**PRESENTED BY JADEN WILLIAMS (cse24-085)  
  
Banking System Requirements Documentation**

**1.1 Functional Requirements**

**Core System Functions**

**Account Management:**

* Support three account types: Savings (50 pula minimum), Investment (500 pula minimum), Checking (0 pula minimum)
* Bank tellers open accounts and capture customer details (no self-registration)
* Customers can hold multiple account types but only one of each type
* Enforce minimum balance validation during account creation

**Customer Management:**

* Handle two customer types: Individual customers (national ID, personal details) and Company customers (company number, business details)
* Store customer information including addresses, contact details, and type-specific data
* Link customers to their accounts through the account opening process

**Transaction Processing:**

* Enable deposits and withdrawals for all account types
* Validate sufficient funds before processing withdrawals
* Display error messages for invalid transactions (e.g., insufficient funds)
* Record all transactions with timestamp, type, amount, and resulting balance

**Interest Management:**

* Automatically calculate and apply monthly interest:
  + Individual savings: 0.025%
  + Company savings: 0.075%
  + Investment accounts: rate based on amount
  + Checking accounts: no interest
* Record interest payments in transaction history

**User Authentication & Access:**

* Customer login to view account information
* Account balance viewing and transaction history access
* Role-based access (customers vs. tellers)
* Account statement generation showing complete transaction records

**1.2 Non-Functional Requirements**

**Security:**

* Secure user authentication system
* Role-based access control separating customer and teller functions
* Protection of sensitive financial and personal data

**Performance:**

* Handle minimum 10-20 records per database table
* Reasonable response times for basic operations
* Support concurrent user access (basic level)

**Usability:**

* Modern, intuitive GUI design reflecting current technology trends
* Appropriate color schemes (green for accept actions, red for cancel)
* Clear error messaging and user feedback
* Logical navigation and user-friendly interface elements

**Reliability:**

* Accurate financial calculations and balance management
* Consistent transaction processing and data integrity
* Proper error handling for system failures
* Complete audit trail through transaction history

**Maintainability:**

* Object-oriented design demonstrating inheritance, encapsulation, abstraction, and polymorphism
* Clear separation between business logic, data access, and presentation layers
* Modular architecture supporting future enhancements

**Appendix: Interview Record**

**Interview Details:**

* **Interviewer:** Jaden Williams (System Analysts/Designers)
* **Interviewee:** Themba Moeng (Banking Manager/Instructor)
* **Method:** Group mock interview session

**Key Questions Asked and Responses:  
  
Interview with Module Lecturer (Mr. Themba Moeng - Acting as Bank Manager)**

* **Date:** September 2025
* **Duration:** 12 minutes
* **Method:** Teams voice call
* **Role:** Primary business stakeholder defining system requirements

**Key Questions and Responses from Interview:**

**Q: Do customers use the same login system as employees or do they have separate credentials like account number and PIN?**

**Lecturer Response:** "You can have it as one integrated system where basically you'd provide a username and password like any system, similar to the employee side. But if you want to take the approach of the PIN and account number stuff... the approach that I gave - the first one where you just log in with username and password, then it gives you access to all the accounts that you can choose which one you want to interact with. So you have that flexibility."

**Q: How long should user sessions remain active before automatic logout for security?**

**Lecturer Response:** "This one we haven't specified a certain time period, but you being in the IT game, you can suggest. Remember we are simulating a real world situation where you may encounter people that require a system... they will not be fully technical people. Then you being the expert can suggest."

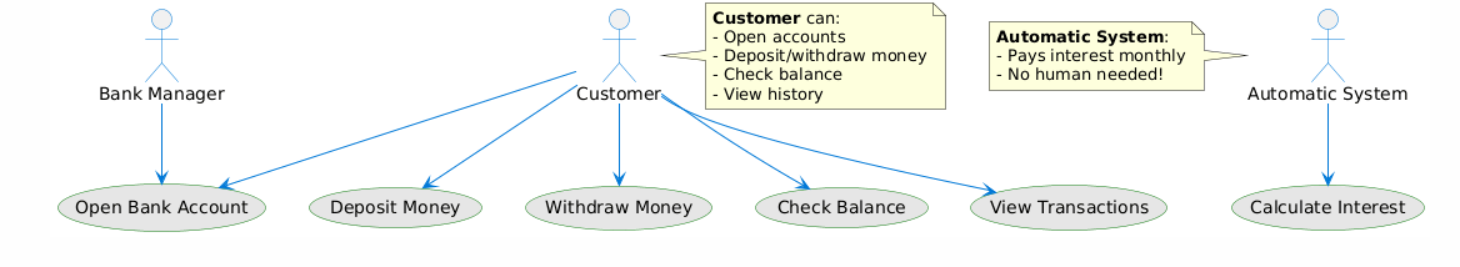
**Q: Should there be specific operations on employee dashboard versus customer dashboard?**

