**Appendix 1: Assignment submission cover sheet ASSIGNMENT SUBMISSION COVER SHEET**

**Student Id: CSE24-027**

**Student names: Erykah Neelo Natefo Daman**

**Student email**:

cse24-027@thuto.bac.ac.bw

**Cohort**: SCIS

**Assignment title**: OOAD Assignment 2025

**Date of submission**: 18/9/2025

**Programme of Study**:

Computer Systems Engineering

Year 2

**Year of Study**:

**Intellectual property statement**

By checking the box below, I certify that this assignment is my own work and is free from plagiarism. I understand that the assignment may be checked for plagiarism by electronic or other means and may be transferred and stored in a database for the purposes of data-matching to help detect plagiarism. The assignment has not previously been submitted for assessment in any other unit or to any other institution. **I**

**have read and understood the Botswana Accountancy College plagiarism guidelines policy.**

* Agree **Signature**……………………………………. **Date**………………………………………….



18/09/2025



**BAC.ACAD.PM.03.F02.V2.0** 3

# Requirements Elicitation

## Functional Requirements

**Q1 :** What are the core services the banking system should provide to customers?  
**A1 :** Customer s should be able to register. When registering, they can choose which type of accounts they want. Since customers can be people or companies, the system should capture personal or company details.

**Q2 :** After registration, what should customers be able to do in the system?  
**A2 :** They should be able to log in and log out. After logging in, they can search for or select an account they want to use, since one customer can have multiple accounts.

**Q3 :** What actions should customers be able to perform on their accounts?  
**A3 :** They can deposit money, withdraw money (depending on the account type), and view their account balance.

**Q4 :** Should the system show customers their transaction history?  
**A4 :** Yes, customers should be able to pick a date range and see all transactions for that account. The system should show the type of transaction (deposit/withdrawal), the amount, the account, and the date and time.

**Q5 :** Will customers be able to access all their accounts after login?  
**A5 :** Yes, after login, they should be able to see and use all their accounts.

**Q6 :** Should the system allow money to move between accounts?  
**A6 :** Yes, for example, when closing a savings account, the money should be transferable into another account.

**1.2 Non-Functional Requirements**

**Q7 :** How should the system handle authentication and passwords?  
**A7 :** The system will use a username and password login. Passwords should be at least 8 characters long with a mix of letters and numbers.

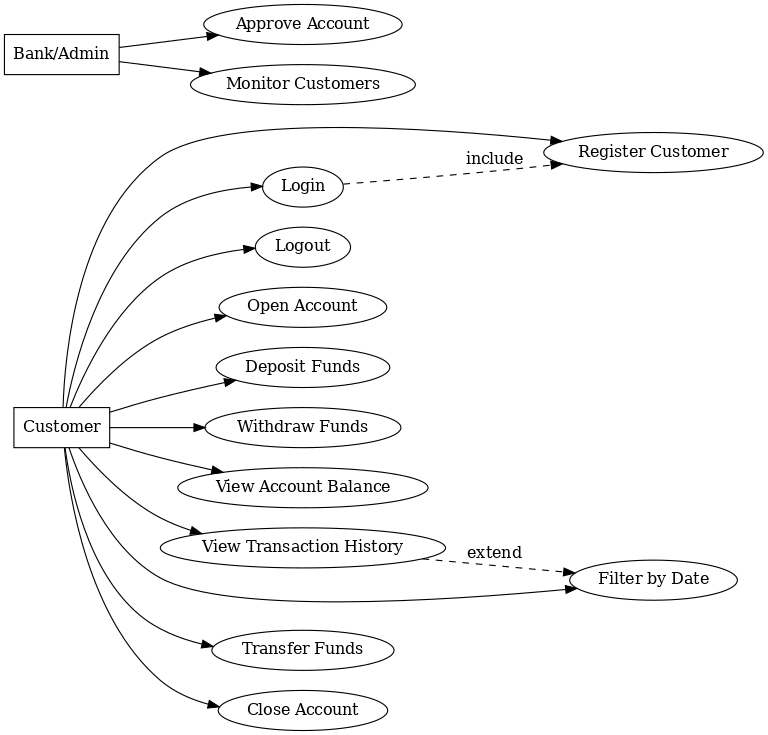
**Q8 :** Do we need stronger authentication, like two-factor login?  
**A8 :** Two-factor would be ideal, but since we don’t have email or SMS gateways, for now we will just use username and password.

**Q9 :** How should sensitive data like passwords be stored in the database?  
**A9 :** Data should be encrypted. If someone looks directly into the database, they should not see plain text, only scrambled (encrypted) values.

