**Software Requirements Document**  
**1. Introduction**  
This document outlines the software requirements for the Bank Account Management System.   
The system is intended to manage customer accounts, handle transactions such as deposits and withdrawals, and provide account balance information.  
It also supports the saving of account details to a file and handles exceptions for invalid inputs.  
**2. Purpose**

The purpose of the Bank Account Management System is to:  
Enable the creation and management of bank accounts.

* Allow customers to deposit and withdraw funds from their accounts.
* Track and display account balances.
* Manage various account types, such as Savings Accounts and Current Accounts.
* Handle transactions with error and exception management.
* Save account details to a file for future use.

**3. Scope**

* Account creation.
* Deposits, withdrawals, and balance checking.
* Viewing transaction records.
* Storing account information to a text file for persistence.
* Supporting multiple accounts under each customer.

1. **FUNCTIONAL REQUIREMENTS**

**Account Types**  
 **Savings Account**

* A savings account should allow customers to deposit funds and withdraw from their balance (i.e. non-overdraft)
* The system display the current balance after every transaction and prevent withdrawals that exceed the available balance.
* Features: No overdraft, only balance-based withdrawal.

**Current Account**

* A current Account allows deposits and withdrawals, but with an overdraft facility.
* The Overdraft limit is set by default to 1000 units but it could be configurable per customer.
* Features: Overdraft protection, withdrawal behaviour based on balance and overdraft limit.

**Transaction Handling**  
 **Deposit**  
Upon deposit, the system checks if the entered amount is positive. If invalid (zero or negative), it throws an error.

Updates the balance and displays the new balance to the user after the successful deposit.

Method: deposit (double amount)

**Withdraw**:

Similar to the deposit operation but checks if the withdrawal amount is within the available funds (including overdraft limits for current accounts). Invalid amounts trigger errors, such as insufficient funds or overdraft limit exceeded.

Method: withdraw (double amount)

Account Information Storage

**Saving Account Data**:

* After transactions or during a program exit, the system saves account information to a text file.
* The file should save the customer’s name, identity number, email, phone number, KRA pin, and the balance, which will be available for later retrieval.
* Each field is saved line-by-line as a text record.
* Saving Functionality: saveToFile (ofstream& outFile)

**Data Integrity**:

* At any point, when a change occurs (deposit or withdrawal), the system should update the balance correctly in real-time.
* If the program encounters any issues with saving (e.g., file system errors), it should notify the user.  
  Customer and Account Interaction

**Managing Multiple Accounts:**

* Each customer can hold one or more accounts.
* A customer can perform actions like deposit/withdrawal on each account individually.
* Multiple types of accounts are allowed for a customer.
* Account details for a customer are printed after all the transactions have been made, showcasing each account and its balance.

**Transaction History**:

Each transaction (deposit, withdraw) is treated as an instance, allowing a history of transactions for each account. This functionality could be expanded in the future for detailed transaction logging.

**Error Handling**

**Invalid Input**:

* If a user inputs invalid data (non-numeric input for amounts), the program will continue by showing an appropriate error message, asking the user to re-enter a valid input.
* Invalid menu options (outside 1-4) result in the user being prompted again.
* There is a check to ensure that deposit and withdrawal amounts must be positive.

**Insufficient Funds:**

* In a Savings Account, withdrawal will only succeed if sufficient funds are available.
* In a Current Account, withdrawal will succeed if the available balance (including overdraft limit) is not exceeded.

**Exceeding Overdraft Limit:**

* For Current Accounts, any attempt to withdraw more than the balance and overdraft limit results in an error message (e.g., “Withdrawal exceeds overdraft limit”).
* The program throws exceptions using `throw invalid\_argument` or `throw runtime\_error` to manage error handling.

**Account Display and Management**

* After executing transactions or on request, the system displays current account balances.
* It supports customer management, where all customer accounts can be displayed by checking balances (including customer details like name and ID).
* Each account (whether Savings or Current) will show the available balance to the user during transaction handling.

1. **Non-Functional Requirements**

**Security**

* While no advanced encryption for account information or passwords is implemented in this initial version, sensitive customer data like KRA PIN should be handled with care. In future versions, the file-saving system could incorporate data encryption.
* Only required fields (balance and identification details) should be saved to the file.

**Usability**

* The user interface, although basic and console-based, should:
* Provide clear instructions for user interactions (like the main menu with numbered options).
* Error handling messages are descriptive, guiding the user on how to proceed after an invalid input.
* After each transaction, clear feedback messages will be shown (i.e., “Deposit of [amount] successful, new balance: [new\_balance]”).

**Scalability and Extensibility**

* Multiple Accounts:
* The system is scalable in the sense that multiple accounts (either Savings or Current accounts) can be handled by adding new ones to the customer’s vector (`vector<Account\*>`).
* Overdraft Limits: Future versions of this application could allow custom overdraft limits per customer.
* Additional Features:
* Transaction records could later be logged in a separate file, enabling more advanced features like account history or reports.
* The architecture allows you to add more account types (e.g., business accounts, student accounts, etc.).

**Availability**

* The system is available during runtime, processing transactions or saving data to file when the user decides to exit.

**User Stories (Use Case Scenario)**

**Use Case 1: Creating an Account**

1. A user enters their name, ID, email, phone, and KRA PIN.
2. They select the account type (Savings or Current).
3. The system validates the input and initializes an account object.
4. A new account object is created for the customer (of the chosen type), and the balance is initialized to zero.

**Use Case 2: Making a Deposit**

1. A customer accesses the deposit menu.
2. They enter the amount to deposit.
3. The system checks that the amount is valid and performs the deposit (i.e., increments the balance).
4. The new balance is displayed to the user.

