Part A: System Documentation

1. Requirements Elicitation

Objective: Elicit and specify the system's requirements to create a solid foundation for development.

1.1. Functional Requirements:

Functional Requirements define what a system should do to meet its purpose, outlining specific tasks and actions it must perform for users. For the Banking System, these requirements were established through an interview with a client.

Requirements for Customers

Customer Management

This module allows customers to manage their personal lifecycle within the system, including registration and profile updates.

ID	Requirement Description	Priority
CM-001	The system shall allow customers to register by providing personal details (first name, surname, address, ID).	High
CM-002	The system shall allow customers to validate registration.	High
CM-003	The system shall allow customers to support multiple customer types.	High
CM-004	The system shall allow customers to update their profiles.	High

CM-005	The system shall allow bank staff to manage customer profiles.	High

Account Management

This module allows customers to manage their account operations, including opening and viewing accounts.

ID	Requirement Description	Priority
AM-001	The system shall allow customers to open Savings, Investment (min BWP500), or Cheque accounts.	High
AM-002	The system shall allow customers to calculate interest (0.05% for Savings, 5% for Investment) monthly.	High
AM-003	The system shall allow customers to display account details in a dashboard.	High
AM-004	The system shall allow customers to enforce customer-account composition.	High
AM-005	The system shall allow bank staff to approve/suspend accounts.	High
AM-006	The system shall allow bank staff to enable account closure.	High

Transaction Processing

This module allows customers to perform financial transactions.

ID	Requirement Description	Priority
TP-001	The system shall allow customers to process deposits with real-time updates.	High
TP-002	The system shall allow customers to process withdrawals (Investment/Cheque) with fund checks.	High
TP-003	The system shall allow customers to support intra-bank transfers.	High

Transaction History

This module allows customers to access and review their transaction records.

ID	Requirement Description	Priority
TH-001	The system shall allow customers to log and display transaction history.	High

Reporting

This module allows the system to generate transaction reports for bank staff.

ID	Requirement Description	Priority
RA-001	The system shall allow bank staff to generate transaction reports.	High

Security Authentication

This module allows the system to enforce authentication controls for bank staff.

ID	Requirement Description	Priority
SA-001	The system shall allow bank staff to enforce role-based authentication.	High

1.2. Non-Functional Requirements:

A clear description of non-functional features, derived from the interview, ensuring security, performance, and usability.

Security

This module allows the system to ensure secure handling of customer data.

ID	Requirement Description	Priority
SC-001	The system must allow customers to encrypt sensitive data.	High
SC-002	The system must allow customers to secure authentication.	High
SC-003	The system must allow bank staff to secure staff data and logs.	High

Performance

This module allows the system to optimize response times for customer interactions.

ID	Requirement Description	Priority
PC-001	The system shall allow customers to respond within 2 seconds for 100 concurrent users.	High
PC-002	The system shall allow bank staff to respond within 5 seconds for reports.	High

Usability

This module allows the system to enhance the user interaction experience for customers.

ID	Requirement Description	Priority
UC-001	The system shall allow customers to provide an intuitive JavaFX interface.	High
UC-002	The system shall allow customers to display error messages.	High

Reliability & Scalability

This module allows the system to guarantee uptime and growth capacity for customers.

ID	Requirement Description	Priority
RC-001	The system shall allow customers to ensure 99.9% uptime and scalability.	High

Maintainability

This module allows the system to support code maintenance through structured design for bank staff.

ID	Requirement Description	Priority
MS-001	The system must allow bank staff to follow MVC for modularity.	High

2. Structural UML Modelling

Objective: Visually model the static structure of the system using UML diagrams.

2.1. System Use Case Diagram

The Use Case Diagram illustrates the interactions between actors and the system, capturing the external behaviour from the user's perspective. It identifies primary actors and their goals through use cases, along with relationships such as includes and extends for modularity and optional behaviours.

Actors:

- **Customer**: Manages personal banking (accounts, transactions).
- **Bank Staff**: Handles administration (approvals, reports).
- **System**: Automates processes (e.g., monthly interest calculation).

