

FUNCTIONAL REQUIREMENTS

Q1: How will a new customer join the bank system?  
 A1: The system must allow a customer to register by providing their details (first name, surname, address, employment information).

Q2: Can a customer have more than one account?  
 A2: Yes, the system must allow a customer to open multiple accounts (Savings, Investment, Cheque).

Q3: What rules apply when opening different accounts?  
 A3:

* Savings Account: customer deposits funds, but no withdrawals are allowed.
* Investment Account: customer must deposit a minimum of BWP500.00, can deposit and withdraw, and earns 5% monthly interest.
* Cheque Account: only employed customers can open; must provide employer name and address; allows deposits and withdrawals.

Q4: How are deposits handled?  
 A4: Customers must be able to deposit money into any of their accounts.

Q5: How are withdrawals handled?  
 A5: Withdrawals are allowed only for Investment and Cheque accounts (not Savings). The system must check available balance before withdrawing.

Q6: How is interest calculated?  
 A6:

* Savings accounts earn 0.05% monthly.
* Investment accounts earn 5% monthly.  
   The system must calculate and apply interest automatically at the end of each month.

Q7: Can customers view their account status?  
 A7: Yes, the system must provide balance inquiry and transaction history per account.

Q8: What records are kept?  
 A8: Every transaction (deposit, withdrawal, interest applied) must be logged with date, type, amount, balance after, and description.

Q9: Can admins interact with the system?  
 A9: Yes, bank admins can generate reports and trigger monthly interest processing.

NON-FUNCTIONAL REQUIREMENTS

Q1: How should the system ensure data protection?  
 A1: The system must use secure authentication, encrypted storage, and role-based access control to protect customer data.

Q2: How reliable should transactions be?  
 A2: All deposits and withdrawals must be atomic (either fully complete or not at all), ensuring no partial transactions.

Q3: How fast should the system respond?  
 A3: The system should process deposits/withdrawals within 2 seconds under normal load.

Q4: How user-friendly should the system be?  
 A4: The system must provide a simple and intuitive interface with clear navigation for both customers and admins.

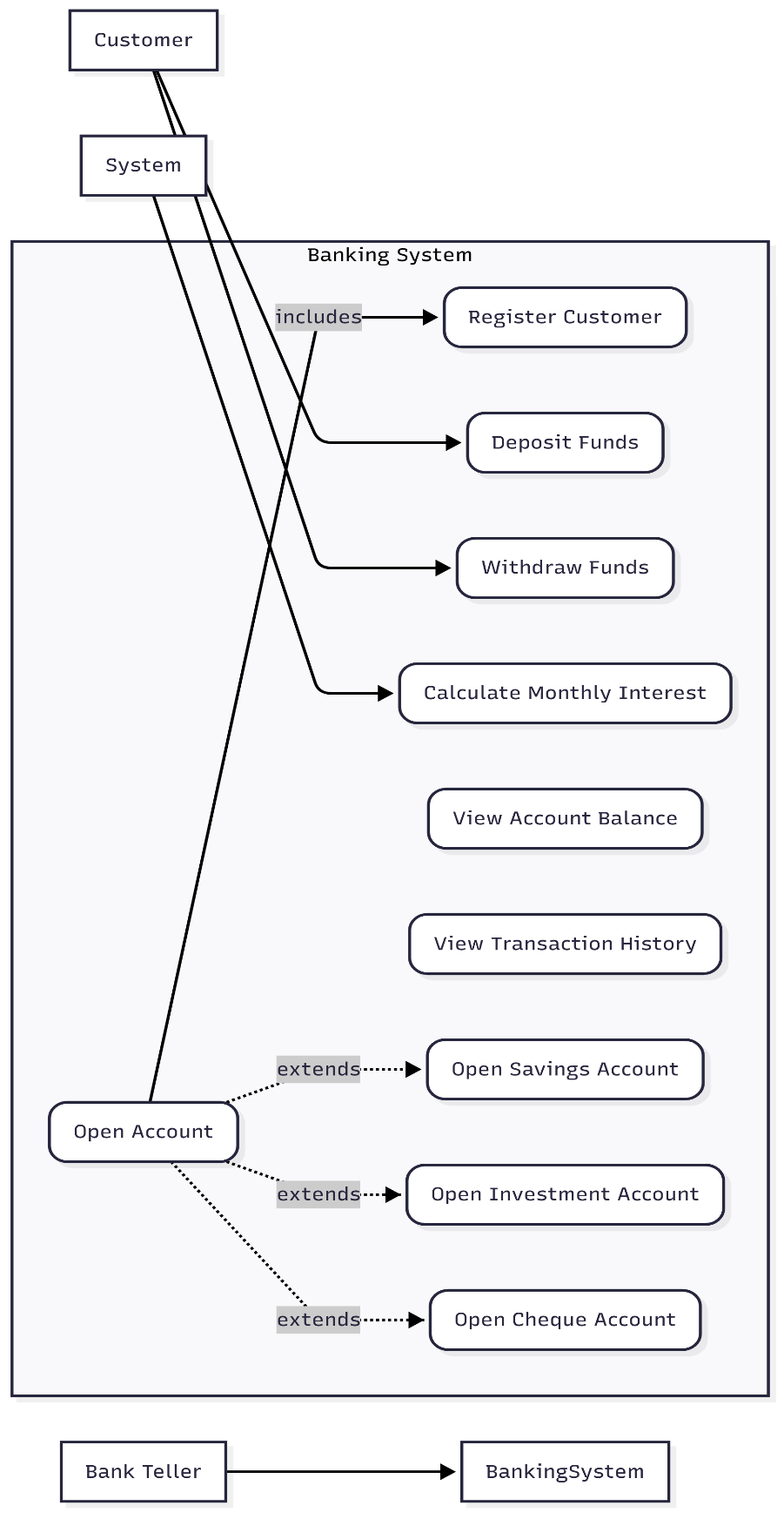
Q5: How should the system scale?  
 A5: It should support hundreds of customers and thousands of transactions without performance degradation.