**Use Case 3: Making a Withdrawal**

1. The customer accesses the withdrawal menu.
2. The withdrawal amount is entered by the user.
3. The system checks for available funds (sufficient balance or available overdraft) and deducts the amount from the account balance if valid.
4. If withdrawal conditions are violated (e.g., insufficient funds, overdraft limit exceeded), an error message is shown.

**Use Case 4: Displaying Account Information**

1. After transactions or when desired, the user can display account details and current balance, including all their registered accounts.

**Dependencies**

* The code depends on C++ Standard Library features:
* I/O (iostream): For user input/output.
* String: For handling string data, such as names and identification numbers.
* Vector: Used to store multiple accounts per customer.
* Fstream: Used for saving account details to a file.

**Performance Considerations**

* Given that this is a relatively simple console application, performance requirements are not stringent.
* No significant overhead is expected in terms of computational power for a small number of accounts and transactions.
* Input/output operations (such as writing to and reading from a text file) could slow down if dealing with very large data files.

**Error Messages and Logging**

* Invalid Input (menu choice or amounts):
* Message: “Invalid input, please enter a number between 1 and 4.”
* Action: Requests re-entry of the option.
* Insufficient Funds / Overdraft Exceeded:
* Message: “Error: Insufficient funds or overdraft limit exceeded.”
* Action: Blocks further withdrawals.

**3, Class Diagram**

**Diagram Description:**

**Account (Abstract):**

* Attributes: name, identityNumber, email, phoneNumber, kraPin, balance
* Methods: deposit(), withdraw(), checkBalance(), saveToFile()

**SavingsAccount (Inherits Account):**

* Overrides: deposit(), withdraw()

**CurrentAccount (Inherits Account):**

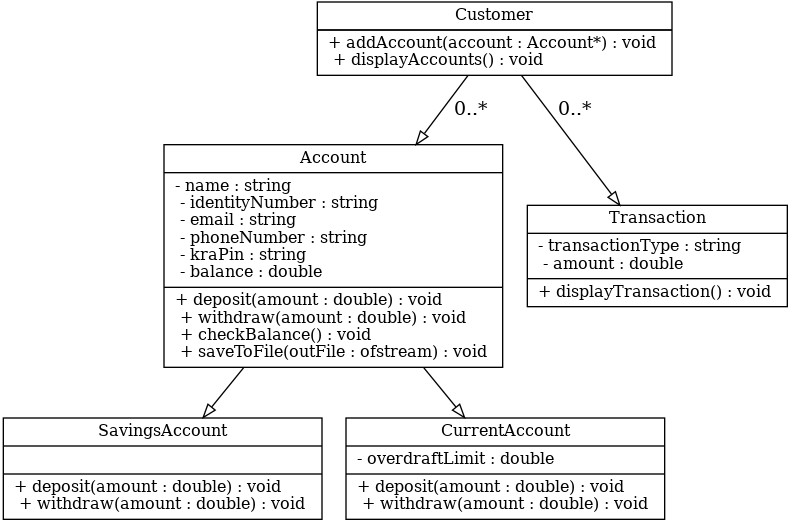
* Attributes: overdraftLimit
* Overrides: deposit(), withdraw()

**Customer:**

* Attributes: customerName, customerId, accounts
* Methods: addAccount(), displayAccounts()

**Transaction:**

* Attributes: transactionType, amount
* Methods: displayTransaction()



**Interaction Diagrams**

**Sequence Diagram: Deposit Transaction**

* User selects Deposit from menu.
* System requests the deposit amount.
* User enters the amount.
* System validates input:
* If valid, update account balance and display confirmation.
* If invalid, prompt error message and request input again.

**Sequence Diagram: Withdraw Transaction**

* User selects Withdraw from menu.
* System requests withdrawal amount.
* User enters the amount.
* System validates input:
* If amount exceeds available balance/overdraft limit, show an error.
* If valid, deduct amount and display confirmation.

**Sequence Diagram: Save Account Data**

* User completes menu operations and exits.
* System saves all account details to account\_data.txt.
* Confirmation message displayed.

**Test Cases and Results**

**Test Case 1: Create SavingsAccount**

* Input: Customer Name: John Doe, ID: 123456, Email: [john@example.com](mailto:john@example.com), Phone: 1234567890, KRA PIN: AB123.
* Action: Create Savings Account.
* Expected Result: Account created with a balance of 0.0.
* Outcome: Pass

**Test Case 2: Deposit into SavingsAccount**

* Input: Deposit Amount: 5000.
* Action: Perform deposit.
* Expected Result: New balance is updated correctly.
* Outcome: Pass

**Test Case 3: Withdraw Exceeding SavingsAccount Balance**

* Input: Withdrawal Amount: 6000 (Balance: 5000).
* Action: Attempt withdrawal.
* Expected Result: Error: Insufficient funds.
* Outcome: Pass

**Test Case 4: Withdraw Within OverdraftLimit of CurrentAccount**

* Input: Withdrawal Amount: 800 (Balance: 500, Overdraft: 1000).
* Action: Perform withdrawal.
* Expected Result: Balance reflects overdraft usage.
* Outcome: Pass

**Test Case 5: Invalid Input Handling**

* Input: Enter characters instead of numbers in the menu.
* Action: Trigger menu option or amount input.
* Expected Result: Error message prompting re-entry.
* Outcome: Pass

**Test Case 6: Save Account Details to File**

* Action: Complete operations and exit the menu.
* Expected Result: Account details are saved to account\_data.txt with accurate data.
* Outcome: Pass