

coachGrasin

CPD

Software Requirements Specification – BANK MANAGEMENT SYSTEM 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: | **Nisarg Nirmalkumar,** 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 |
| 12/11/2024 | 1.0 | Initial Version |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

TABLE OF CONTENTS

1. Introduction 1

1.1. Purpose of this Document 1

1.2. Scope of this Document 1

1.3. Overview 1

2. GenERAL Description 2

2.1. PLANNING 3

2.2. FEASABILITY STUDY 3

2.3. Design 3

2.4. Development (Implementation) 4

2.5. References 4

3. ROLES AND RESPONSIBILITIES 4

3.1. Project Executive Steering Committee 5

3.1.1. ROLE 5

3.1.2. RESPONSIBILITY 5

3.2. Project Sponsors 5

3.2.1. Project Executive Sponsor 5

3.2.1.1 ROLE 5

3.2.1.2 RESPONSIBILITY 5

3.2.2. Project Business Sponsor 5

3.2.2.1 ROLE 5

3.2.2.2 RESPONSIBILITY 5

3.3. Project Director 5

3.3.1. ROLE 5

3.3.2. RESPONSIBILITY 6

3.4. Project Manager 6

3.4.1. State Project Manager 6

3.4.1.1 ROLE 6

3.4.1.2 RESPONSIBILITY 6

3.4.2. Vendor Project Manager 6

3.4.2.1 ROLE 6

3.4.2.2 RESPONSIBILITY 6

3.5. Requirements Managers 6

3.5.1. State Requirements Manager 6

3.5.1.1 ROLE 6

3.5.1.2 RESPONSIBILITY 7

3.5.2. Vendor Requirements Manager 7

3.5.2.1 ROLE 7

3.5.2.2 RESPONSIBILITY 7

4. REQUIREMENTS MANAGEMENT PROCESS 7

4.1. The Process 8

5. Establish Controlled Requirements Repository 10

5.1. Requirements Repository 11

5.2. Create Requirements Baseline 11

5.2.1. Load Requirements Baseline 12

5.2.2. Identify and Tag Requirement 12

5.2.3. Set Requirement Attributes 12

6. Perform Requirements Traceability 12

7. Managing Requirements Change 13

7.1. Requirements Change Impact Assessment 13

7.2. Updating Baseline Requirements 18

7.2.1. Modification Update 18

7.2.2. Addition Update 19

7.2.3. Deletion Update 19

7.3. Reviewing Traceability 19

8. Requirement Metrics and Status Reporting 20

APPENDIX A. Overview of Requirements Traceability A-1

APPENDIX B. Example Requirements Tagging, Attributes, and Relationships B-1

APPENDIX C. Traceability Metrics and Reporting C-1

APPENDIX D. COTS, MOTS, AND CUSTOM CONSIDERATIONS D-1

# Introduction

## 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 "Bank Management System" software application. This document serves as a foundation for both the development and the validation processes, ensuring that all stakeholders, including developers, testers, and end-users, have a shared understanding of the project's objectives. By detailing the 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.

## Scope of this Document

This document encompasses all the essential details needed to develop the "Bank Management System" software application. It covers the full range of functionalities, including user interface requirements, system architecture, and integration with other platforms or services. The scope also includes system performance and security requirements, data management strategies, 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 the end-users' needs. The value this document provides to the customer includes a clear framework for understanding the project's goals, deliverables, and the expected outcomes.

## Overview

The "Bank Management System" is a secure, user-friendly platform for managing customer accounts and performing essential banking operations. Bank employees can log in to create accounts, check balances, transfer funds, deduct amounts, and close accounts. The intuitive interface ensures ease of use for employees with varying technical expertise. Designed to be secure, responsive, and efficient, it aims to streamline banking tasks. Overall, the system provides a reliable solution for managing banking activities in an organized manner.

# GenERAL Description

The "Bank Management System" is a secure and user-friendly software solution designed to help bank employees efficiently manage customer accounts and perform essential banking operations. Built for both ease of use and reliability, the application provides a central platform for maintaining detailed account information, including customer names, account types, balances, and transaction history.

The primary functions of the application include account creation, balance inquiry, money transfer, funds deduction, and account deletion. With a focus on simplicity and efficiency, the application ensures that employees with varying technical skills can perform banking tasks with ease, supporting smooth and accurate account management for a seamless banking experience.