Q6: How is auditing handled?  
 A6: The system must maintain an immutable log of all transactions for compliance and auditing.

Q7: What about system availability?  
 A7: The system should be available 24/7 with backup and recovery mechanisms to ensure business continuity.

Q8: How easy is it to maintain or extend?  
 A8: The design must follow OOP principles (abstraction, inheritance, polymorphism, interfaces) so new account types or features can be added easily.

**SYSTEM USE CASE DIAGRAM**

****

Use Case Name: Open Investment Account

Actor: Bank Teller (Primary), Customer (Secondary)

Description: This use case allows a Bank Teller to create a new Investment Account for an existing customer or a new customer, ensuring the initial deposit meets the minimum requirement of BWP 500.00.

Preconditions:

1. The Bank Teller is logged into the system.
2. (For existing customers) The customer must already be registered in the system.

Postconditions:

1. A new Investment Account is created and associated with the customer.
2. The initial deposit is recorded in the account's balance.
3. A transaction record for the initial deposit is created.

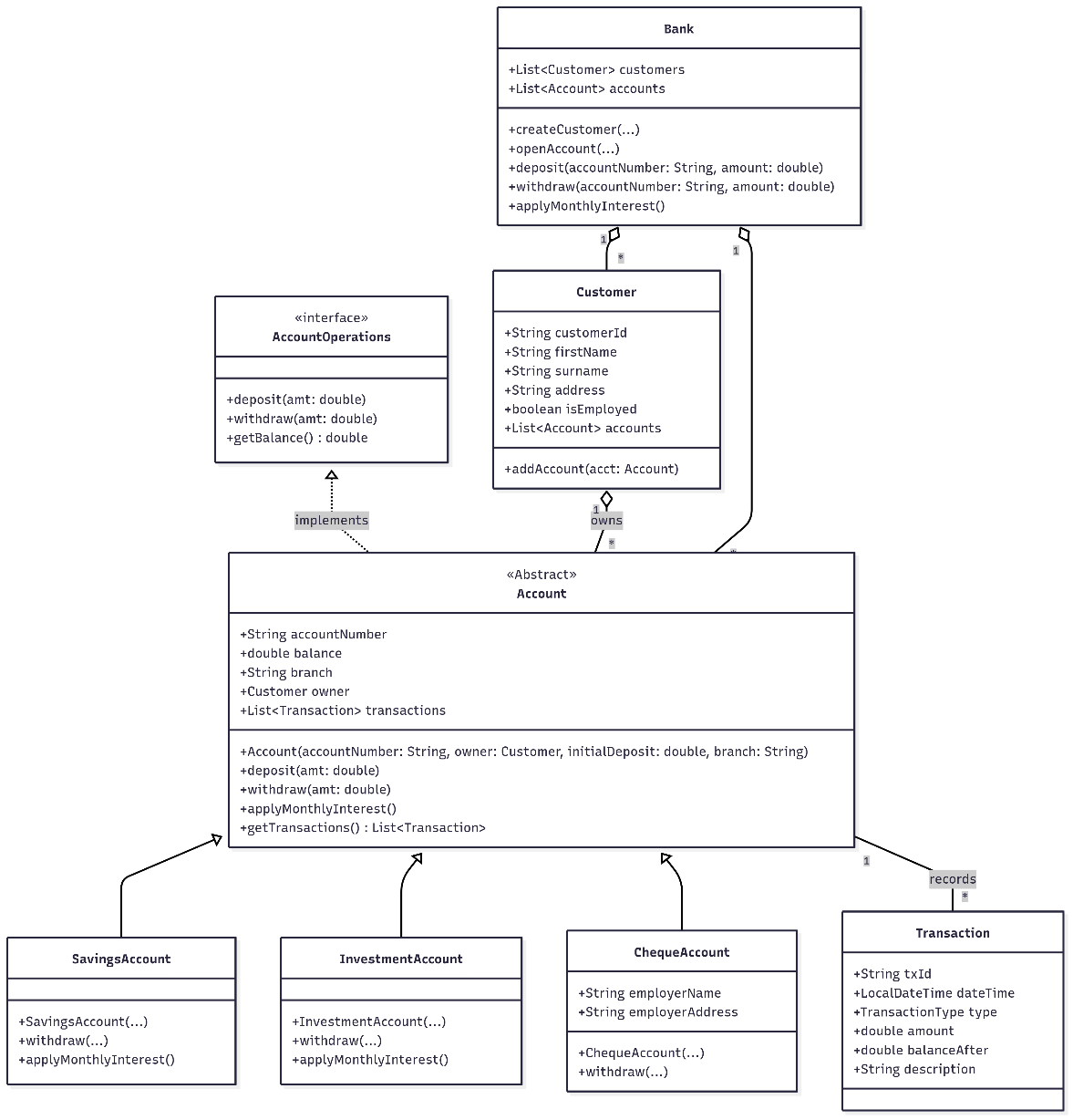
Basic Flow of Events:

1. The use case begins when the Bank Teller selects the "Open New Account" option and chooses "Investment Account".
2. The system prompts the teller to enter the customer's identifier (e.g., name or ID number).
3. The Bank Teller enters the customer identifier.

Extension Point A: Register New Customer (if the customer is not found).

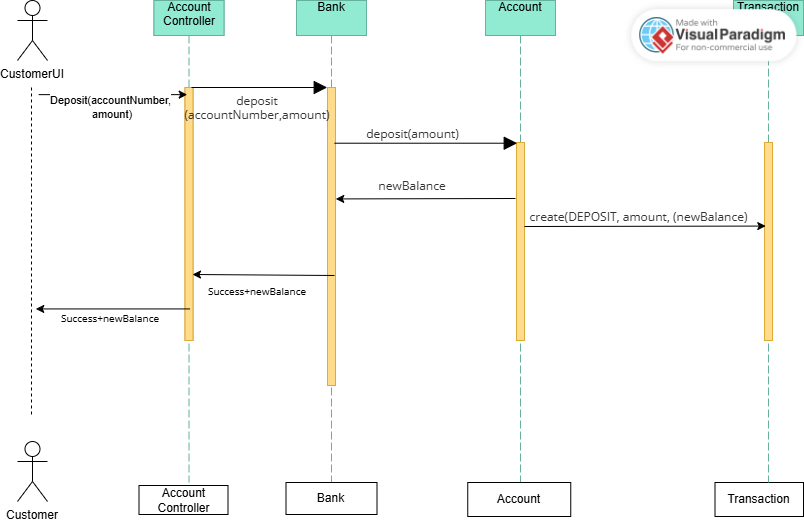
1. The system displays the customer's details for verification.
2. The system prompts for the initial deposit amount and the branch for the account.
3. The Bank Teller enters the initial deposit amount and selects a branch.
4. The Bank Teller submits the information.
5. The system validates that the deposit amount is at least BWP 500.00.
6. The system creates a new Investment Account with a unique account number, sets the balance to the initial deposit amount, and links it to the customer and branch.
7. The system creates a "DEPOSIT" transaction for the initial amount.
8. The system displays a success message with the new account number.
9. The use case ends.