**Q10 :** Any other qualities the system should have?  
**A10 :** It should be simple and user-friendly so that customers can easily interact with their accounts.

# Structural UML Modeling

## System Use Case Diagram



The Use Case Diagram for the Banking System shows how different users interact with the system. There are two main actors: **Customer** and **Bank/Admin**.

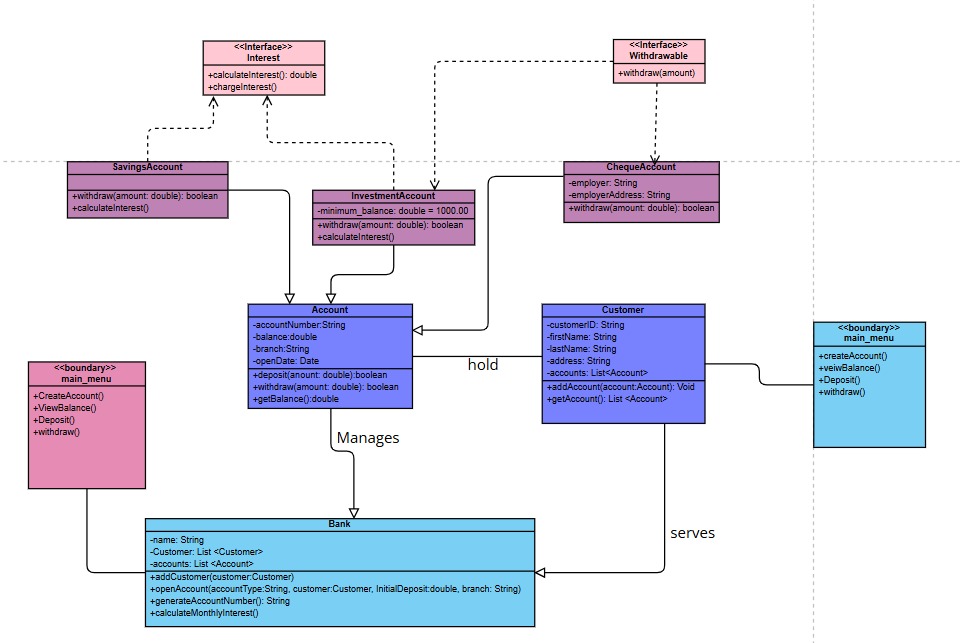
* The **Customer** can register, log in, and manage multiple accounts. Once logged in, they can open accounts (Savings, Investment, or Cheque), deposit and withdraw funds (depending on account type), view balances, and access transaction history. Customers can also filter their history by date, transfer funds between accounts, and close accounts with the option to move funds into another account.
* The **Bank/Admin** actor is responsible for approving accounts and monitoring customer activities.

The diagram also shows relationships between use cases:

* **Login** includes authentication before accessing other features.
* **View Transaction History** can be extended with the option to filter results by date range.

This diagram gives a high-level overview of the system’s external behavior and how users will interact with it.

## 2.2 Class Diagram

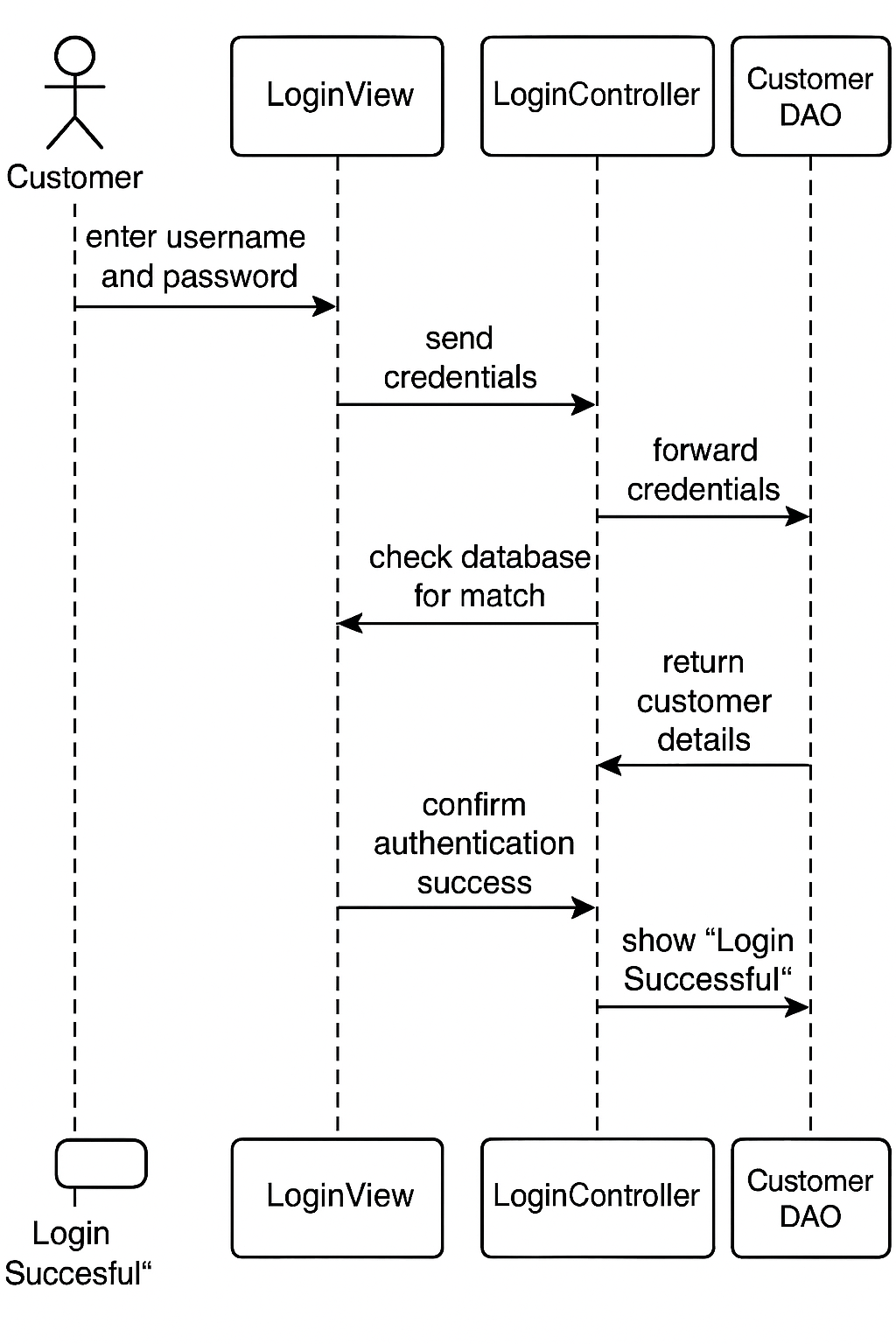


The class diagram models the Banking System with the main classes, their attributes, methods, and relationships.

* **Bank** manages customers and accounts. It can add customers, open accounts, generate account numbers, and calculate monthly interest.
* **Customer** holds personal details and a list of accounts. A customer can add or view accounts.
* **Account** is a base class with attributes like account number, balance, branch, and open date. It provides deposit, withdraw, and balance methods.
* **SavingsAccount**, **InvestmentAccount**, and **ChequeAccount** inherit from Account.
  + *Savings* and *Investment* implement the **Interest** interface for calculating interest.
  + *Cheque* implements the **Withdrawable** interface for withdrawals and includes employer details.
* **Boundary classes (main\_menu)** represent the user interface, allowing account creation, deposits, withdrawals, and balance checks.
* **Abstraction:** Interfaces (Interest, Withdrawable) define common behaviors.
* **Inheritance:** Account types extend the Account class.
* **Polymorphism:** Different account types implement withdraw and interest in their own way.
* **Encapsulation:** Attributes like balance and customer details are private, accessed through methods.

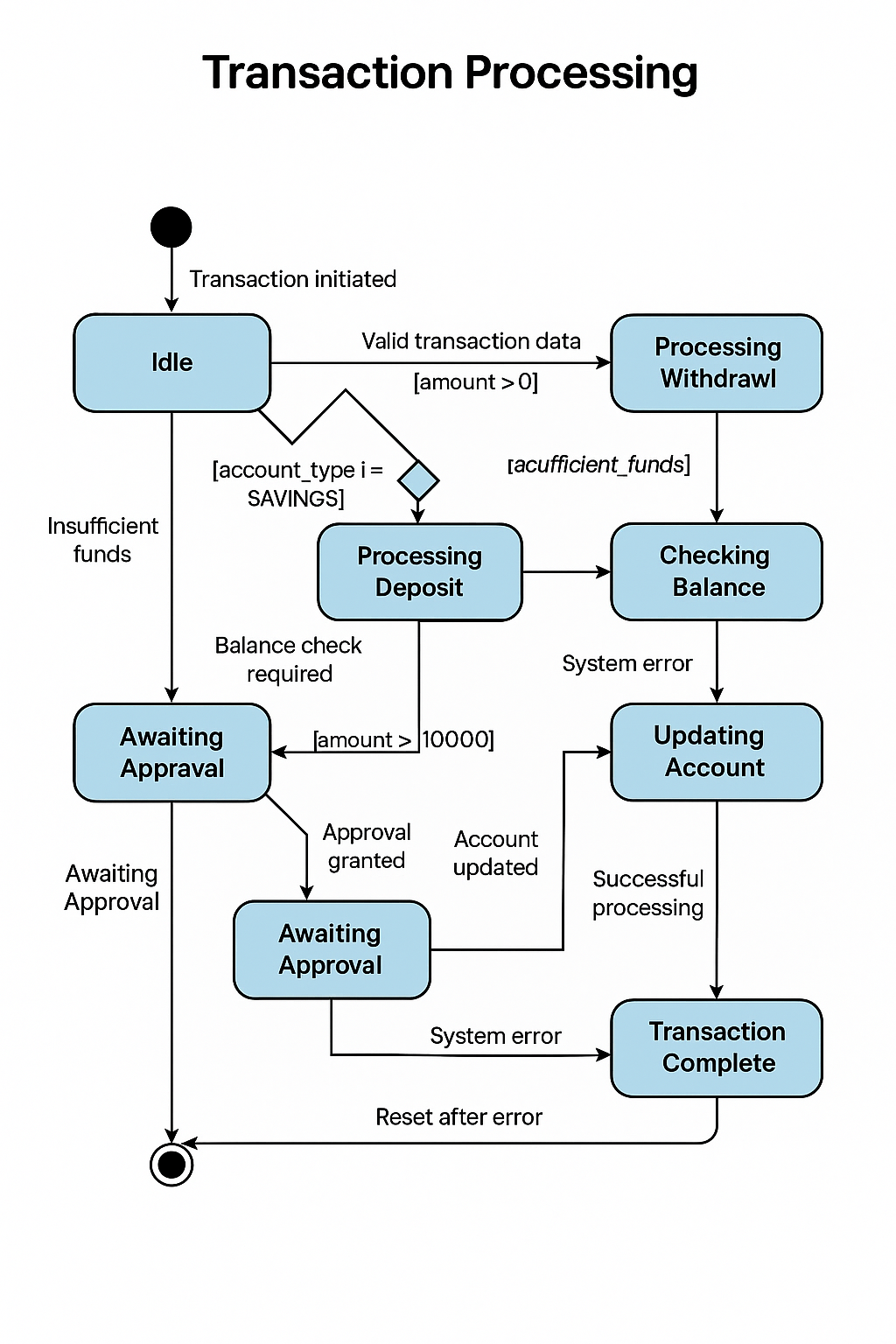
# **Behavioral UML Modelling**

## Sequence Diagrams:



This sequence diagram shows a successful user login process. A customer enters their credentials into the LoginView, which sends them to the LoginController. The controller asks the Customer DAO to check the database for a match. The DAO returns the customer's details, confirming authentication. The controller then informs the LoginView to display a "Login Successful" message to the user.

## **3.2 State Diagram**



This state diagram models the lifecycle of a transaction in a banking system. The process begins **Idle** until a transaction is initiated.

The system first checks for **Valid transaction data** (e.g., amount > 0). It then branches based on the transaction type:

* **Withdrawals** are checked for sufficient funds.
* **Deposits** proceed directly.

For transactions over a certain threshold (e.g., **amount > 10,000**), the system moves to an **Awaiting Approval** state until manual approval is granted.

Once validated (and approved if necessary), the system enters **Updating Account** before reaching the **Transaction Complete** final state. The diagram also includes error handling paths for **Insufficient Funds** and **System Errors**, which reset the process after the error is handled.

# Requirement Elicitation Interview

A requirement elicitation interview was conducted between the developer (Erykah Neelo Natefo Daman) and the Lecturer/Tutor (Themba Moeng). The interview took place to define the core functional and non-functional requirements for a new banking system.

**Key Details:**

* **Participants:** Erykah Neelo Natefo Daman (Interviewer) and Themba Moeng (Lecturer/tutor).
* **Purpose:** To gather detailed customer and system requirements for the banking application.
* **Duration:** The transcript indicates the meeting occurred and lasted approximately 10 minutes (from 0:03 to 9:56).
* **Date and location**: It happened 18/09/2025, On teams. It started at 1833 and ended at 1843.

**Outcome:** The interview successfully captured critical requirements, including user registration, login/logout, account management (multiple accounts per customer), transaction processing (deposits, withdrawals, balance checks), transaction history viewing, security protocols (password complexity, data encryption), and business rules (preventing overdrafts, maintaining minimum balances).

New chat