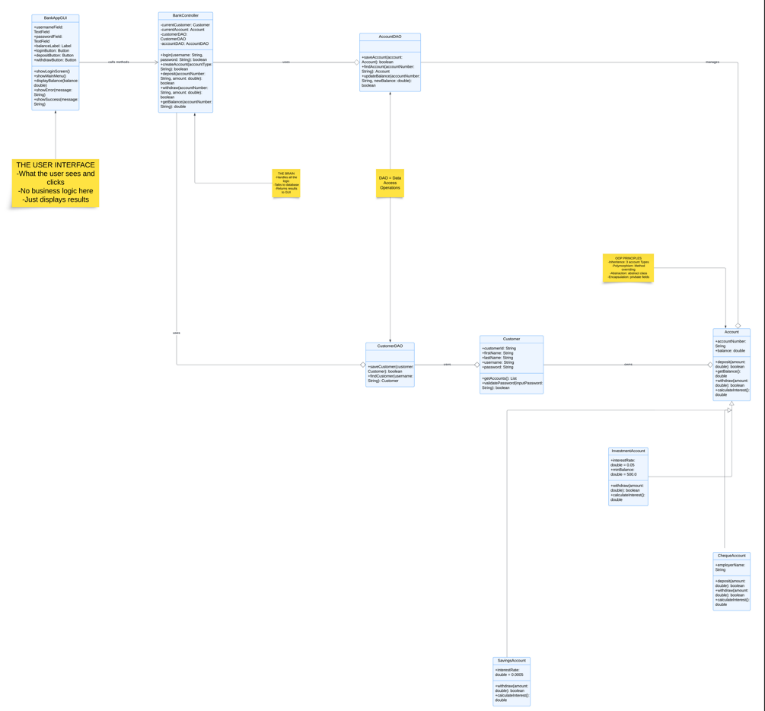
**Lecturer Response:** "Once the employee logs in, the dashboard will enable them to pick operations they can perform, such as customer functions, the ability to add customer details, the ability to update, the ability to search for a specific customer... then you can create an account for them and associate the account with them and those operations that facilitate you to calculate interest on accounts that allow that operation."

**Q: How should we handle transaction failures for system reliability?**

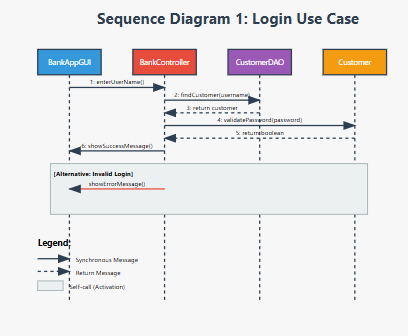
**Lecturer Response:** "If a system failed during a transaction, it means the transaction did not complete. Remember when you're dealing with database operations, it either fails completely or it succeeds completely... If you're trying to withdraw and the withdrawal has not completed, it will be as if the withdrawal has not happened and the same can be said about the deposit. But we can keep a system log saying there was an attempt."

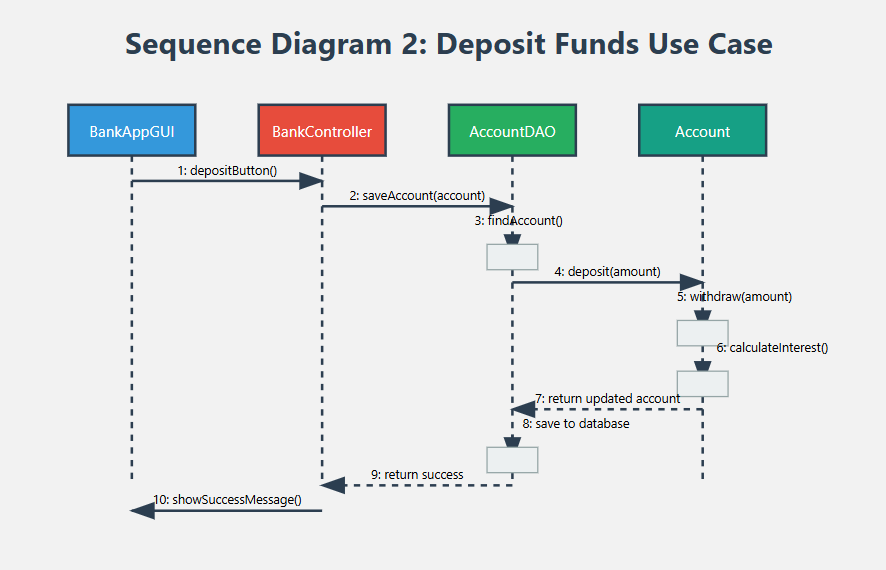
**Q: What about automatic logout timing for security?**

**Lecturer Response:** "This one we haven't specified a certain time period, but you being in the IT game you can suggest... you being the expert can suggest. We can suggest. So it will be just a statement that we can add in terms of that feature."  
  
These series of questions and answers helped narrow down the overall project and guide me towards a better goal/end product. Some of these I may not implement within the code/program itself but we’ll see as time moves along how that may change or not.  
  
**2. Structural UML Modelling**2.1 Use Case:  
2.2 Class Diagram:

  
  
**3. Behavioural UML Modelling**

3.1 Sequence Diagrams:



  
3.2 State Diagram: Account Lifecycle

