Table of Contents

Banking Transaction Management System

Chapter 1 – Introduction

1.1 About the System

The **Banking Transaction Management System** is a C-programming-based application designed to streamline the operations of financial institutions by managing customer accounts and transactions efficiently. The system acts as a centralized platform for handling a wide range of banking operations including account creation, customer data management, transaction recording, and balance inquiry.

Each customer is provided with a unique account number, and all financial activities—such as deposits, withdrawals, and fund transfers—are securely recorded using file handling mechanisms. Banking staff and administrators can insert, update, delete, or search records with ease through a simple text-based interface. The system ensures data accuracy, secure storage, and ease of access, making it a practical solution for small to medium-scale banking operations.

In addition to managing individual transactions, the system keeps track of transaction histories, allowing for better financial auditing and customer service. It reduces manual workload and minimizes errors in customer account management, contributing to more reliable and secure banking operations.

1.2 Purpose

The primary goal of this system is to improve the efficiency and accuracy of banking operations by automating account and transaction management. Traditionally, banks relied heavily on manual processes and paper records, which were error-prone and time-consuming.

This system is built to:

- Reduce the risk of human error in transaction records.
- Provide secure, structured storage of customer and account data.
- Allow quick access to transaction history and account status.

- Improve customer service by offering faster processing of deposits, withdrawals, and fund transfers.
- Assist bank staff with a user-friendly system to manage all account-related operations.

By integrating structured C programming concepts like file handling, structures, and functions, the system offers a practical, cost-effective solution to everyday banking challenges.

1.3 Why This System Is Necessary?

In an era where financial security and efficient data management are critical, traditional banking systems face major drawbacks:

- Manual records are prone to loss, damage, or duplication.
- Transactions may be delayed or inaccurately recorded.
- Difficulties arise when trying to track transaction history or balance updates.
- Paper-based systems fail to comply with modern digital standards and security protocols.

This **Banking Transaction Management System** addresses those concerns by:

- Providing a reliable and secure way to store and retrieve customer and transaction data.
- Ensuring fast, accurate, and transparent financial operations.
- Allowing administrators to generate reports, verify balances, and oversee transaction activities efficiently.
- Supporting compliance with banking regulations by securely logging financial activity.

Functional Requirements :

ID	Requirement	Description
FR1	Account Management	The system shall allow staff to register, update, and delete customer accounts.
FR2	Transaction Processing	The system shall process deposits, withdrawals, and fund transfers.
FR3	Record Tracking	The system shall process deposits, withdrawals, and fund transfers.
FR4	Search Functionality	The system shall enable searching accounts by number or name.
FR5	Report Generation	The system shall generate reports of account balances and transaction histories.
FR6	Balance Inquiry	The system shall display the current balance for any account on request.
FR7	Transaction Validation	The system shall validate transactions to ensure sufficient funds.
FR8	Account Lock/Unlock	The system shall allow administrators to lock/unlock accounts for security.
FR9	Interest Calculation	The system shall compute and apply interest to savings accounts monthly.
FR10	Backup Creation	The system shall create daily backups of all account and transaction data.

Non-Functional Requirements:

ID	Requirement	Description
NFR-1	Usability	The system shall be menu-driven and easy to navigate
NFR-2	Reliability	The system shall prevent data loss via structured file handling
NFR-3	Portability	The system shall work across Windows, Linux, and Mac
NFR-4	Performance	The system shall respond within 2 seconds for any query.
NFR-5	Security	User data shall be securely stored in structured files and not shared.
NFR-6	Scalability	The system shall handle up to 50,000 accounts without performance issues.
NFR-7	Availability	The system shall ensure 99.9% uptime for banking operations.
NFR-8	Maintainability	The system shall allow updates to code with minimal downtime.
NFR-9	Compliance	The system shall adhere to banking regulations (e.g., data privacy laws).
NFR-10	Recovery	The system shall restore data from backups within 1 hour after a failure.

SDLC model used for this project :

Choosing the appropriate Software Development Life Cycle (SDLC) model for the Banking Transaction Management System is critical to ensure the project meets its requirements for reliability, security, scalability, and regulatory compliance. Based on the project's characteristics, such as the need for robust stakeholder involvement, complex financial processes, regulatory compliance, and secure data handling, the Spiral Model is the most suitable SDLC model for this project. Below is an analysis of why this model is ideal and a comparison with other models to justify the choice.

3. Why the Spiral Model?

The Spiral Model, a risk-driven and iterative SDLC approach, is chosen for its ability to handle the project's complexity, security demands, and stakeholder involvement. Below is the rationale for its selection:

3.1 Key Reasons for Selection

- **Risk Management**: Each iteration includes risk analysis, critical for addressing security vulnerabilities (e.g., data breaches) and ensuring regulatory compliance, vital for a banking system.
- **Stakeholder Collaboration**: Iterative prototyping allows continuous feedback from bank staff, management, and regulators, resolving conflicts like usability versus security early.
- **Flexibility**: The model supports evolving requirements, such as new regulations or features, through incremental development.
- **Resource Efficiency**: Early spirals prioritize high-priority features (e.g., account management, transaction logging), optimizing resources like developers, budget, and equipment.
- **Iterative Refinement**: Each cycle includes planning, risk analysis, development, and testing, ensuring a secure, reliable, and user-friendly system.