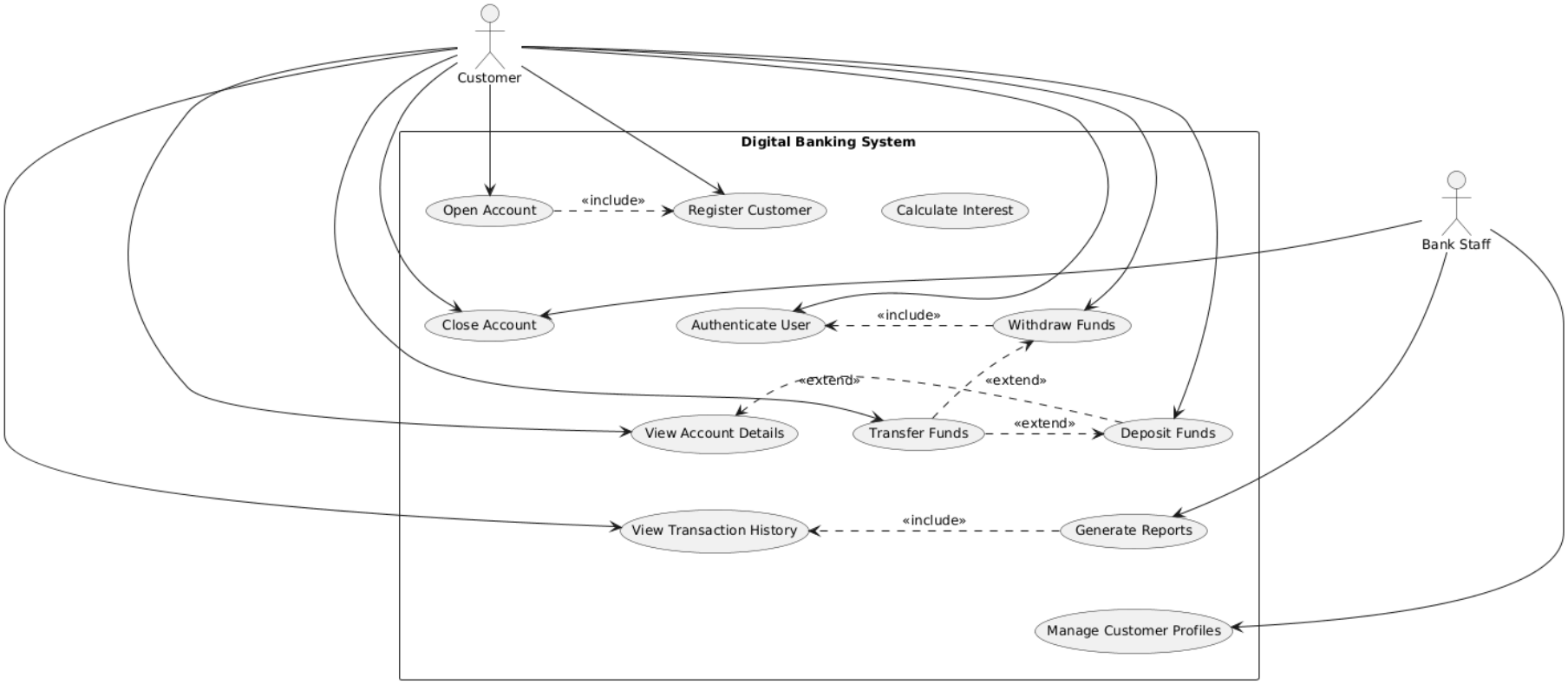
Use Cases:

- **Register Custome**r: Create profile with personal details.
- **Authenticate User**: Verify credentials with role-based access.
- **Open Account**: Open Savings, Investment (min BWP500), or Cheque (employment details required).
- **Deposit Funds:** Add funds to any account, update balance real-time.
- **Withdraw Funds**: From Investment/Cheque, with sufficient funds check.
- **Transfer Funds:** Intra-account/customer transfers.
- **View Account Details:** Display dashboard with balances and types.
- **View Transaction History:** Retrieve past transaction logs.
- **Calculate Interest:** Apply 0.05% to Savings, 5% to Investment monthly.
- **Generate Reports:** Create transaction/account reports (staff-only).
- **Manage Customer Profiles:** Update/approve/suspend data (staff-only).
- **Close Account:** Verify and close account (staff-approved).

