

**Banking System**

**1. Requirements Elicitation**

* 1. **Functional Requirements**

The functional requirements define essential services and operations that banking system must provide to meet user’s needs. The following requirements are requirements for the banking system.

**1.Customer Registration & Authentication**

* The system must allow new customers to register by providing personal details , such as name, address , contact information
* Existing customers must be able to log in securely using valid credentials.

**2. Account Management**

* Customers must be able to open multiple accounts (savings, Investment, Cheque).
* Each account must be linked to a customer; accounts cannot exist without a valid customer.
* The system must enforce account-specific rules:

**Savings Account**: Deposits only, earns 0.05% monthly interest, no withdrawals.

**Investment Account**: Initial minimum deposit of BWP500.00, allows deposits and withdrawals, earns 5% monthly interest.

**Cheque Account**: Used for salary deposits, allows deposits and withdrawals, requires employment details at creation.

**3.**  **Interest Calculation**

* The system must automatically calculate and apply monthly interest:
  + 0.05% for Savings Accounts.
  + 5% for Investment Accounts.

**4. Transaction Management**

* Every transaction (deposit, withdrawal, interest payment) must be recorded in a transaction history.
* Customers must be able to view their transaction history for each account.
  1. **Non-Functional Requirements**

**1.Security**

* The system must use authentication methods (e.g. username and password).
* Customer data and transactions must be protected from unauthorized access.

**2.Performance**

* The system must process deposits, withdrawals and balance inquiry in real time response in less than 2 seconds under normal load.
* The system must support multi users without significant degradation of performance.

**2.Usability**

* The user interface must be simple, intuitive, and easy to navigate, ensuring accessibility for non-technical users.
* Error messages must be clear and informative to guide users in correcting mistakes (e.g., insufficient funds during withdrawal).

Requirement Interview Session

Date: 18/09/2024

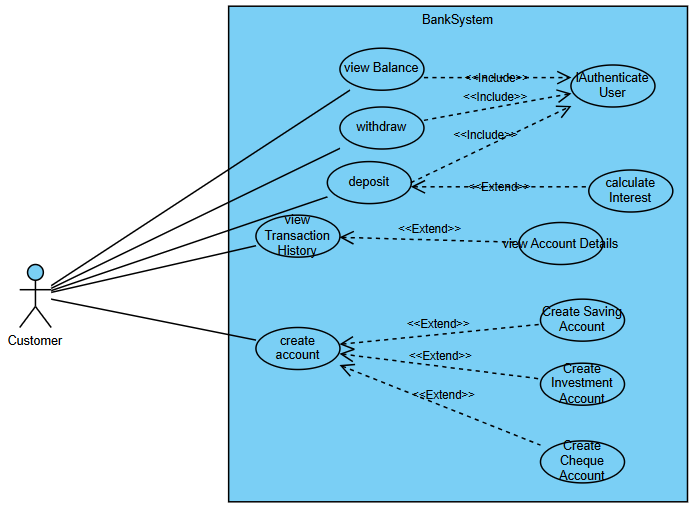
Interviewer Leon Green , interviewee Themba Moeng

Location : teams

Time:11:00 am – 12:00

Agenda: **Bank System** **Requirements**

**2. Structural UML Modelling**

****

**2.1Use Case Diagram Explanation**

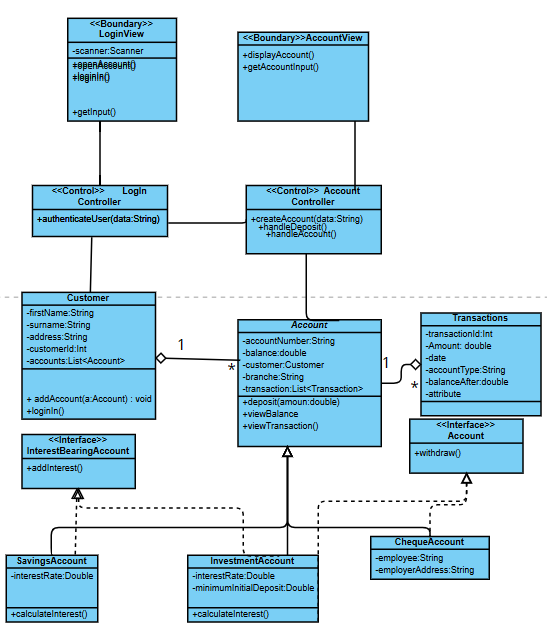
Actors

* Customer: The primary user of the system. This actor interacts with the banking system to perform tasks such as logging in, creating accounts, depositing/withdrawing money, and viewing account details.

Main Use Cases

1. Register
   * A new customer registers into the system.
   * *Includes*: Validation of customer data.
2. Login / Logout
   * Allows a customer to authenticate before accessing services.
   * *Includes*: Authenticate User use case.
3. Create Account
   * Customer creates a new bank account.
   * *Includes*: Customer authentication and validation.
   * *Specialization*:
     + Create Savings Account
     + Create Investment Account
     + Create Cheque Account
4. Deposit Money
   * Customer deposits money into their account.
   * *Includes*: Authenticate User (to ensure only authorized deposits).
   * *Extends*: Apply Interest/Bonus (only for savings or special cases).
5. Withdraw Money *(not shown in your screenshot but usually included)*
   * Customer withdraws money from their account.
   * *Includes*: Authenticate User.
6. View Account Details
   * Customer views details of their account.
   * *Extends*: View Transaction History (optional, only if the user requests).
7. View Transaction History
   * Customer checks past transactions.
   * Triggered as an extension of viewing account details

**2.2 Class Diagram & Diagram Explanation**

****

**Boundary Classes**

* **Login View**
* **Account View**  
  These handle user interaction (input/output). They are marked with <<Boundary>>.