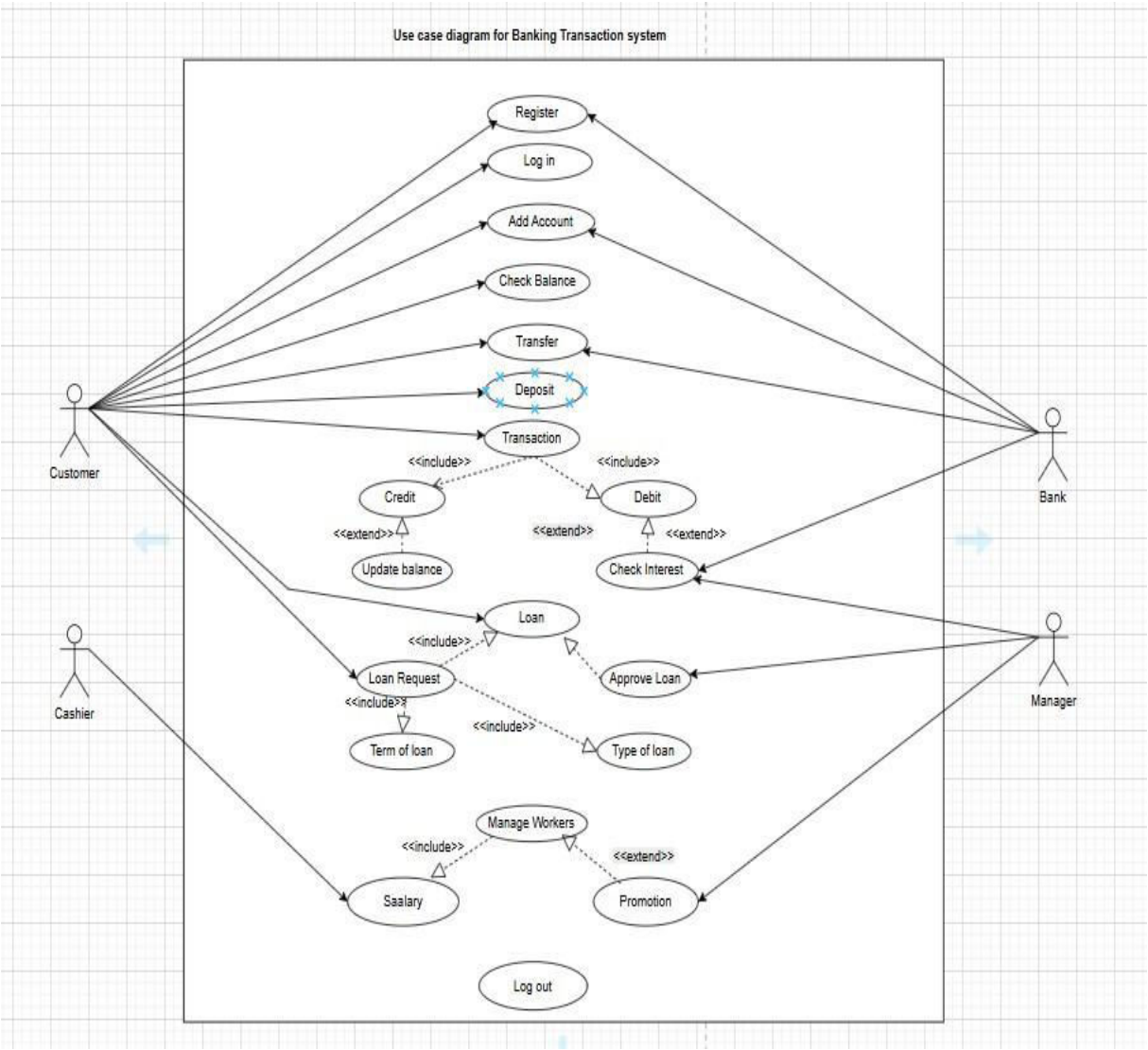
3.2 How the Spiral Model Works

The Spiral Model operates in cycles, each with four phases:

1. **Planning**: Define objectives (e.g., secure transaction processing) and constraints (e.g., regulations).
2. **Risk Analysis**: Identify and mitigate risks like security issues or compliance gaps.
3. **Development**: Implement features using C programming, such as file handling for data storage.
4. **Evaluation**: Test prototypes and gather stakeholder feedback for the next cycle.

Early spirals focus on core functionalities (e.g., account creation), while later ones enhance the user interface and reporting features.

Use case Diagram:



A user must provide appropriate details to securely Login. Software must check and verify the details at every attempt to Login. Here LOGIN is the Base Use Case and AUTHENTICATE is the Included Use Case.

If a user enters appropriate details , user is allowed to Login. However if the details entered by the user are incorrect, software must be able to catch and display problem to the user and allow the user to re-enter details. Login is hence a complete use case. However under certain situations it might use action corresponding to INVALID PASSWORD. Here LOGIN is the Base Use Case and INVALID PASSWORD is Extended Use Case.

Here, we will try to understand the design of a use case diagram for the Online Banking System. Some possible scenarios of the system are explained as follows :

A Customer is required to create an account to avail services offered by Bank. Bank verifies detail and creates new account for each new customer. Each customer is an actor for the Use-Case Diagram and the functionality offered by Online Banking System to Add Account is Use-Case.

Each customer can check the balance in bank account and initiate request to transfer an account across distinct branches of Bank. Cashier is an employee at bank who supports service to the customer.

A customer can execute cash transactions where the customer must either add cash value to bank account or withdraw cash from account. Either of two or both that is credit as well as debit cash, might be executed to successfully execute one or multiple transactions.