Relationships:

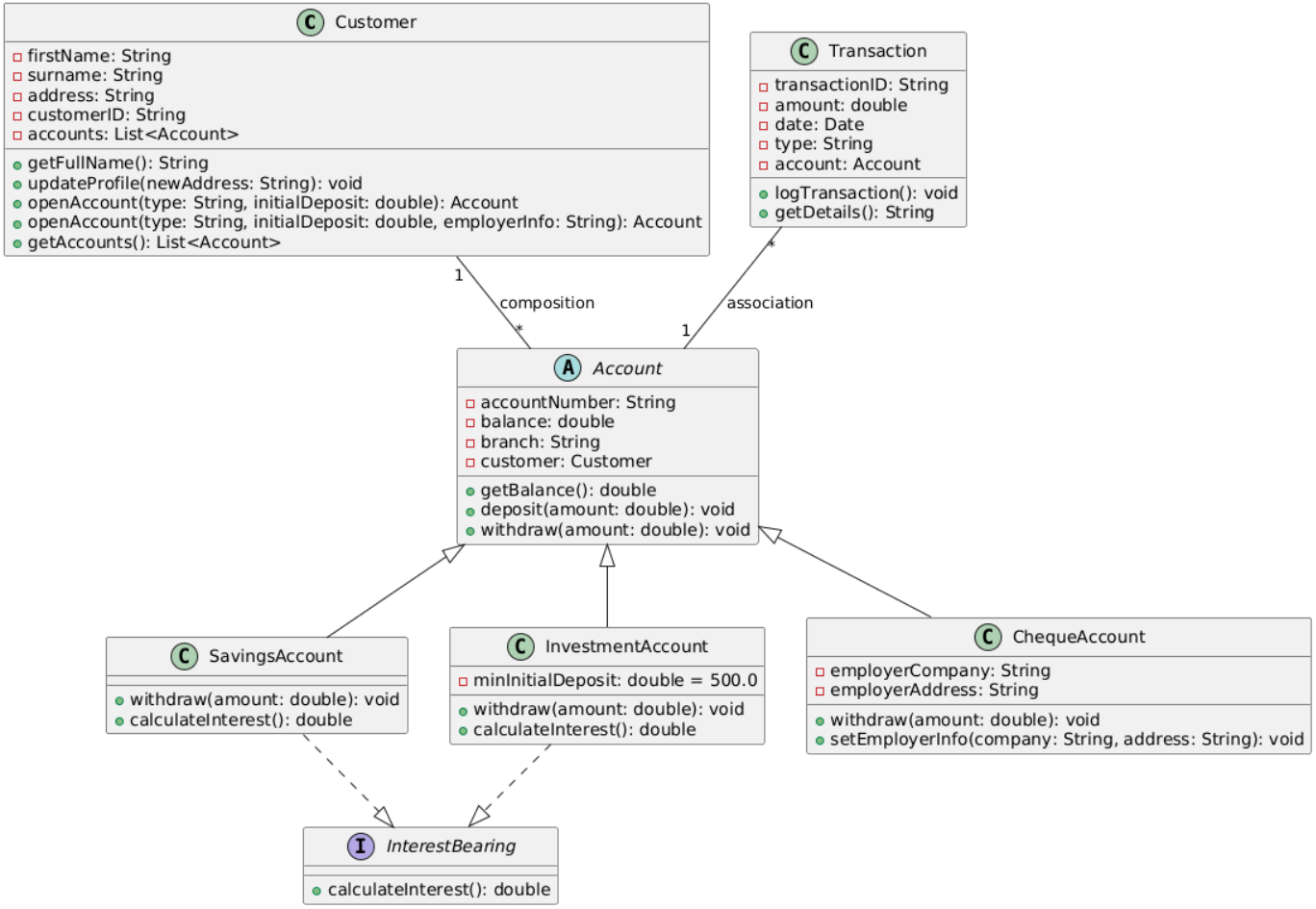
- **Includes**: Open Account includes Register Customer; Withdraw Funds includes Authenticate User; Generate Reports includes View Transaction History.

- **Extends**: Deposit Funds extends View Account Details; Transfer Funds extends Withdraw Funds and Deposit Funds.



2.2. Class diagram

The Class Diagram shows classes, attributes, methods, and relationships, incorporating OOP principles: abstraction (abstract Account), inheritance (subclasses), interface (InterestBearing), encapsulation (private attributes), and composition (Customer-Accot).