**Control Classes**

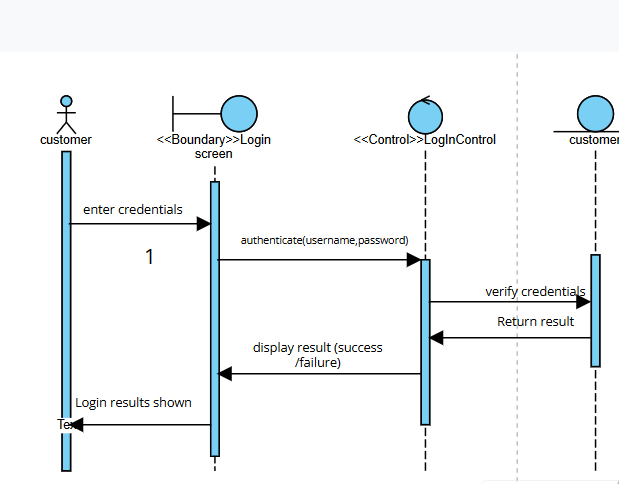
* **Login Controller**
* **Account Controller**  
  These coordinate the flow between boundary and entity classes. They handle the main logic of authentication, account creation, deposits, and withdrawals. They are marked with <<Control>>.

**Entity Classes**

* **Customer** (stores customer data and manages their accounts).
* **Account** (general abstraction of all account types).
* **SavingsAccount, InvestmentAccount, ChequeAccount** (specialized accounts, inherit from Account).
* **InterestBearingAccount** (interface for accounts with interest).
* **Account (interface)** (handles withdraw functionality).

**3.Behavioural UML Modelling)**

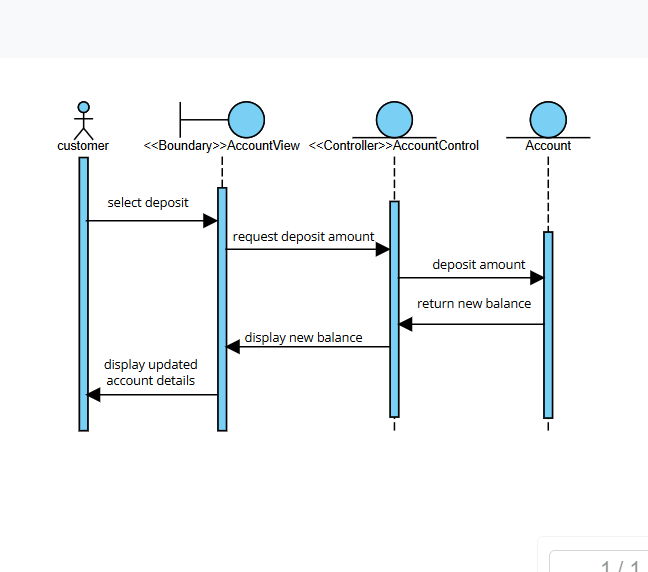
**3.1 Sequence Diagram**



**login sequence Diagram**

* messages start from Actor (User)
* Boundary Class: LoginView
  + Responsible for interacting with the user (display login, get credentials).
* Control Class: LoginController
  + Handles the login request, authenticates user data, and coordinates between LoginView and the Customer entity.

**Deposit Funds sequence diagram**



**User selects deposit option**

* The **Actor (User)** chooses "Deposit Funds" from the interface.
* Message: select deposit → DepositView (<<boundary>>).

**Boundary captures input**

* DepositView asks the user for the deposit amount.
* Message: input amount → DepositControl (<<control>>).

**Control processes request**

* DepositControl receives the deposit amount.
* It validates the input (e.g., must be a positive number).
* Message: send amount → Account (<<entity>>).

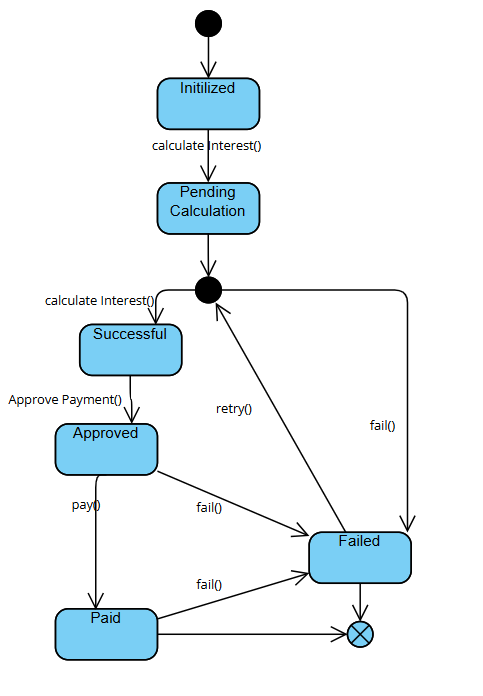
**Entity updates balance**

* Account.deposit(amount) updates the balance in the account.
* A new transaction record may also be created internally (optional: link to Transaction).

**Confirmation flows back**

* Account sends back confirmation (e.g., "Deposit successful, new balance: X").
* DepositControl relays the result → DepositView.
* DepositView shows the updated account details to the **User**

3.2State Diagram – **Pay Interest Class**



The Pay Interest class represents the process of applying interest to accounts. The lifecycle is defined by a series of states and events that trigger transitions between them:

* Initialized – The object is created.
* Pending Calculation – Interest calculation is scheduled but not yet performed.
* Calculated – The interest has been computed.
* Approved – The calculated interest has been reviewed and verified.
* Paid – Interest has been credited to the customer’s account.
* Failed – A failure occurred at any stage (calculation, approval, or payment)

**Part B - System Implementation**