Key features of the “**Bank Management System**” include:

* **Account Management:** Employees can create new customer accounts, check existing account details, transfer funds, deduct amounts, and close accounts. Each account record includes essential information such as account number, balance, and customer details.
* **Transaction Handling:** The application supports secure money transfers and deductions, with necessary checks to ensure accuracy and prevent unauthorized transactions, providing a reliable and efficient transaction process.
* **User Interface (UI):** Featuring a clean and straightforward interface, the application simplifies navigation and enhances usability for employees with varied technical expertise, allowing them to handle tasks with ease.
* **Secure and Responsive:** Built with security and responsiveness in mind, the application offers a smooth user experience while protecting customer data. Optimized for performance, it ensures fast access to account details and transaction processing.

The overall goal of the **Bank Management System** is to provide a secure, organized, and efficient solution for managing banking operations. With features like quick data entry, secure transactions, and intuitive navigation, it caters to the needs of bank employees managing customer accounts, creating a dependable tool for essential banking tasks.

# FUNCTional Requirements

**3.1. BANK ACCOUNT MANAGEMENT**

**3.1.1. ACCOUNT CREATION**

* The application must allow employees to create new bank accounts by entering customer details such as:

1. Customer Name
2. Account Type (e.g., Savings, Current)
3. Initial Deposit Amount

* The system should validate required fields (e.g., name, deposit amount) and ensure correct data formats (e.g., numerical values for deposits).
* Validated data should be saved to a file in the Windows file system for secure storage.

**3.1.2. BALANCE CHECK**

* The application must allow employees to check the balance of an account by entering the account number or customer name.
* The system should retrieve and display the account balance from the stored data file in real-time.
* This feature should only display the balance without modifying any other account details.

**3.1.3. MONEY TRANSFER**

* Employees should be able to transfer money between accounts by entering:

1. Source Account Number
2. Destination Account Number
3. Transfer Amount

* The system should verify sufficient funds in the source account and validate account details before processing the transfer.
* Upon successful transfer, both accounts' balances should be updated and saved to the file.

**3.1.4. MONEY DEDUCTION**

* The application must allow employees to deduct an amount from a specified account by entering:

1. Account Number
2. Deduction Amount

* The system should verify sufficient funds in the account and update the balance accordingly.
* The updated account balance must be saved to the Windows file system to maintain accurate records.

**3.1.5. ACCOUNT DELETION**

* Employees can delete customer accounts after confirming with the customer.
* The system should prompt for the account number and display the details for final confirmation before deletion.
* Deleting an account should require confirmation to prevent accidental removal and must save this change to the file system.

**3.1.6. DATA IMPORT/EXPORT**

**3.1.6.1. IMPORTING ACCOUNT DATA**

* The application should allow employees to import account data from other file formats (e.g., CSV, txt).
* The system must ensure correct mapping of imported fields (e.g., account number, balance, customer name) to the bank data structure.

**3.1.6.2. EXPORTING ACCOUNT DATA**

* + Employees must be able to export account data to various formats (e.g., CSV, txt) for backup, reporting, or transfer to another system.

# Interface Requirements

**4.1. USER INTERFACE (UI) REQUIREMENTS**

**4.1.1. GENERAL UI LAYOUT**

* + The application should have a clean, intuitive, and user-friendly layout to facilitate smooth navigation.
  + The main screen should display an overview of bank accounts, allowing employees to perform actions such as creating accounts, checking balances, transferring funds, and closing accounts.
  + The interface should minimize the steps required to complete common banking tasks to enhance efficiency and usability.

**4.2. EXTERNAL INTERFACE REQUIREMENTS**

**4.2.1. IMPORT/EXPORT INTERFACES**

* + The application should provide interfaces for importing and exporting account data in common formats (e.g., CSV, txt).
  + For importing, employees should be able to upload account data files to add or update records.
  + For exporting, employees should be able to download the account list or transaction history in CSV or txt format for reporting, backup, or transfer to other systems.