- **Customer** (Concrete Class - Represents a bank customer with multiple accounts.)
    - Attributes: -firstName: String, -surname: String, -address: String, -customerID: String, -accounts: List<Account>
    - Methods: +getFullName(): String, +updateProfile(String): void, +openAccount(String, double): Account, +openAccount(String, double, String): Account, +getAccounts(): List<Account>

- **Account** (Abstract Class - Base class for all account types.)
    - Attributes: -accountNumber: String, -balance: double, -branch: String, -customer: Customer
    - Methods: +getBalance(): double, +deposit(double): void, +withdraw(double): void (abstract)

- **SavingsAccount** (Concrete Class - Extends Account, implements InterestBearing; no withdrawals.)
    - Attributes: (Inherits from Account)
    - Methods: +withdraw(double): void (no action), +calculateInterest(): double (0.05%)

- **InvestmentAccount** (Concrete Class - Extends Account, implements InterestBearing; requires minimum deposit.)
    - Attributes: (Inherits from Account), -minInitialDeposit: double = 500.0
    - Methods: +withdraw(double): void (check balance), +calculateInterest(): double (5%)

- **ChequeAccount** (Concrete Class - Extends Account; requires employment info.)
    - Attributes: (Inherits from Account), -employerCompany: String, -employerAddress: String
    - Methods: +withdraw(double): void (allow if sufficient), +setEmployerInfo(String, String): void

- **Transaction** (Concrete Class - Tracks account transactions.)
    - Attributes: -transactionID: String, -amount: double, -date: Date, -type: String, -account: Account
    - Methods: +logTransaction(): void, +getDetails(): String

- **InterestBearing** (Interface - Defines interest calculation.)
    - Attributes: (None)
    - Methods: +calculateInterest(): double

Relationships:

- Composition: Customer 1 to * Account
- Inheritance: Arrows from SavingsAccount, InvestmentAccount, ChequeAccount to Account
- Implementation: Dashed arrows from SavingsAccount, InvestmentAccount to InterestBearing
- Association: Transaction * to 1 Account

3. Behavioural UML Modelling

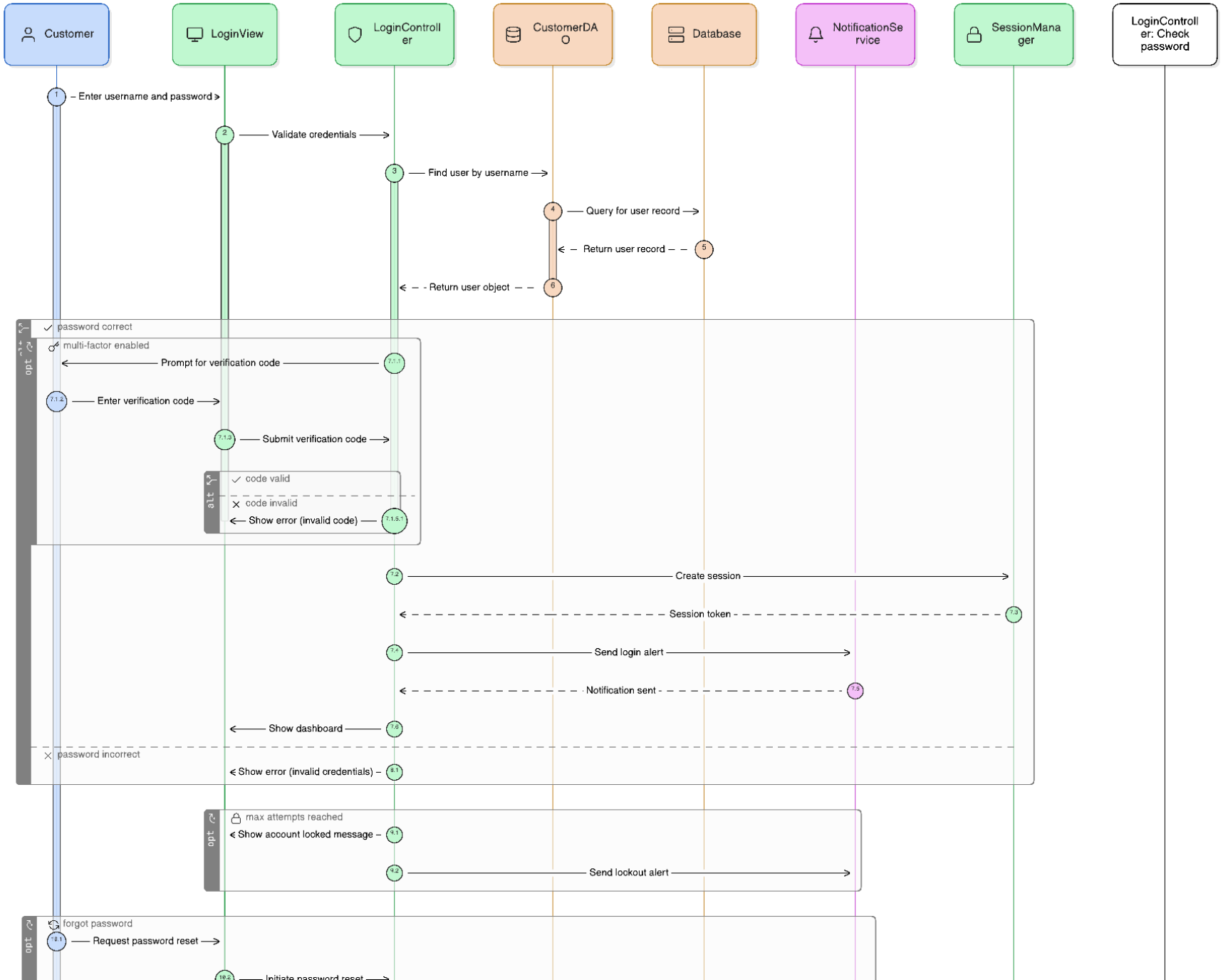
Objective: Model the dynamic behaviour of the system and how objects interact over time.