After each successful transaction customer might or might not want to get details for action. Manager can check interest value for each account corresponding to transaction to ensure and authenticate details.

A customer can also request loan from bank where customer must add request for loan with the appropriate details.

The type of loan in accordance with purpose or the need for loan and term or duration to pay back the loan must be provided by customer.

The manager of each branch of bank has choice to either accept or approve loan to initiate process further or just reject request for loan based on terms and conditions.

The record for each employee of bank is maintained by bank and bank manages all employees of each branch of bank. The manager of each branch has choice to offer bonus to employees. Note here that each employee is paid as part of management of staff but promotion or bonus might or might not be offered certainly to each employee.

This is the complete design and description for Use-Case of an Online Banking System specifying the use of <<include>> and <<extend>> for certain specific Use-Cases.

USE-CASE-DESCRIPTION:

Use-Case-Registration

Use Case	Registration
Goal	Customer wants to create a new account to access booking services.
Precondition	Customer is not yet registered in the system.
Success End Condition	Customer account is successfully created and stored.
Failed End Condition	Registration is canceled or data is invalid; no account is created.
Primary Actors Secondary Actors	Customer System Database
Trigger	Customer selects "Register" option.
Main Success Scenario	1. Customer selects registration option. 2. System displays form. 3. Customer fills out required fields. 4. System validates data. 5. System creates new account. 6. Confirmation message is shown.
Alternative Flows	3a. Some fields are left empty → Show error message. 4a. Data invalid → Prompt correction.
Quality Requirements	Form should validate inputs in real-time and show feedback within 2 seconds.

Use-Case: Login

Use Case	Log in
Goal	User wants to access their account to perform actions.
Preconditions	User is already registered.
Success End Condition	User successfully logged into their account.
Failed End Condition	Incorrect login credentials; access denied.
Primary Actors Secondary Actors	Customer, Agent Authentication service
Trigger	User clicks on “Log in” and enters credentials.
Main Success Scenario	1. User selects “Log in”. 2. Enters username and password. 3. System verifies credentials. 4. User is granted access.
Alternative Flows	2a. Incorrect input → Show error. 2a. Incorrect input → Show error.
Quality Requirements	Authentication must complete in under 3 seconds. Account lock after 3 failed attempts.

Use-Case: Add Account

Use Case	Add Account
Goal	Admin wants to add a new user account to the system.
Precondition	Admin is logged into the system with necessary privileges.
Success End Condition	New user account is successfully created and stored in the system.
Failed End Condition	Account creation fails due to invalid data or system error.
Primary Actors	Admin
Secondary Actors	System Database
Trigger	Admin selects "Add Account" option from the dashboard.
Main Success Scenario	1. Admin selects 'Add Account' option. 2. System displays account creation form. 3. Admin fills in user details (username, email, role, etc.). 4. System validates the input data. 5. System creates the new account. 6. Confirmation message is shown.
Alternative Flows	3a. Required fields left empty → Show error message. 4a. Data validation fails (e.g., invalid email) → Prompt correction.
Quality Requirements	System should validate inputs in real-time and respond within 2 seconds with feedback.

Use-Case: Check Balance

Use Case	Check Balance
Goal	Customer wants to view their current account balance.
Precondition	Customer is logged into the system.
Success End Condition	System displays the customer's current account balance.
Failed End Condition	Balance inquiry fails due to system error or invalid session.
Primary Actors	Customer
Secondary Actors	System Database
Trigger	Customer selects "Check Balance" option from the menu.
Main Success Scenario	1. Customer selects 'Check Balance' option. 2. System sends request to database. 3. Database retrieves current balance. 4. System displays the balance to the customer.
Alternative Flows	2a. Database connection fails → Show error message.3a. Session expired → Prompt re-login.
Quality Requirements	System should display balance within 2 seconds after the request.

Use-Case: Balance Transfer

Use Case	Balance Transfer
Goal	Customer wants to transfer balance from their account to another account.
Precondition	Customer is logged into the system and has sufficient balance.
Success End Condition	Balance is successfully transferred to the recipient’s account.
Failed End Condition	Transfer fails due to insufficient balance, invalid account details, or system failure.
Primary Actors	Customer
Secondary Actors	System Database, Payment Processor
Trigger	Customer selects "Balance Transfer" option from the transaction menu.
Main Success Scenario	1. Customer selects 'Balance Transfer'. 2. System displays transfer form. 3. Customer inputs recipient account number and amount. 4. System validates balance and recipient info. 5. System processes transfer. 6. Confirmation message and transaction ID are shown.
Alternative Flows	3a. Invalid account number → Show error message. 4a. Insufficient balance → Display warning. 5a. System error during processing → Retry or notify failure.
Quality Requirements	Transfer should complete within 5 seconds and system must provide immediate success/failure feedback.

Use-Case: Withdraw Balance

Use Case	Withdraw Balance
Goal	Customer wants to withdraw balance from their account.
Precondition	Customer is logged into the system and has sufficient available balance.
Success End Condition	Withdrawal is successfully completed and balance is deducted.
Failed End Condition	Withdrawal fails due to insufficient balance, invalid details, or system error.
Primary Actors	Customer
Secondary Actors	System Database, Payment Processor
Trigger	Customer selects "Withdraw" option from the dashboard or menu.
Main Success Scenario	1. Customer selects 'Withdraw Balance'. 2. System displays withdrawal form. 3. Customer enters amount and selects withdrawal method. 4. System validates balance. 5. System processes withdrawal request. 6. System confirms withdrawal and updates balance.
Alternative Flows	3a. Amount exceeds available balance → Show insufficient balance message. 4a. Invalid withdrawal method → Prompt correction. 5a. System error → Retry or notify failure.
Quality Requirements	Withdrawal process should be completed within 5 seconds with real-time balance update and confirmation feedback.

