SOFTWARE REQUIREMENTS SPECIFICATION – BANKING APPLICATION

VERSION 1.0

APPROVAL SIGNATURES

CONTRACTOR		DATE
<<Deliverable Owner>> <<John Doe, Manager>>	<<Signature>>	
<<Vendor Project Manager>>	<<Signature>>	
<<Vendor Managing Director>>	<<Signature>>	

PROJECT MANAGER			
Date:	12.11.2024	To:	**Venugopalareddy G S,** Project Manager
☐	I approve this deliverable and have no further questions or comments.		
☐	I approve this deliverable conditionally, contingent on the review and approval of the following corrections (see comments).		
☐	I reject this deliverable for the following reasons identified (see comments).		
<<SIGNATURE>>		**<<DATE>>**	
Comments			

DOCUMENT HISTORY

DOCUMENT APPROVAL HISTORY	
Prepared By	
Reviewed By	
Approved By	

DOCUMENT REVISION HISTORY			
DATE	DOCUMENT VERSION	REVISION DESCRIPTION	AUTHOR
02/04/2025	1.0	Initial Version	

TABLE OF CONTENTS

1. INTRODUCTION

1.1 PURPOSE OF THIS DOCUMENT

The purpose of this Requirements Specification document is to clearly define and outline the functional and non-functional requirements for the development of the **Banking Software** application. This document serves as a foundation for both the development and validation processes, ensuring that all stakeholders, including developers, testers, and end-users, have a shared understanding of the project's objectives. By detailing specific requirements, expectations, and constraints of the system, this document aims to minimize any ambiguity and set a clear roadmap for the design, development, and implementation of the application. It will also act as a reference point throughout the project's lifecycle to ensure that the delivered product aligns with the defined goals.

1.2 SCOPE OF THIS DOCUMENT

This document encompasses all the essential details needed to develop the **Banking Software** application. It covers functionalities including user authentication, account management, transaction processing, and administrative control. The scope also includes security measures, data integrity management, and user accessibility features. Additionally, the document highlights the expected development timeline, resource allocation, and associated costs to deliver the product to completion. By thoroughly addressing both technical and business considerations, this document will ensure that the project stays within budget, meets deadlines, and satisfies end-user needs.

1.3 OVERVIEW

The **Banking Software** application is designed to provide a user-friendly and secure platform for managing banking operations. Users will be able to create accounts, log in, check balances, transfer money, and receive funds. Additionally, an administrator will have control over user account creation, deletion, balance management, and password changes. The system aims to be reliable, secure, and easy to use for both customers and banking staff.

2. GENERAL DESCRIPTION

The **Banking Software** application is a secure and user-friendly platform for handling banking operations. The primary functions include account management, secure login, money transfers, and administrative controls. Key features include:

- **User Account Management:** Users can create and manage their banking accounts.
- **Authentication & Security:** Secure login with username and password.
- **Money Transactions:** Users can transfer and receive funds securely.
- **Admin Control:** An admin can create/delete users, assign account numbers, credit/debit accounts, and reset passwords.
- **Audit Logging:** All transactions and administrative actions are logged for security and tracking purposes.

The overall goal is to ensure that the application provides a seamless, efficient, and secure banking experience for users and administrators alike.

---

3. FUNCTIONAL REQUIREMENTS

3.1 USER MANAGEMENT

User Registration: Users should be able to register with their name, mobile number, email, and create a password.

User Authentication: Secure login with username and password.

View Account Details: Users should be able to check their account balance and transaction history.

Transfer Money: Users should be able to transfer funds to another account securely.

Receive Money: Users should be able to receive funds from other users.

3.2 ADMIN MANAGEMENT

User Creation/Deletion: Admin can create and delete user accounts.

Assign Account Numbers: Admin can assign a unique account number to each user.

Credit/Debit Accounts: Admin can add or remove funds from any user's account.

Reset Passwords: Admin can reset a user's password if requested.