**4.3. SYSTEM INTERFACES**

**4.3.1. DATABASE/STORING INTERFACE**

The application should use a backend file (e.g., txt or similar) to securely store account information. The file interface should:

* + Support CRUD (Create, Read, Update, Delete) operations for account and transaction data.
  + Ensure data security and encryption to protect customer information and transaction history.
  + Maintain file integrity to prevent data loss or corruption during use, safeguarding the consistency and accuracy of records.

# Performance Requirements

**5.1. RESPONSE TIME**

**5.1.1. ACCOUNT BALANCE INQUIRY**

* The application must return balance inquiry results within 1 second under normal conditions.
* For databases with more than 1,000 accounts, the response time should remain under 5 seconds to display balance information.

**5.1.2. ACCOUNT CREATION AND DELETION**

* When creating or deleting an account, the system must reflect the change and display the updated account list within 1 second after the action is submitted.
* The interface should instantly update to show the new or modified list of accounts without delay.

**5.1.3. TRANSACTION PROCESSING**

* Processing transactions such as money transfers or deductions should not take longer than 2 seconds under normal load conditions.
* The system must update account balances immediately after a transaction to ensure real-time accuracy.

**5.2. THROUGHPUT**

**5.2.1. DATA IMPORT/EXPORT**

* The application should import account data from external files (CSV, txt) at a rate of 100 records per second.
* For export, the application must allow users to download an account list (up to 10,000 accounts) within 30 seconds, ensuring efficient data handling.

**5.3. SYSTEM RESOURCE UTILIZATION**

**5.3.1. CPU AND MEMORY USAGE**

* The application should operate efficiently, with CPU usage not exceeding 30% during typical operations such as account inquiries, transfers, and balance checks.
* Memory usage should be limited to 150 MB for managing up to 10,000 accounts, ensuring stability and performance.

**5.4. AVAILABILITY AND RELIABILITY**

**5.4.1. SYSTEM UPTIME**

* The system should achieve a 99.9% uptime over a rolling 30-day period, minimizing downtime and ensuring constant availability to employees.
* Scheduled maintenance or updates should occur during off-peak hours and should not exceed 1 hour per month.

**5.4.2. ERROR HANDLING AND RECOVERY**

* The system must be resilient, with mechanisms to handle and recover from common errors (e.g., data file access issues).
* In the event of an error, the application should display a clear message to employees and ensure no data loss occurs during transactions or updates.

**5.5. LOAD TESTING**

**5.5.1. STRESS TESTING**

* The application should undergo stress testing to verify its ability to handle unexpected spikes in activity, such as a high number of concurrent transactions or account queries.

**5.5.2. SYSTEM FAILOVER AND REDUNDANCY**

* The application must implement failover mechanisms, including data backup, load balancing, and clustering, to maintain performance and prevent disruption during system failures or high-load scenarios.

# Design Constraints

**6.1. PLATFORM AND DEVICE COMPATIBILITY**

**6.1.1. CROSS-PLATFORM SUPPORT**

* The application is designed exclusively for Windows PCs, compatible with Windows 10 and higher, and does not offer cross-platform support.

**6.1.2. DEVICE SPECIFICATIONS**

* The application should operate on any PC with Intel or AMD processors to ensure compatibility across common desktop hardware.

**6.2. VISUAL DESIGN CONSTRAINTS**

* The application's design should limit the use of high-resolution images or complex graphics to optimize load times and reduce CPU usage.
* A simple and neutral color scheme should be used to maintain a professional, user-friendly appearance.

**6.3. DATA STORAGE AND MANAGEMENT**

* All account data is to be stored locally using the Windows file system, with files saved in a standard text (txt) format for easy access and management.

**6.4. PERFORMANCE CONSTRAINTS**

**6.4.1. LATENCY AND LOAD TIME**

* The application must display account balance information or query results within 1 second under normal operating conditions.
* For large databases exceeding 1,000 accounts, query results should load within 2 seconds.

**6.4.2. ACCOUNT CREATION AND DELETION**

* When an account is created or deleted, the system should reflect the change and display the updated account list within 1 second.
* Updates to account lists should be immediate to ensure a responsive user experience.

**6.4.3. TRANSACTION PROCESSING**

* Transactions, such as money transfers or deductions, should process within 2 seconds, ensuring immediate updates to account balances and transaction history.