Use-Case: Deposit Balance

Use Case	Deposit Balance
Goal	Customer wants to deposit money into their account.
Precondition	Customer is logged into the system.
Success End Condition	Deposit is successfully completed and balance is updated.
Failed End Condition	Deposit fails due to invalid details, payment gateway failure, or system error.
Primary Actors	Customer
Secondary Actors	System Database, Payment Gateway
Trigger	Customer selects "Deposit Balance" option from the menu.
Main Success Scenario	1. Customer selects 'Deposit Balance'. 2. System displays deposit form. 3. Customer enters deposit amount and selects payment method. 4. System validates the input. 5. System processes payment via payment gateway. 6. Confirmation message is displayed and account balance is updated.
Alternative Flows	3a. Required fields left empty → Show error message. 4a. Invalid payment details → Prompt correction. 5a. Payment gateway error → Retry or notify failure.
Quality Requirements	Deposit transaction should complete within 5 seconds with real-time balance update and feedback.

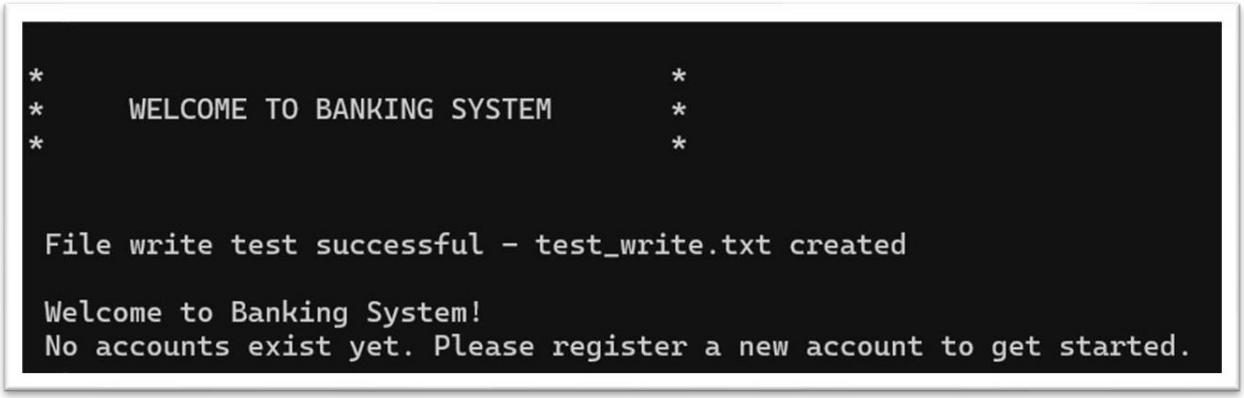
Use-Case: Loan Application

Use Case	Loan Application
Goal	Customer wants to apply for a loan through the system.
Precondition	Customer is logged in and eligible to apply for a loan.
Success End Condition	Loan application is successfully submitted for processing.
Failed End Condition	Loan application fails due to invalid data, ineligibility, or system error.
Primary Actors	Customer
Secondary Actors	System Database, Loan Processing System
Trigger	Customer selects "Apply for Loan" option from the dashboard.
Main Success Scenario	1. Customer selects 'Apply for Loan' option. 2. System displays loan application form. 3. Customer fills in required details (loan amount, term, purpose, etc.). 4. System validates the provided information. 5. System submits the application for review. 6. Confirmation message is shown to the customer.
Alternative Flows	3a. Required fields are left empty → Show error message. 4a. Customer does not meet eligibility criteria → Show rejection message. 5a. System error during submission → Prompt retry or show failure message.
Quality Requirements	Application process should complete within 5 seconds, with proper validation and confirmation feedback.

Prototype Design Based on Functional Requirements

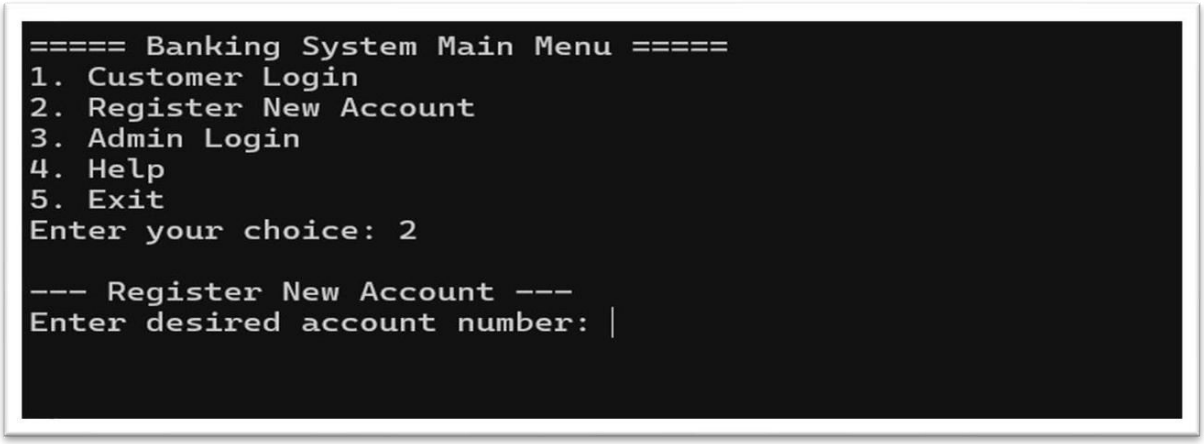
1. Welcome message:

This function displays a welcome message when the program starts.
It serves as the introduction screen for the Banking Transaction Management System.



