Requirements Elicitation

Functional Requirements

The system will:

- ❖ Allow a new customer to register by providing personal details (first name, surname, address).
- ❖ Authenticate users through a secure login process using a Customer ID and password.
- ❖ Allow an authenticated customer to open an account, enforcing specific rules for each type:
    - ➢ **Savings Account:** No initial deposit required. Withdrawals are not allowed.
    - ➢ **Investment Account:** Requires an initial deposit of at least BWP 500.00 Allows both deposits and withdrawals.
    - ➢ **Cheque Account:** Requires proof of employment (company name, address). Allows deposits and withdrawals.
- ❖ Allow customers to deposit funds into any of their active accounts.
- ❖ Allow customers to withdraw funds only from their Investment or Cheque accounts (not from Savings).
- ❖ Automatically calculate and pay monthly interest to the applicable accounts:
    - ➢ **Investment Account:** 5% monthly interest.
    - ➢ **Savings Account:** 0.05% monthly interest.
    - ➢ **Cheque Account:** No interest.
- ❖ Provide a dashboard for customers to view all their accounts and current balances

Appendice: Mock Interview Record

Interviewer: Student
Interviewee (Client): Lecturer
Date: September 18, 2025
Purpose: To elicit core functional and quality requirements for the new Banking System.

Interview Transcript Excerpt:

Interview Record:

Interviewer: "Thank you for your time, Mr. Baseki. The initial document is very helpful. To ensure we build exactly what you need, I'd like to ask a few clarifying questions. First, regarding customer registration: what specific information is absolutely essential for us to collect about a new customer?"

Mr. Baseki: "Welcome. For our system, we need to know who they are. So, we must have their first name, surname, and physical address. A unique customer ID generated by the system will be sufficient for your project."

Interviewer: Noted. When a customer makes a deposit or withdrawal, what information about that transaction needs to be permanently saved?"

Mr. Baseki: We need a full audit trail for compliance. For each transaction, we must record the date and time, the account it was performed on, the type of transaction—deposit or withdrawal the amount involved, and the new balance of the account after the transaction is complete.

Interviewer: Perfect. The interest rates are specified. How should this interest be applied? Is it on a specific day of the month?

Mr. Baseki: For your system, we don't need anything complex. We simply need the functionality to apply the correct interest to the correct accounts. You can have a special command or method, perhaps called `applyMonthlyInterest()`, that we can run to update all account balances. This demonstrates the logic correctly."

Interviewer: That makes the development clearer. Finally, what are the key things a customer should be able to see about their accounts?

Mr. Baseki: A customer should be able to see a summary of all their accounts with us, showing the account number, the type of account it is, and the current balance in each. Furthermore, they should be able to select an account and see its full history the list of all transactions we just discussed.

Interviewer: "Thank you, Mr. Baseki. This has been extremely valuable and has clarified all the functional points we need to develop. We will proceed based on this."