**6.4.4. DATA IMPORT/EXPORT**

* The application should be capable of importing account data from external files (CSV, txt) at a rate of 100 accounts per second.
* The export process should allow employees to export up to 10,000 accounts within 30 seconds, optimizing efficiency for large datasets.

**6.4.5. CPU AND MEMORY USAGE**

* The application should operate efficiently, with CPU usage not exceeding 30% during standard operations, including account queries and transactions.
* Memory usage should be limited to 150 MB for databases up to 10,000 accounts to ensure smooth performance.

**6.4.6. SYSTEM UPTIME**

* The system should maintain 99.9% uptime over a 30-day rolling period, ensuring continuous availability with minimal downtime.
* Scheduled maintenance or updates should occur during off-peak hours, with a maximum downtime of 1 hour per month.

**6.4.7. ERROR HANDLING AND RECOVERY**

* The system must be robust, with error-handling capabilities to recover gracefully from typical issues (e.g., data access errors).
* If an error occurs, the application should display a clear, informative error message to the user.
* The system should ensure no data is lost during errors or interruptions, especially during critical transactions or data updates.

# Non-functional attributes

**7.1. SECURITY**

**7.1.1. DATA PROTECTION**

* All account and transaction data must be encrypted, ensuring the security of sensitive information such as account numbers, balances, and personal user details.

**7.1.2. ACCESS CONTROL**

* The application should restrict unauthorized access to sensitive banking data through authentication measures, such as secure login credentials for employees.

**7.1.3. COMPLIANCE**

* The system must comply with relevant financial regulations and privacy laws, including GDPR (General Data Protection Regulation) and PCI-DSS (Payment Card Industry Data Security Standard).
* Users must have control over their data, including the ability to delete it upon request, and should be informed of data handling practices upon initial use.

**7.1.4. INCIDENT RESPONSE AND RECOVERY**

* The system should include monitoring and logging capabilities to detect potential security incidents (e.g., unauthorized access attempts or breaches).
* In the event of a security breach, affected users must be notified within 72 hours, and recovery protocols should be followed to secure data and restore functionality.

**7.2. RELIABILITY**

**7.2.1. AVAILABILITY**

* The system must achieve 99.9% uptime to ensure availability, with scheduled maintenance limited to 1 hour per month.
* Automatic failover mechanisms should be in place to maintain service continuity in case of hardware or software failures.

**7.2.2. FAULT TOLERANCE**

* The system should be resilient and recover gracefully from errors, with backup systems to protect data.
* If a transaction fails, the system should retry or roll back the operation to maintain data integrity.

**7.2.3. DATA INTEGRITY**

* Account balances, transaction records, and other user data should be kept accurate and consistent, especially during database updates or synchronization.
* **7.3. USABILITY**

**7.3.1. USER EXPERIENCE (UX)**

* The application must provide an intuitive, user-friendly interface for banking tasks, making it accessible to employees with varying technical backgrounds.
* Essential operations, such as creating an account, processing a transaction, and checking balances, should require no more than 3 steps.

**7.3.2. ACCESSIBILITY**

* The application should be accessible via keyboard navigation and should accommodate users with disabilities.

**7.3.3. LOCALIZATION**

* The application should support multiple languages (e.g., English, Spanish, French) and adjust date, currency symbols, and number formats based on the user's regional settings.

**7.4. MAINTAINABILITY**

**7.4.1. MODULARITY AND EXTENSIBILITY**

* The application should be modular, allowing for isolated updates, bug fixes, and future scalability.
* New features, such as additional transaction types, should be easily integrated into the system without major changes to the core infrastructure.

**7.4.2. CODE QUALITY**

* The codebase must follow clean coding practices, ensuring readability and adherence to industry standards.
* Version control (e.g., Git) and regular code reviews should be implemented to maintain a high-quality codebase.

**7.4.3. AUTOMATED TESTING AND CONTINUOUS INTEGRATION**

* Automated testing (unit, integration, and functional tests) should be used to verify the correct functioning of components and prevent regressions.
* A CI/CD pipeline should automate building, testing, and deployment, facilitating rapid and safe updates.

**7.5. SCALABILITY**

**7.5.1. HORIZONTAL SCALABILITY**

* The system should be designed for horizontal scalability to accommodate an increasing number of accounts and transactions.
* The database should support scalability to handle millions of accounts and transactions, ensuring stable performance as the user base grows.