2. Main menu:

This function displays the main menu of the system. It provides users with different options to perform banking operations such as customer login, registration, admin login, help etc.

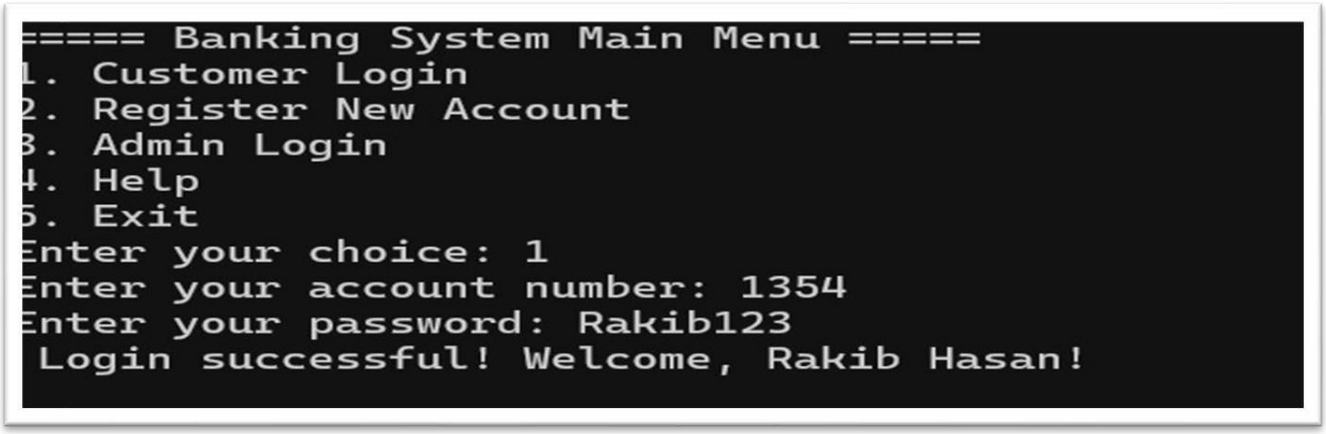


3.Registration Account:

This function allows a new customer to register an account in the system.
The user provides an account number, name, and initial deposit, and the system confirms the successful account creation.

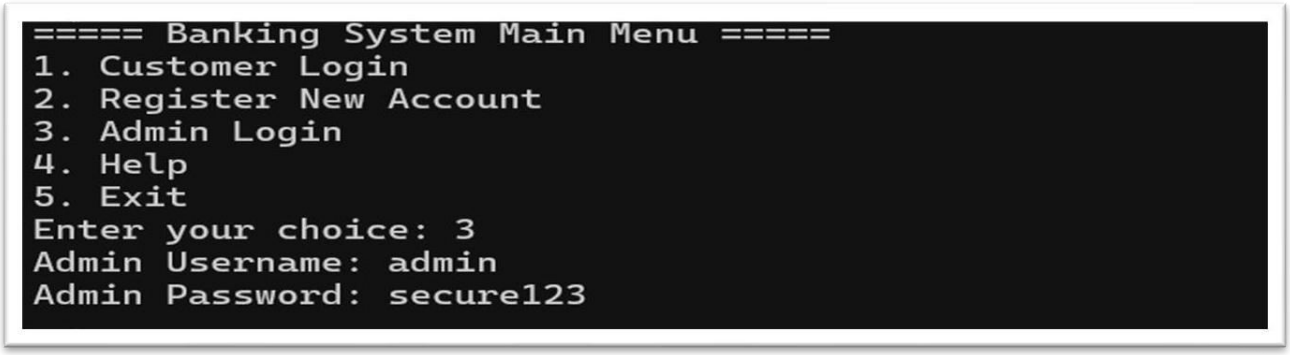
4.Login Account:

It checks the entered account number and PIN against stored values and grants or denies access accordingly**.**