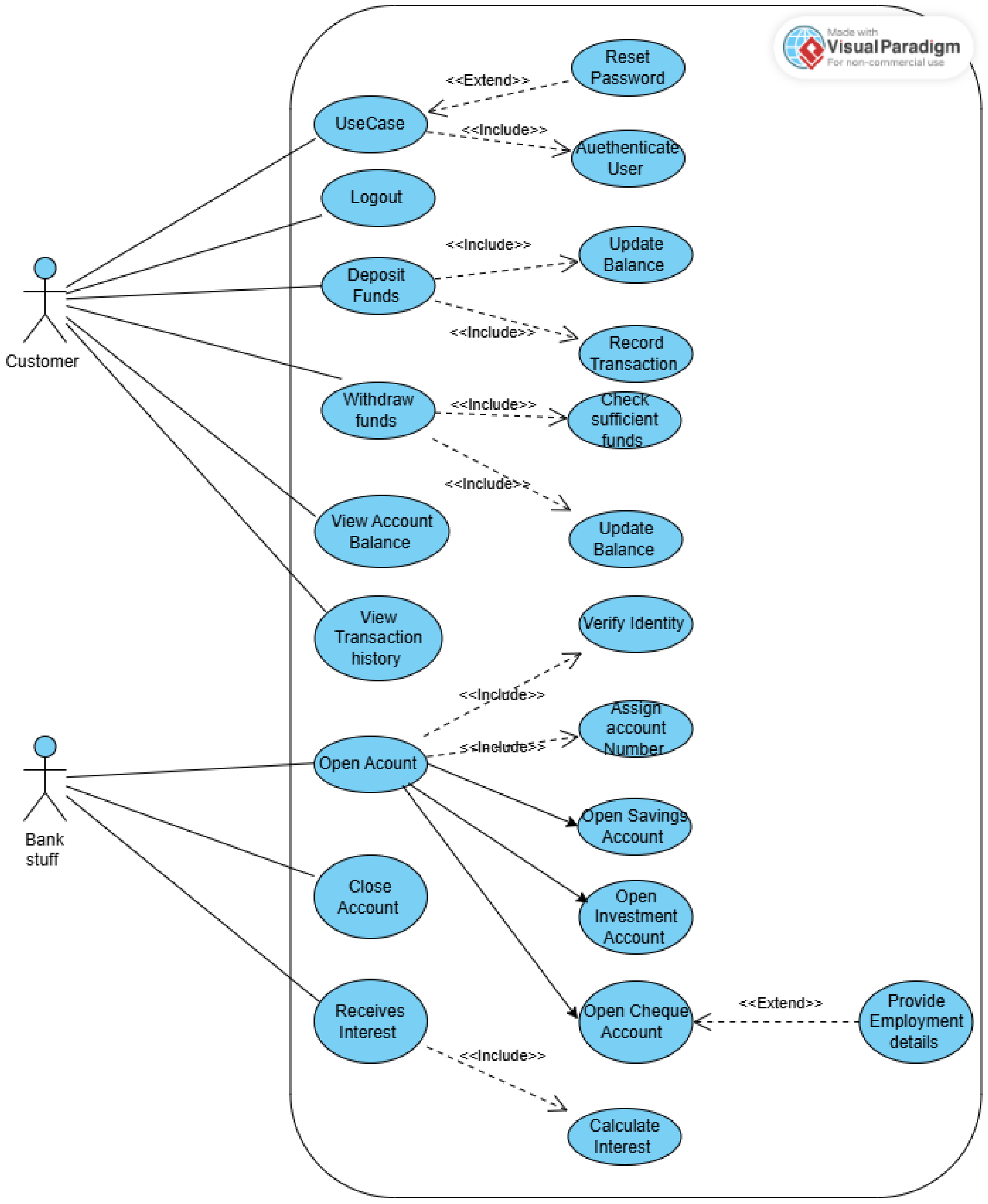
Mr. Baseki: "You are welcome. I look forward to seeing your design.

Non-Functional Requirements

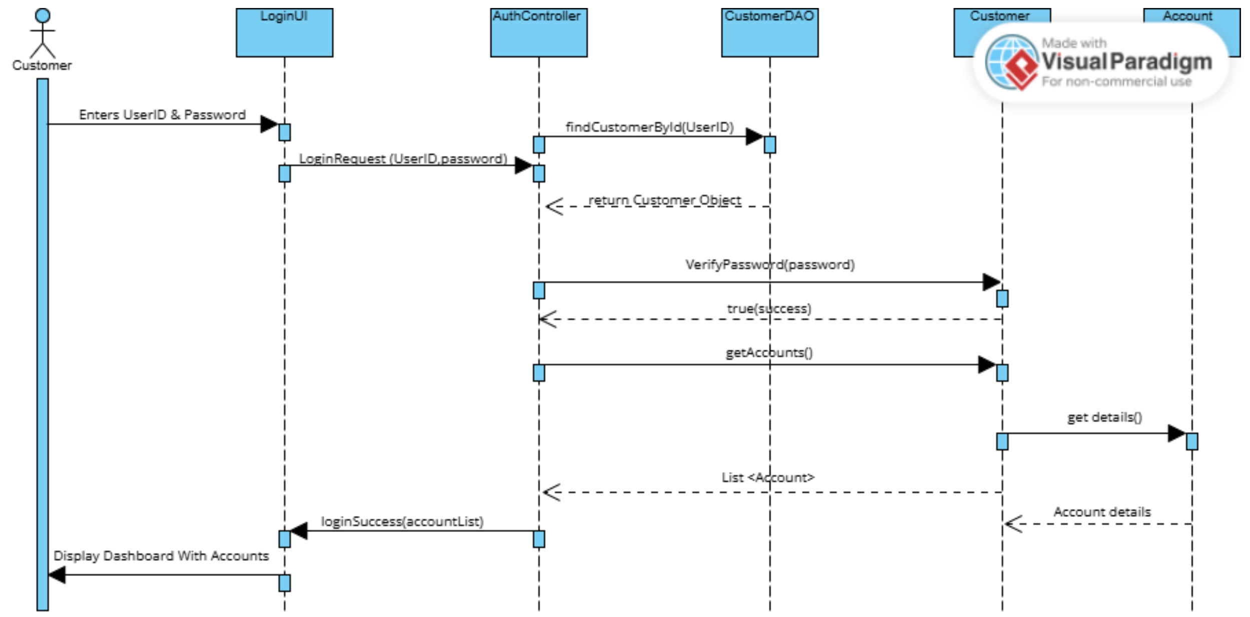
The system shall be:

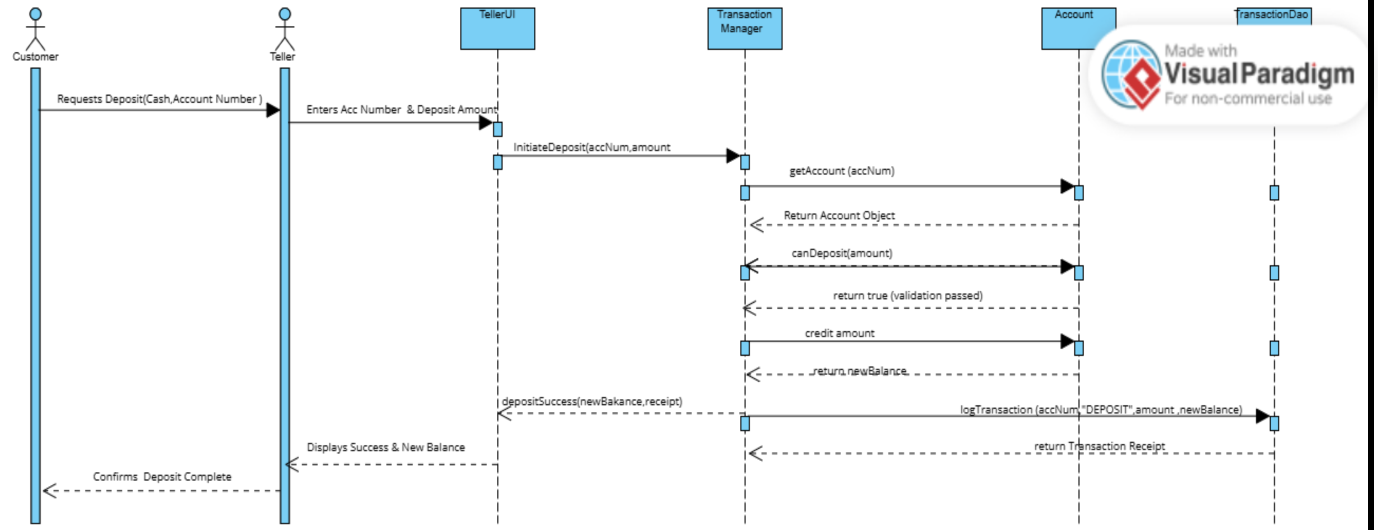
1. **Secure:** All passwords must be stored in a hashed format. Authentication is required for all actions.
2. **Performance:** The system must respond to user inputs (e.g., login, transactions) in a short time
3. **Usable:** The interface must be intuitive and easy to navigate for non-technical users, with clear error messages.
4. **Reliable & Available:** The system must be highly available and process all transactions accurately without failure.
5. **Maintainable:** The code must be well-structured and follow OOP principles to allow for easy future updates and expansion.