3.3 TRANSACTION MANAGEMENT

Fund Transfers: Users should be able to transfer money between accounts.

Transaction History: Users should be able to view past transactions.

Audit Logging: All transactions should be recorded for security and monitoring.

---

4. INTERFACE REQUIREMENTS

4.1 USER INTERFACE (UI) REQUIREMENTS

The application should have a clean, intuitive UI for seamless navigation.

Users should be able to view their account details and transaction history easily.

Admin should have a separate interface for managing user accounts and transactions.

4.2 EXTERNAL INTERFACE REQUIREMENTS

Bank Database: The application should integrate with a secure database to store user data and transactions.

Email Notification System: Users should receive email notifications for significant transactions or security changes.

---

5. PERFORMANCE REQUIREMENTS

5.1 RESPONSE TIME

Login and Authentication: Should be completed within **2 seconds**.

Transaction Processing: Should not exceed **5 seconds**.

Account Balance Retrieval: Should be displayed within **1 second**.

5.2 SYSTEM RESOURCE UTILIZATION

The application should maintain efficient CPU and memory usage to ensure smooth operation.

- It should function properly under normal banking traffic without performance degradation.

---

6. DESIGN CONSTRAINTS

6.1 SECURITY REQUIREMENTS

Encryption: All sensitive data, including passwords and transactions, should be encrypted.

Multi-Factor Authentication (MFA): Additional security measures should be implemented for login and transactions.

Role-Based Access Control: Only the admin can manage accounts and alter balances.

6.2 RELIABILITY AND AVAILABILITY

System Uptime: The application should ensure **99.9% uptime**.

Error Handling: In case of failures, the system should display appropriate error messages and allow for error recovery.

Data Integrity: Transactional consistency should be maintained at all times.

6.3 SCALABILITY

The system should be scalable to handle **thousands of users and transactions per minute** without performance degradation.

7. SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC) STAGES

7.1 REQUIREMENTS GATHERING

Collect user and admin requirements for the banking application.

Define security, scalability, and performance needs.

7.2 SYSTEM DESIGN

Define the database schema and system architecture.

Create UI/UX design prototypes.

7.3 IMPLEMENTATION

Develop frontend and backend components.

Implement database interactions and API integrations.

7.4 TESTING

Conduct unit, integration, and system testing.

Ensure security compliance and performance benchmarks.

7.5 DEPLOYMENT

Deploy the system on a secure cloud or on-premise server.

Set up backup and monitoring solutions.

7.6 MAINTENANCE AND UPDATES

Monitor system performance and security.

Release periodic updates for new features and bug fixes.

---

8. HIGH LEVEL DESIGN

The Banking Application will be implemented in C and designed to run natively on Windows OS. The application will provide a **Command-Line Interface (CLI)** for users to perform banking operations, including account creation, balance inquiry, deposits, withdrawals, and transaction history. This high-level design outlines the architecture, key components, and overall system structure for implementing the Banking Application in C on Windows.

8.1 ARCHITECTURE OVERVIEW

The application will follow a **Modular Architecture** with separate layers for:

- **User Interface (CLI):** Command-line input/output handling.
- **Core Logic:** Banking operations and business logic.
- **Data Storage:** File-based storage for accounts and transactions.
- **File I/O:** For storing and retrieving banking data.
- **Utilities:** Common utility functions like input validation, searching, and logging.

The system will be entirely local, meaning no cloud synchronization or networking components are involved in this version of the Banking Application. All banking data will be stored in files (e.g., CSV files) on the local filesystem.