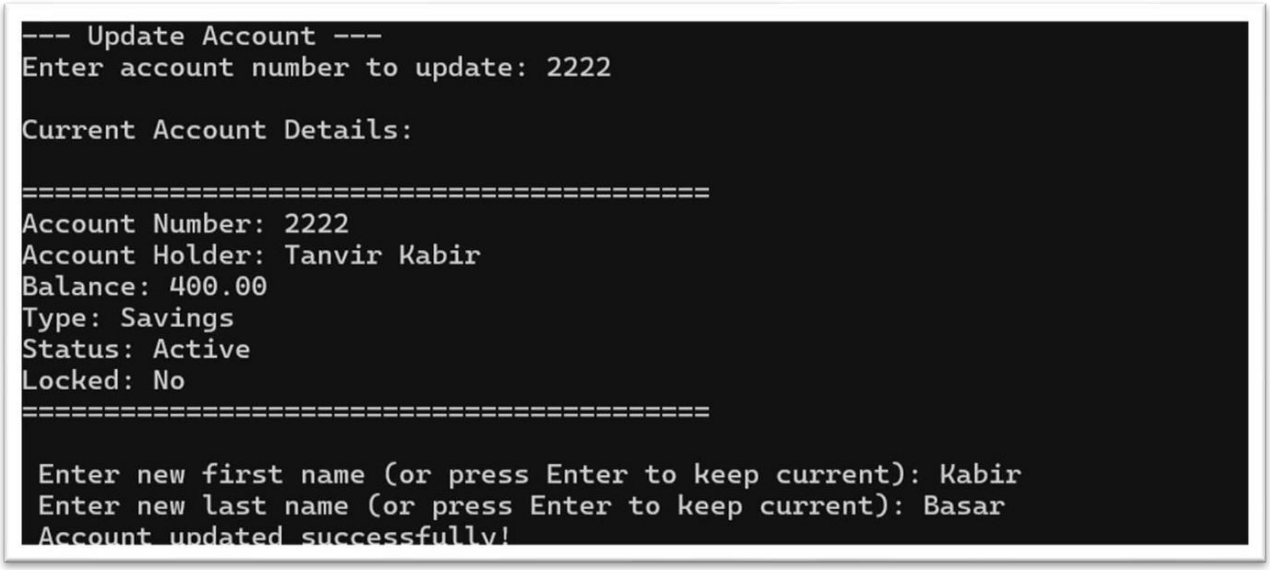
5.Admin Login:

This function allows the administrator to log in to the system.It verifies the entered username and password against stored credentials and grants or denies access accordingly.



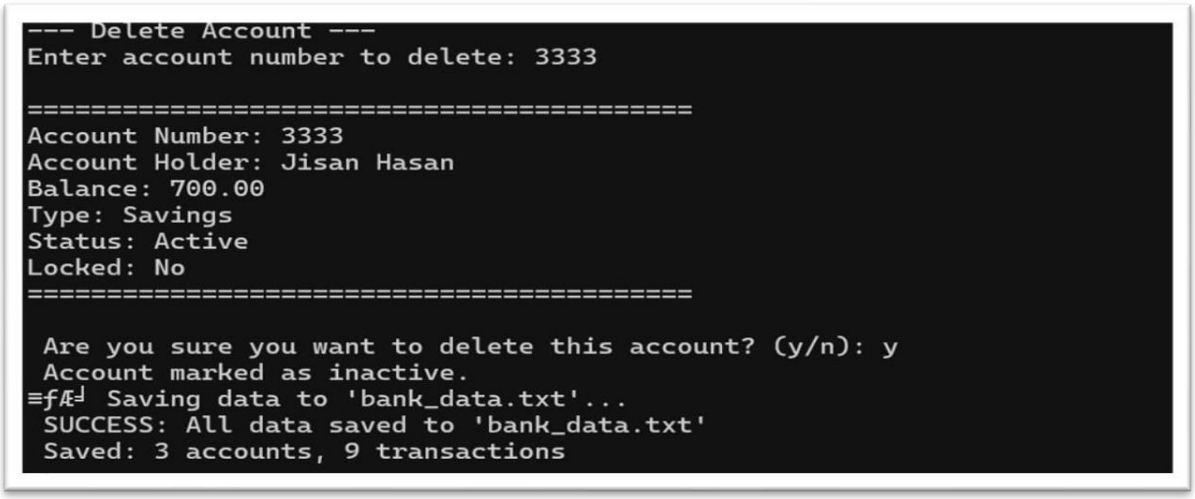
6.Update Account:

This function is used to update customer account details.
It asks for the account number and new information, then confirms the successful update of the record.



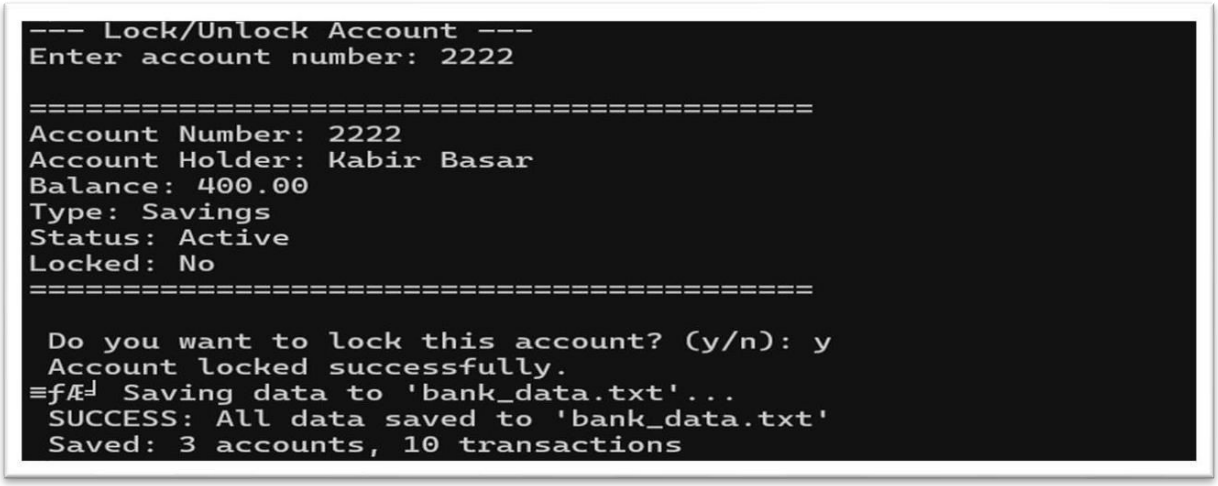
7. Delete Account:

The user enters the account number, and the system confirms the account has been successfully deleted.