Use Case Diagram

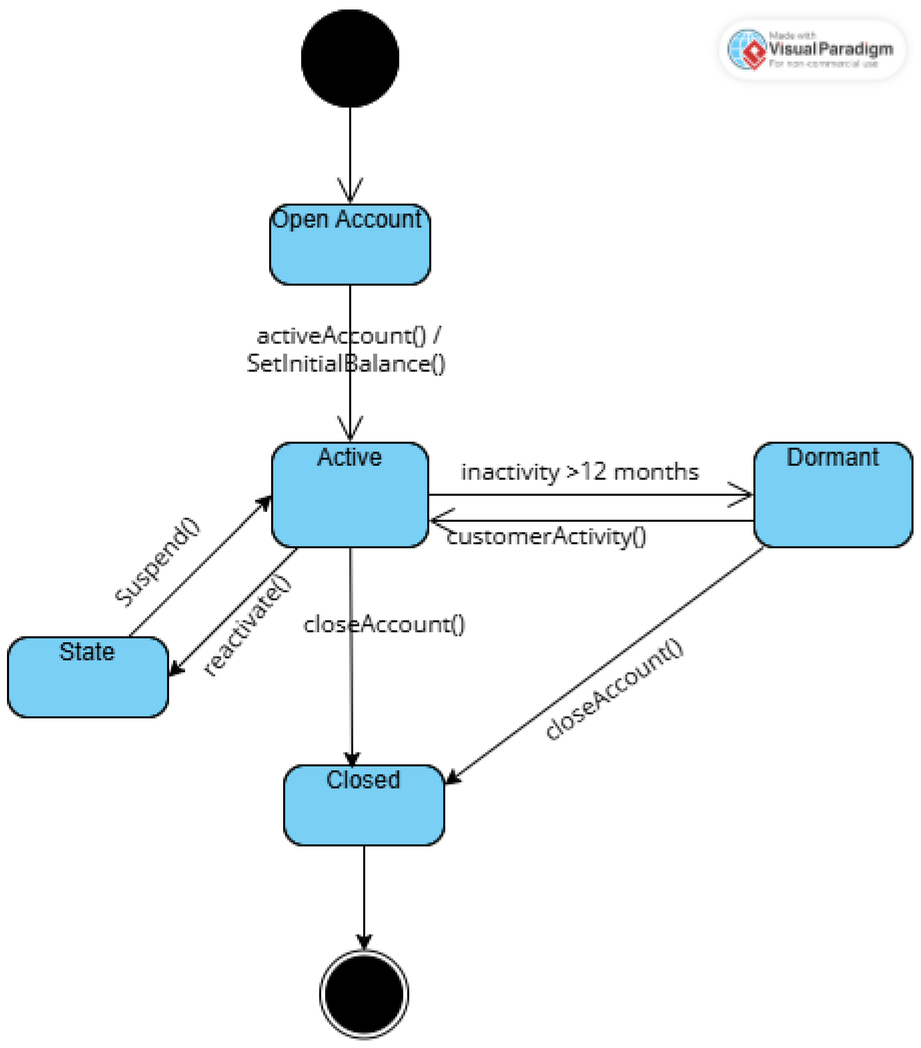


Sequence Diagrams Login & Deposit Funds

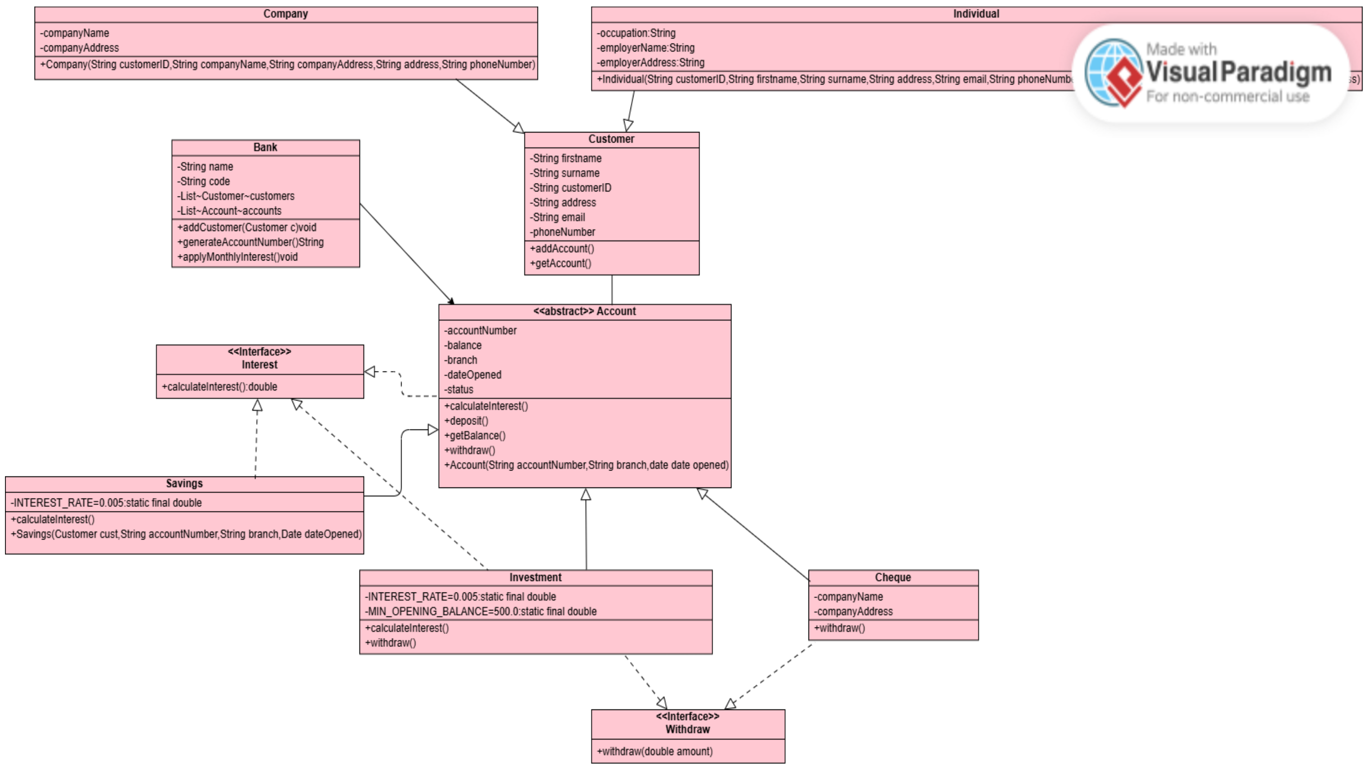



Customer
Teller
TellerUI
Transaction Manager
Account
TransactionDao
Made with Visual Paradigm For non-commercial use
Requests Deposit(Cash,Account Number)
Enters Acc Number & Deposit Amount
InitiateDeposit(accNum,amount
getAccount (accNum)
Return Account Object
canDeposit(amount)
return true (validation passed)
credit amount
return newBalance
depositSuccess(newBakance,receipt)
logTransaction (accNum,"DEPOSIT",amount ,newBalance)
return Transaction Receipt
Displays Success & New Balance
Confirms Deposit Complete

State Diagram



Class Diagram


Company
-companyName
-companyAddress
+Company(String customerID,String companyName,String companyAddress,String address,String phoneNumber)
Individual
-occupation:String
-employerName:String
-employerAddress:String
+Individual(String customerID,String firstname,String surname,String address,String email,String phoneNumb
ss)
Made with Visual Paradigm For non-commercial use
Bank
-String name
-String code
-List~Customer~customers
-List~Account~accounts
+addCustomer(Customer c)void
+generateAccountNumber()String
+applyMonthlyInterest()void
Customer
-String firstname
-String surname
-String customerID
-String address
-String email
-phoneNumber
+addAccount()
+getAccount()
<<abstract>> Account
-accountNumber
-balance
-branch
-dateOpened
-status
+calculateInterest()
+deposit()
+getBalance()
+withdraw()
+Account(String accountNumber,String branch,date date opened)
<<Interface>> Interest
+calculateInterest():double
Savings
-INTEREST_RATE=0.005:static final double
+calculateInterest()
+Savings(Customer cust,String accountNumber,String branch,Date dateOpened)
Investment
-INTEREST_RATE=0.005:static final double
-MIN_OPENING_BALANCE=500.0:static final double
+calculateInterest()
+withdraw()
Cheque
-companyName
-companyAddress
+withdraw()
<<Interface>> Withdraw
+withdraw(double amount)