**CLASS DIAGRAM**

****

**BEHAVIORAL UML MODELLING**

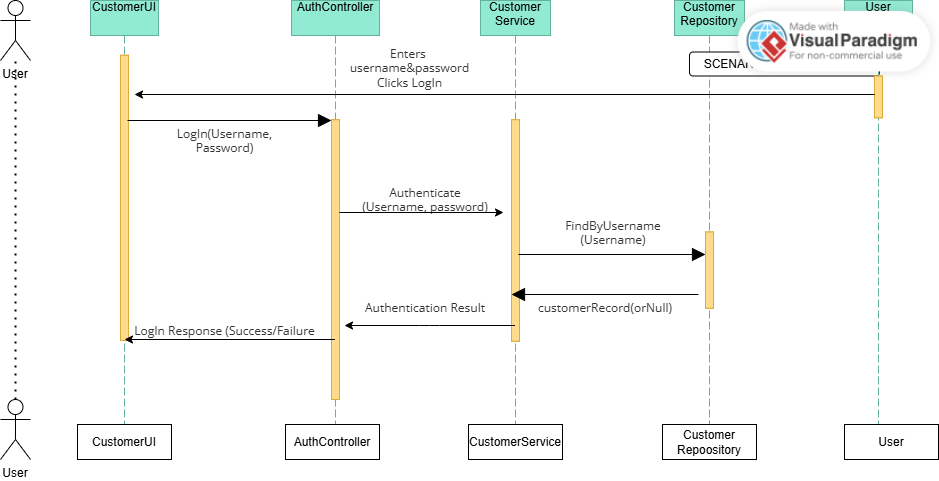
**3.1 SEQUENCE DIAGRAM : DEPOSIT**

****

Message Flow:

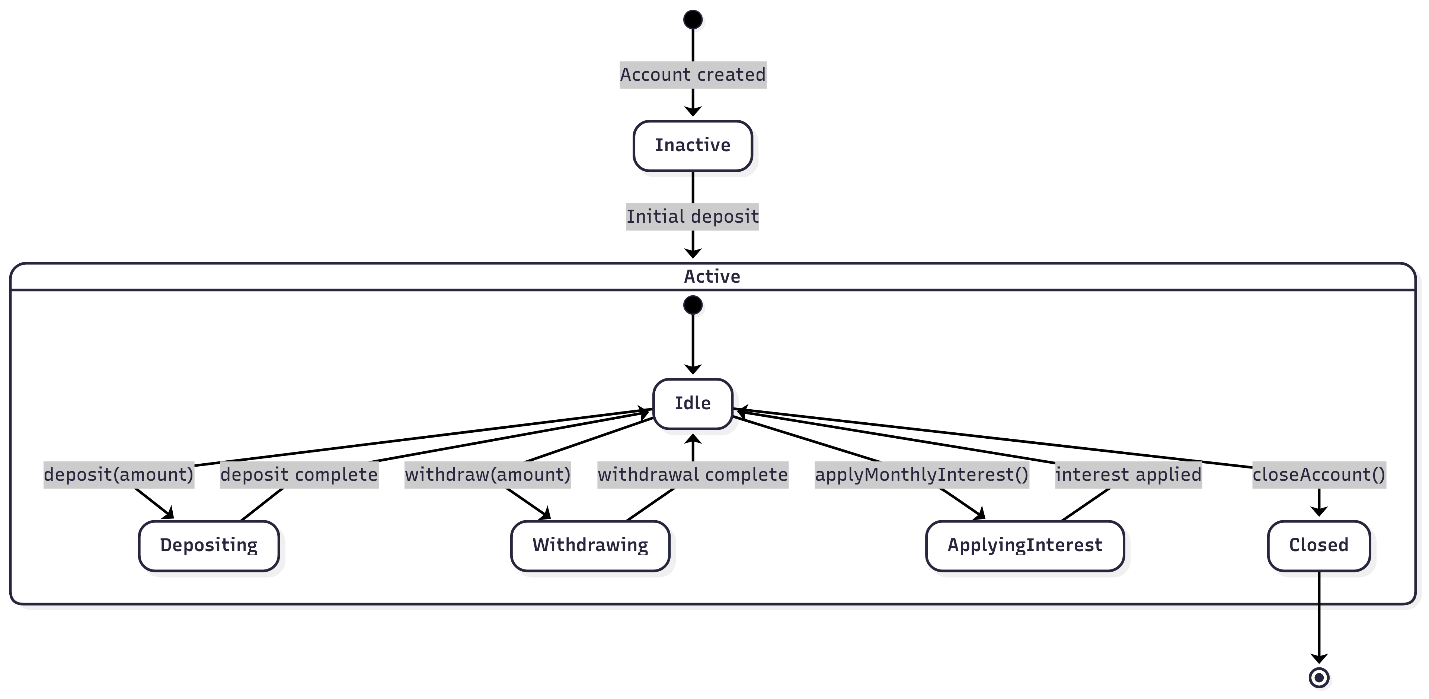
1. CustomerUI sends a deposit(accountNumber, amount) message to AccountController.
2. AccountController finds the correct Account object.
3. AccountController calls account.deposit(amount).
4. The Account object updates its balance.
5. The Account object creates a new Transaction record of type "DEPOSIT".
6. A success confirmation is returned back up the chain to the CustomerUI