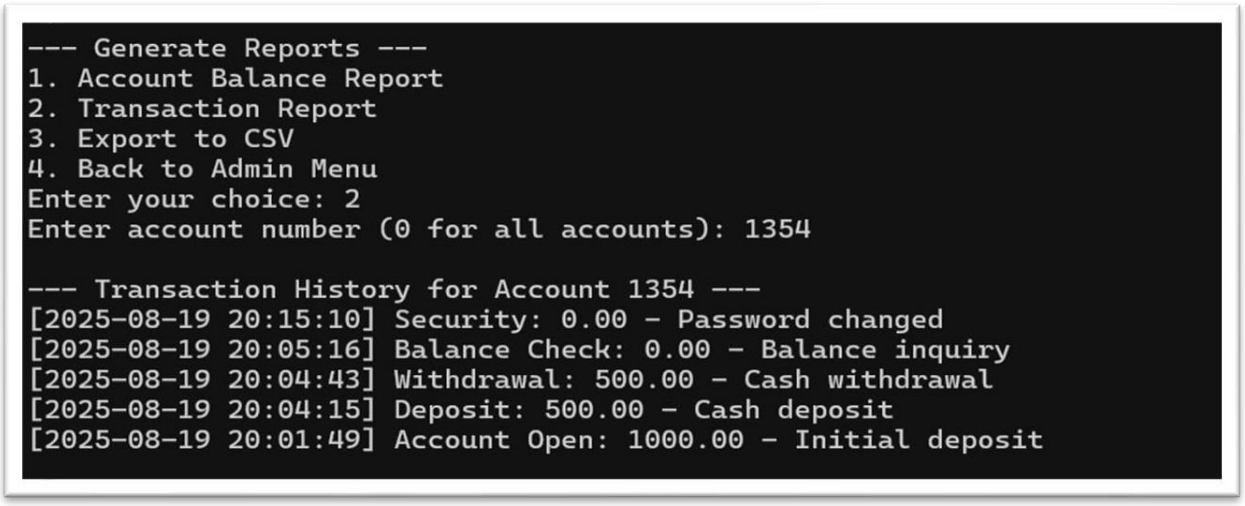
8. Lock Account:

This function allows the administrator to lock a customer's account temporarily. Once locked, the account cannot perform any transactions until it is unlocked by the admin.



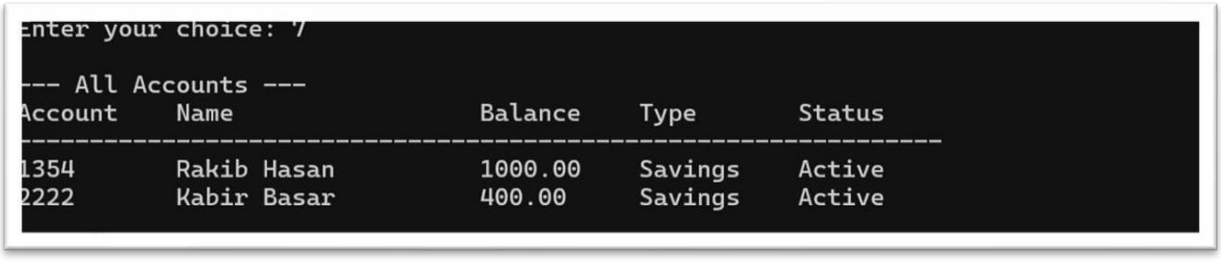
9. Generates Reports:

This function generates a summary report of the banking system.
It shows key information like total accounts, security, Initial deposite, Withdraws, and total deposits for administrative review.



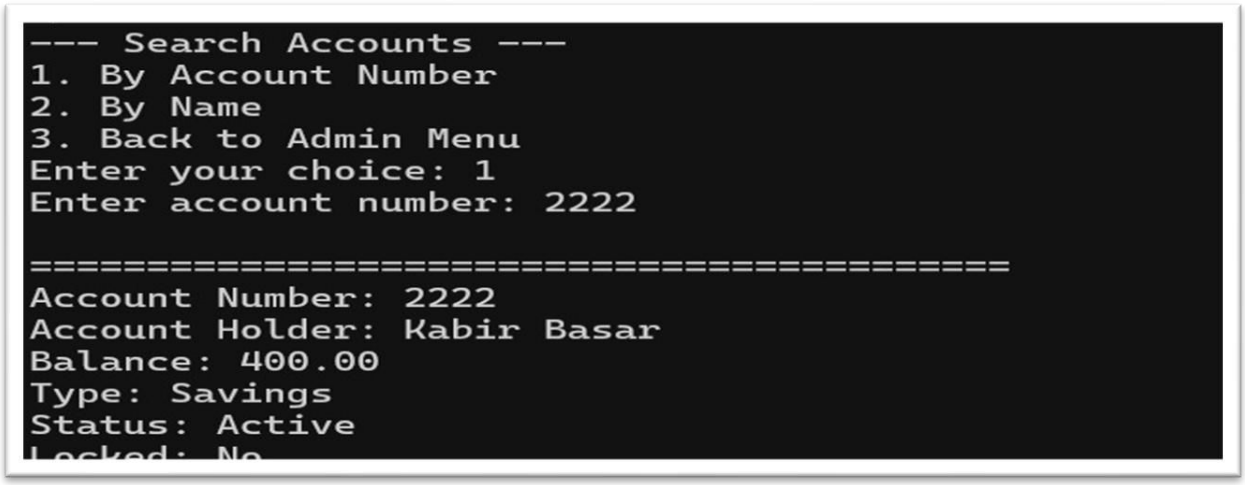
10. View All Accounts:

This function allows the administrator to view all customer accounts in the system. It displays important details like account number, holder name, and account balance for management purposes.