8.2 KEY COMPONENTS

8.2.1 USER INTERFACE (CLI)

The CLI will provide the interface through which users interact with the application. The main interface components include:

- **Main Menu:** Display options for various banking operations like creating an account, checking balance, making deposits, withdrawing money, viewing transaction history, and exiting the application.
- **Account Creation:**
    - Users can create a new account by entering details such as Name, Account Number, and Initial Deposit.
    - The system should generate a confirmation message with account details.
- **Balance Inquiry:**
    - Users can check their account balance by entering their account number.
    - The system will fetch and display the account balance from the stored data.
- **Deposit Money:**
    - Users can deposit money by entering the account number and deposit amount.
    - The updated balance should be saved and displayed to the user.
- **Withdraw Money:**
    - Users can withdraw money by entering the account number and withdrawal amount.
    - The system should ensure that the withdrawal does not exceed the available balance.
- **Transaction History:**
    - Users can view the transaction history of their account, including deposits and withdrawals.

Main Menu Options:

1. Create Account
2. Balance Inquiry
3. Deposit Money
4. Withdraw Money

5. View Transaction History
6. Exit

The CLI will display appropriate prompts, read user input, validate commands, and show results based on user actions.

8.2.2 CORE LOGIC

The core logic handles the processing behind each user action, including managing accounts, processing transactions, and updating balances.

- **Creating an Account:** Storing a new user's information in memory and saving it to a file.
- **Depositing Money:** Updating the account balance and recording the transaction.
- **Withdrawing Money:** Checking the balance before processing the withdrawal and updating the account.
- **Viewing Transaction History:** Retrieving and displaying past transactions from the data file.

These functions will interact with the **Data Storage Layer** to load, save, and update banking data.

8.2.3 DATA STORAGE

Since the application will be implemented in C, banking data will be stored in local files (e.g., CSV files), which is a simple and efficient way to persist data on Windows.

- **Data File Format (CSV):**
    - Accounts and transactions will be stored in CSV (Comma Separated Values) format.
    - Each record will include fields such as Account Number, Name, Balance, and Transaction History.

Example of CSV Format for Accounts:

1234567890, John Doe, 5000.00

9876543210, Alice Smith, 12000.00

- **File I/O Operations:**
    - **Load Accounts:** The program will load account details from a CSV file at startup.
    - **Save Transactions:** Changes to balances and transaction history will be saved back to the file.
    - **Data Persistence:** The program will ensure that all changes are correctly stored.

8.2.4 FILE I/O FOR TRANSACTIONS

- **Saving Transactions:**
    - Every deposit and withdrawal will be recorded in a transaction history file.
    - The file format will store the transaction type (Deposit/Withdrawal), amount, and timestamp.
- **Retrieving Transactions:**
    - Users can view their past transactions by retrieving records from the file.

8.2.5 UTILITIES AND HELPER FUNCTIONS

The application will use several utility functions to assist with input validation, error handling, and searching:

- **Input Validation:**
    - Ensure that account numbers are numeric and within a valid range.
    - Validate deposit and withdrawal amounts.
- **Transaction Searching:**
    - Implement a function to search transactions by date or amount.
- **File Management:**
    - Utility functions will handle reading from and writing to CSV files.

8.2.6 DATA FLOW

The data flow will follow these steps:

1. **Initialization:**
    - When the program starts, it will check if the account data file exists.
    - If it exists, the program loads account details into memory.
    - If it does not exist, the application initializes an empty account list.
2. **User Interaction:**
    - The user selects an operation from the menu.
    - The system processes the user input and invokes the appropriate function.
3. **Data Persistence:**
    - After any transaction (deposit/withdrawal), the updated data is saved to the file.
    - Transaction history is logged for future reference.

8.2.7 TECHNOLOGY AND TOOLS

- **Programming Language:** C
    - C will be used for its efficiency and control over system resources.
- **Windows OS:**
    - The program will be designed to run on Windows using standard libraries.
- **Libraries/Tools:**
    - Standard C libraries (stdio.h, stdlib.h, string.h) will be used for file operations and data handling.

8.2.8 ERROR HANDLING

The application should handle errors gracefully, such as:

- **Invalid User Input:**
    - Display error messages for incorrect account numbers or transaction amounts.
- **File Errors:**
    - Handle missing or corrupted files.
- **Memory Allocation Errors:**
    - Ensure the application does not crash due to memory issues.

Error messages should guide users on how to resolve issues effectively.