3.1. Sequence Diagrams:

- **Accurate and logical flow of messages between objects**

  Sequence Diagram 1: Login

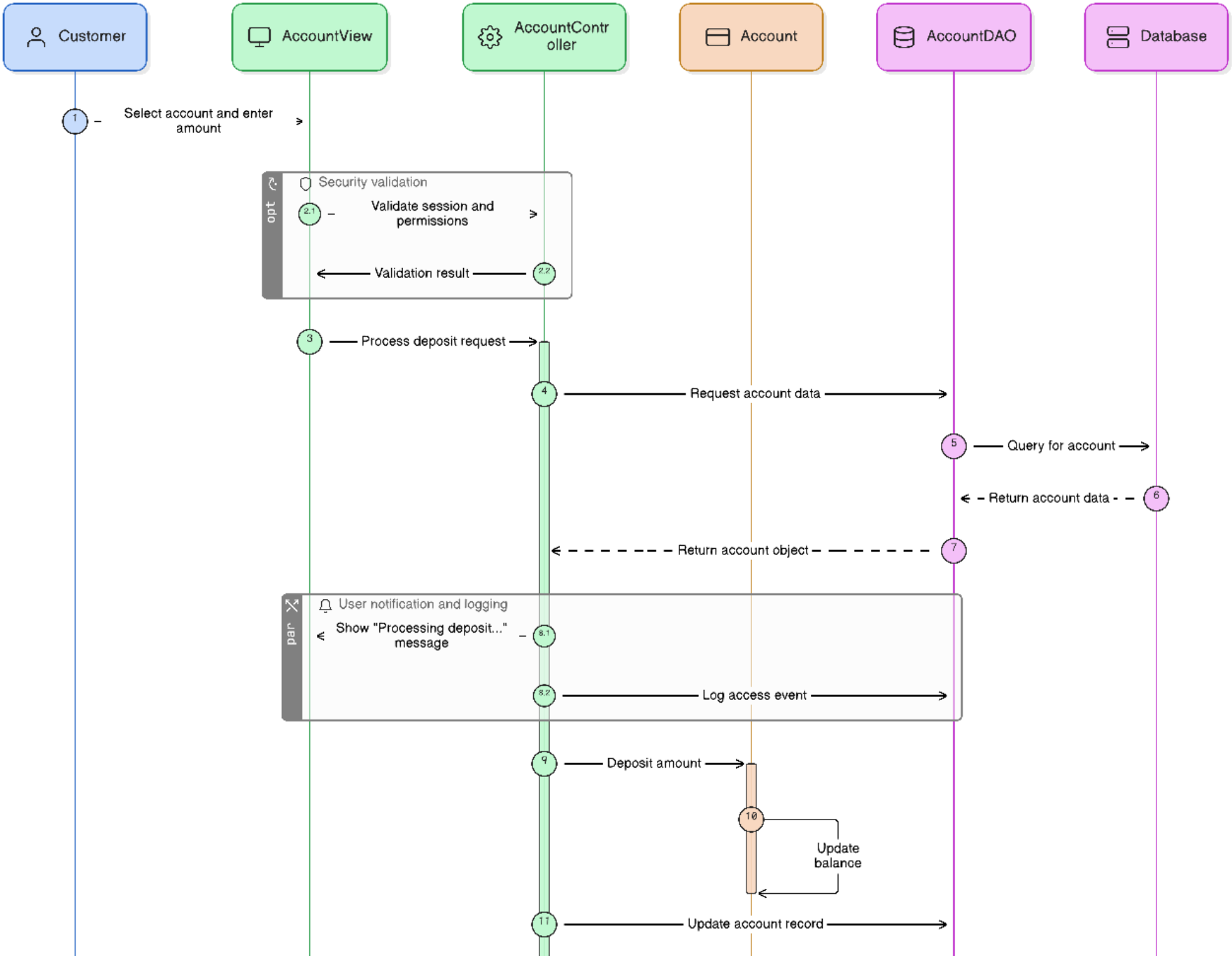
  This diagram shows how a **Customer** logs into the system.
- The **LoginView** takes the username and password.
- The **LoginController** asks the **CustomerDAO** to validate the credentials against the **Database**.
- If correct, the system handles an optional **Multi-Factor Authentication (MFA)** step.
- Upon successful verification, the **SessionManager** creates a session token.
- The user is then shown the **Dashboard**.
- The diagram also includes flows for a **forgot password** request and an **account lockout** if maximum attempts are reached.


