transfer** – The movement of funds from one account to another.

1.3. References

- Course lecture notes
- Use Case Specification Document
- UML Use Case Diagrams
- Class Diagrams
- Sequence Diagrams

1.4. Overview

The Banking System is designed to manage and process basic financial operations for a bank. It allows customers to securely access their accounts, perform transactions such as deposits, withdrawals, and transfers, and view account information. Because financial data must be accurate and secure, the system emphasizes controlled access, transaction integrity, and clear separation of user roles such as customers, employees, and ATM users.

2. Overall Description

1.5. Product Perspective

The Banking System is a Java-based application designed to simulate core banking operations using the client-server architecture. The system communicates over TCP/IP, allowing multiple clients to connect to a centralized server that's responsible for managing accounts, transactions, and user authentication. The system is modular in design, with components such as account management, transaction processing, ATM services, and employee management handled on the server side.

2.1. Product Architecture

The system will be organized into _4__ major modules: the _Account Management__ module, the _Transaction Processing__ module, the ___ATM Services__ module, and the __Employee Management___ module.

Note: System architecture should follow standard OO design practices.

2.2. Product Functionality/Features

The high-level features of the system are as follows (see section 3 of this document for more detailed requirements that address these features):

- Support client–server communication over TCP/IP
- Authenticate customers and employees using credentials such as passwords or PINs
- Allow customers to create and access bank accounts
- Enable deposits, withdrawals, and transfers between accounts
- Provide ATM-style access for basic banking transactions
- Allow employees to perform administrative account management tasks
- Enforce role-based access control for customers, employees, and ATM users
- Maintain accurate account balances and transaction records
- Support multiple clients connected to the server concurrently

1.6. Constraints

- SR1 The system shall be implemented using the Java programming language.
- SR2 The system shall use TCP/IP socket-based communication for client–server interactions.
- SR3 The system shall follow standard object-oriented design principles.
- SR4 The server shall be responsible for processing all banking logic and maintaining account data.
- SR5 Clients shall not directly modify account data and shall communicate with the server only through defined requests.
- SR6 The server shall support multiple concurrent client connections.

1.7. Assumptions and Dependencies

- It is assumed that the system will be used for academic purposes only and not for real financial transactions.
- It is assumed that clients and the server will run on machines that support the Java Runtime Environment.
- It is assumed that users will have a stable network connection when interacting with the system.
- It is assumed that the number of concurrent clients connected to the server will be limited to a manageable size appropriate for a classroom project.
- It is assumed that users will provide valid input and follow intended usage of the system.

3. Specific Requirements

3.1. Functional Requirements

3.1.1. Common Requirements:

Provide requirements that apply to all components as appropriate.

Example:

3.1.1.1 Users should be allowed to log in using their issued id and pin, both of which are alphanumeric strings between 6 and 20 characters in length.

3.1.1.2 The system should provide HTML-based help pages on each screen that describe the purpose of each function within the system.

3.1.2. _____ Module Requirements:

Provide module specific requirements as appropriate.

Example:

3.1.2.1 Users should be allowed to log in using their issued id and pin, both of which are alphanumeric strings between 6 and 20 characters in length.

3.1.3. _____ Module Requirements:

Provide module specific requirements as appropriate.

Example:

3.1.2.1 Users should be allowed to log in using their issued id and pin, both of which are alphanumeric strings between 6 and 20 characters in length.

3.1.4. _____ Module Requirements:

Provide module specific requirements as appropriate.

Example:

3.1.2.1 Users should be allowed to log in using their issued id and pin, both of which are alphanumeric strings between 6 and 20 characters in length.

3.2. External Interface Requirements

Provide module specific requirements as appropriate.

Example:

3.2.1 The system must provide an interface to the University billing system administered by the Bursar's office so that students can be automatically billed for the courses in which they have enrolled. The interface is to be in a comma-separated text file containing the following fields: student id, course id, term id, action. Where "action" is whether the student has added or dropped the course. The file will be exported nightly and will contain new transactions only.

3.3. Internal Interface Requirements

Provide module specific requirements as appropriate.

Example:

3.3.1 The system must process a data-feed from the grading system such that student grades are stored along with the historical student course enrolments. Data feed will be in the form of a comma-separated interface file that is exported from the grading system nightly.
3.3.2 The system must process a data-feed from the University billing system that contains new student records. The feed will be in the form of a comma-separated text file and will be exported from the billing system nightly with new student records. The fields included in the file are student name, student id, and student pin number.

4. Non-Functional Requirements

4.1. Security and Privacy Requirements

Example:
4.1.1 The System must encrypt data being transmitted over the Internet.

4.2. Environmental Requirements

Example:
4.2.1 System cannot require that any software other than a web browser be installed on user computers.
4.2.2 System must make use of the University's existing Oracle 9i implementation for its database.
4.2.3 System must be deployed on existing Linux-based server infrastructure.

4.3. Performance Requirements

Example:
4.3.1 System must render all UI pages in no more than 9 seconds for dynamic pages. Static pages (HTML-only) must be rendered in less than 3 seconds.