11. Search Account:

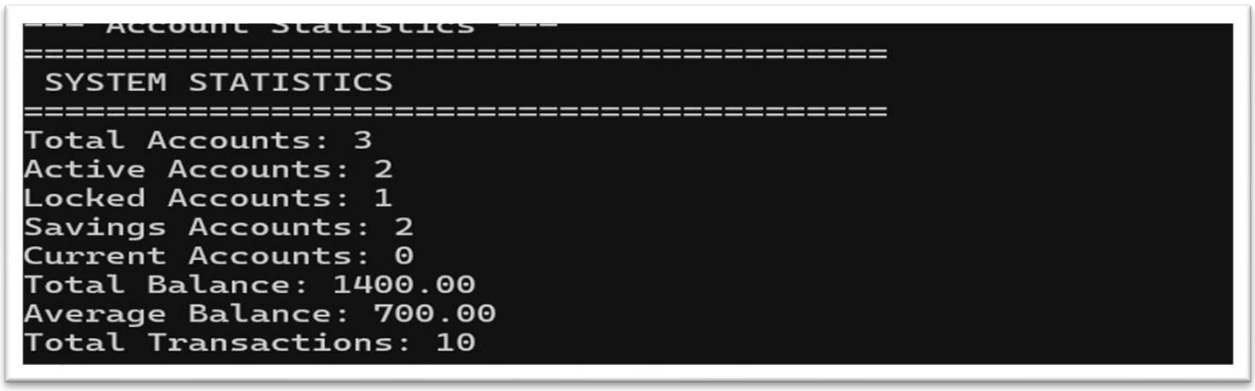
This function allows users or admins to search for a specific account using the account number.It displays the account's details if found, helping to quickly verify customer information.



12. Account Statistics:

This function provides statistical information about all accounts in the system.

It shows total accounts, active and locked accounts, and the total balance, helping administrators monitor the system's status**.**



13. Logout:

This function logs the user out of the banking system.It ensures that the current session ends safely, preventing unauthorized access to the account.

System Testing for Banking Transaction Management System

System testing is a critical phase in the software development lifecycle where a complete, integrated banking transaction management system is tested to verify that it meets the specified requirements. This level of testing is performed after integration testing and before acceptance testing. The main goal of system testing is to evaluate the system's compliance with the specified requirements. It involves testing the system as a whole in a fully integrated environment to ensure that it functions correctly and reliably.

Testing Analysis for Whole System

The Banking Transaction Management System, developed in C, has undergone comprehensive system testing to ensure it meets all functional and non-functional requirements.

Our Objectives Were:

- Functional Validation: Confirm all features and functionalities work as intended.

- Non-Functional Evaluation: Assess performance, usability, and reliability.
- Defect Identification and Resolution: Detect and resolve issues before deployment.
- User Experience Assurance: Ensure the system is intuitive and user-friendly.

Testing Outcomes

1. Functional Testing:

For Customers:

- Successfully created and logged into user accounts.
- Viewed account details, including balance and recent transactions.
- Performed deposits, withdrawals, and fund transfers successfully.
- Verified transaction history and account statements.

For Bank Administrators:

- Successfully added, updated, and deleted customer accounts.
- Monitored customer transactions and account activities.
- Verified and generated account and transaction reports.

2. Non-Functional Testing:

- Performance: The system maintained responsiveness under peak banking load conditions.
- Security: Prevented unauthorized access and safeguarded sensitive financial data.
- Usability: Users found the system easy to navigate, with clear instructions and menus.

3. Regression Testing:

- Verified that recent code updates did not negatively impact existing functionalities.
- Ensured that bug fixes did not introduce new issues.

4. End-to-End Testing:

- Confirmed seamless workflow from user login to performing transactions and receiving confirmations.
- Verified complete administrator processes from logging in to generating customer transaction reports.

Defects Identified and Resolved

- Defect Logging: All identified defects were logged with detailed descriptions and reproduction steps.
- Defect Tracking: Utilized a defect tracking tool to monitor the status and resolution of each defect.
- Defect Resolution: Prioritized and addressed defects based on their severity and impact on the system.

Documentation and Reporting

- Maintained comprehensive documentation, including test plans, test cases, test results, and defect logs.
- Generated detailed test reports summarizing the testing activities, outcomes, and identified issues.

The Banking Transaction Management System has successfully passed through rigorous system testing, demonstrating its readiness for deployment. All functionalities have been validated, non-functional requirements have been met, and identified defects have been resolved. The system is now verified to be reliable, secure, and user-friendly, ensuring a smooth and efficient experience for both customers and bank administrators. The next step is deployment, where continuous monitoring and minor adjustments will ensure optimal performance in the production environment.

Conclusion:

This project creates a Banking Transaction Management System that streamlines banking operations for both customers and administrators. By making the system user-friendly, secure, and efficient, it saves time, reduces errors, and ensures smooth financial transactions.

Future Work:

In future versions, we could add features such as mobile banking integration, advanced financial analytics for administrators, and personalized services for customers. These improvements would enhance user experience and make banking more convenient and accessible for everyone.