Customer
LoginView
LoginControll er
CustomerDA O
Database
NotificationSe rvice
SessionMana ger
LoginControll er: Check password
1 – Enter username and password
2 Validate credentials
3 Find user by username
4 Query for user record
5 Return user record
6 Return user object
opt
password correct
multi-factor enabled
Prompt for verification code
7.1.1
7.1.2 Enter verification code
7.1.3 Submit verification code
alt
code valid
code invalid
Show error (invalid code)
7.1.5.1
7.2 Create session
Session token
7.3
7.4 Send login alert
Notification sent
7.5
Show dashboard
7.6
password incorrect
Show error (invalid credentials)
8.1
opt
max attempts reached
Show account locked message
9.1
9.2 Send lockout alert
opt
forgot password
10.1 Request password reset
10.2 Initiate password reset

Sequence Diagram 2: Deposit Funds

1. This diagram details the transaction flow when a **Customer** deposits money.

2. The **AccountView** sends the deposit amount to the **AccountController**.
3. The **AccountController** performs **security validation** and retrieves the account record from the **Database** via the **AccountDAO**.
4. The controller updates the balance on the **Account** object (the core model).
5. Simultaneously, the system logs the event and shows a processing message (**par** fragment).
6. The **AccountController** tells the **AccountDAO** to save the updated balance to the **Database**.
7. A deposit success message is displayed on the **AccountView**.