**SEQUENCE DIAGRAM : LOGIN**

****

* User enters credentials and clicks login in the UI
* AuthController receives the request and calls CustomerService
* CustomerService uses CustomerRepository to authenticate
* Repository queries the database (FindByUsername)
* Repository returns customer record or null if not found
* Response flows back through the layers to the UI

**STATE DIAGRAM**

****

**Inactive** → When the account is created but has no deposit yet.

**Active** → Once the first deposit is made.

**Depositing** → Temporary state while money is being added.

**Withdrawing** → Temporary state while money is being withdrawn (only for eligible accounts).

**ApplyingInterest** → State when monthly interest is calculated and added.

**Closed** → Final state when the account is closed.

**Appendix**

FILE: [OOAD Elicitation Session-20250918\_103316-Meeting Recording.mp4](https://mybac.sharepoint.com/:v:/s/20252026cse202objectorientedanalysisanddevelopmentwithjav/ETK2KNDOJNNCgI3WOcFy_XUB57nHHAzNSlG9JaieWIF_jg?e=lZSTM1)

**Mock Interview Record**

**Date:** 18 September 2025  
**Mode:** Mock Interview (Teams Online)  
**Participants:**

* **Analyst (Student)**
* **Client (Lecturer/Bank Representative):** Kentsenao Baseki

**Interview (Q&A)**

Q1 (Analyst): Will the actual balance representation (currency, format, etc.) affect the design of the system?  
A1 (Client): Not really. What matters is the class structure and whether it implements encapsulation and proper attributes/methods. The data type can handle values regardless of currency.

Q2 (Analyst): What about limits for withdrawals and deposits since different account types have different rules?  
A2 (Client): Yes, limits are predefined:

* Savings Account: Minimum deposit ~BWP50, no withdrawals allowed.
* Investment Account: Requires a minimum opening balance of BWP500, withdrawals are allowed.
* Cheque Account: Can be opened with balance = 0, withdrawals and deposits allowed, mainly for salaries.

Q3 (Analyst): What happens if a customer tries to withdraw more than the available balance?  
A3 (Client): The system should reject the withdrawal and notify the user (e.g., if balance = 2 pula, trying to withdraw 10 pula should fail).

Q4 (Analyst): Should transaction charges be included?  
A4 (Client): Ideally, deposits and withdrawals could have charges, but for this assignment we exclude transaction charges to keep the scope manageable.

Q5 (Analyst): Are there additional features like authentication, GUI, or database integration required?  
A5 (Client): No, we should avoid scope creep. Focus on OOP concepts (abstraction, inheritance, polymorphism, encapsulation) and core account functions: opening accounts, deposits, withdrawals, and applying interest. GUI and database integration would make the project too complex for the timeframe.