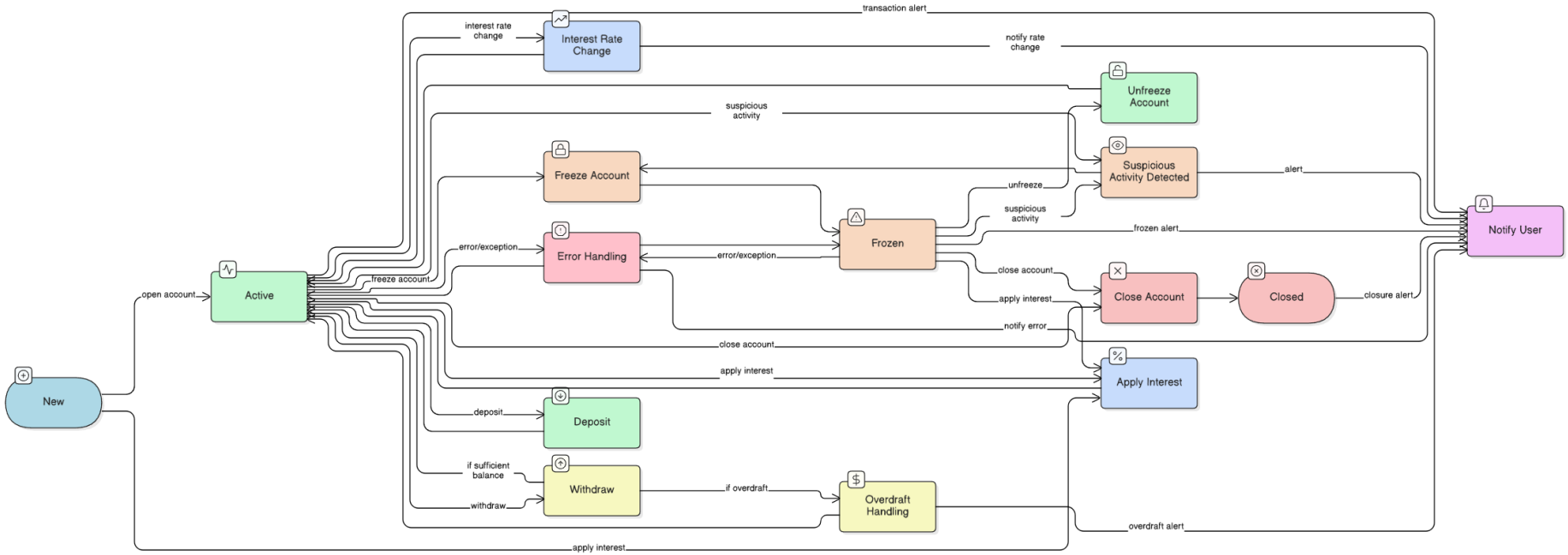

Customer
AccountView
AccountController
Account
AccountDAO
Database
1
Select account and enter amount
opt
Security validation
2.1
Validate session and permissions
Validation result
2.2
3
Process deposit request
4
Request account data
5
Query for account
Return account data
6
Return account object
7
par
User notification and logging
Show "Processing deposit..." message
8.1
8.2
Log access event
9
Deposit amount
10
Update balance
11
Update account record

3.2. State Diagram:

Select a key class (Account) with a complex lifecycle.

This diagram models the entire life of an **Account** object, showing its states and transition events.

- The account starts in the **New** state and moves to **Active** upon being opened.
- In the **Active** state, the account handles deposit, withdraw, and apply interest events.
- Events like suspicious activity or a major error/exception can transition the account to the **Frozen** state.
- A **Frozen** account can be returned to **Active** via an **Unfreeze** event.
- If a withdrawal causes insufficient balance, the account enters **Overdraft Handling**.
- The account moves to the **Closed** state upon a close account event from either **Active** or **Frozen**.
- Various events trigger the **Notify User** final action.


New
Active
Interest Rate Change
Freeze Account
Error Handling
Deposit
Withdraw
Frozen
Unfreeze Account
Suspicious Activity Detected
Close Account
Closed
Apply Interest
Overdraft Handling
Notify User
open account
interest rate change
transaction alert
notify rate change
suspicious activity
freeze account
error/exception
deposit
if sufficient balance
withdraw
if overdraft
unfreeze
suspicious activity
alert
frozen alert
close account
apply interest
notify error
closure alert
overdraft alert
apply interest
close account
apply interest

ASSIGNMENT COVER PAGE: PART A

Module Code: CSE202

Module Title: OBJECT ORIENTED ANALYSIS & DESIGN WITH JAVA

Assignment Title: OBJECT-ORIENTED ANALYSIS AND DESIGN FOR A BANKING SYSTEM (PART A: SYSTEM DOCUMENTATION)

Date of Submission: 13 OCT 2025

Programme of Study: BSc Computer Systems Engineering

Year of Study: Year 2, Semester 1

Student Information

Student ID: FCSE23-018

Student Name: Donovan Ntsima

Student Email:fcse23-018@thuto.bac.ac.bw

Intellectual Property Statement:

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 stored in a database for data-matching purposes. The assignment has not been previously submitted for assessment in any other unit or institution. I have read and understood the Botswana Accountancy College plagiarism guidelines policy.

Agree: [X]

Signature: