**Appendix A – Requirements Elicitation Interview Record**

**Project:** Banking System Application  
**Date:** 18 September 2025  
**Time:** 10:30 AM-12:16 PM  
**Place:** Botswana Accountancy College-**Online Session**  
**Interviewer:** Katso Kgengwenyane (Student Developer)  
**Interviewee (Client):** Mr Kentsenao Baseki (Lecturer)

**Interview Transcript**

**Interviewer:** Good morning, Mr Kentsenao. Thanks for meeting with me today.  
**Client:** Good morning, no problem at all.

**Interviewer:** To start, could you tell me in your own words what you expect this Banking System to do?  
**Client:** Sure. The main thing is that we want a system that can open accounts for customers with the bank, and handle transactions like deposits and withdrawals.

**Interviewer:** Okay, and what types of accounts should the system support?  
**Client:** We work with three types of accounts: Savings, Investment, and Cheque accounts.

**Interviewer:** Right, and are there specific rules for those account types?  
**Client:** Yes. For Savings, customers can only deposit, they can’t withdraw. The system should also calculate a small monthly interest. For Investment, customers need at least BWP 500 to open the account, and they can deposit and withdraw. That one should get a higher monthly interest. For Cheque accounts, we need to capture the customer’s employer details, and they should allow deposits and withdrawals.

**Interviewer:** Understood. About the interest, how often should it be applied, and what are the rates?  
**Client:** It should be automatic. For individual savings accounts, apply 0.025% monthly. For companies, it should be 0.075% monthly.

**Interviewer:** Great. What kind of information do we need to collect when registering a new customer?  
**Client:** Basic details like first name, surname, address, cell number, and in the case of a company account, their company name and address as well.

**Interviewer:** Got it. Should a customer be able to open more than one type of account?  
**Client:** Yes, definitely. Many of our clients will have both a Savings and a Cheque account, for example.

**Interviewer:** And from a security point of view, what’s important for the system?  
**Client:** Security is very important. Customers should not be able to access other people’s accounts. And the system should be reliable — transactions should go through without errors and balances must always be accurate.

**Interviewer:** Do you want customers to be able to see their past transactions?  
**Client:** Yes, they should be able to check their balances and view transaction history.

**Interviewer:** Okay. Finally, are there any future features you might want, even if not right away?  
**Client:** Later, maybe online access so customers can log in remotely. But for now, we just want to get the basics working properly. But the system to relate to current trends.

**Interviewer:** Perfect, thank you for clarifying all of that.

Section: Use Case Diagram Explanation

A Use Case Diagram for the Banking System visually represents the system's functions and its interactions with external entities. It helps to define the scope of the system and shows what it does from the perspective of the users (actors) without getting into the internal details.

1. Actors

The actors are the external entities that interact with our system. We have identified two main actors based on the project's requirements:

Customer: The primary user who interacts with the system for personal banking needs.

Bank Staff: The internal user who is responsible for managing customer information and accounts.

2. Use Cases

"The use cases represent the specific functions provided by the system. We have identified the following use cases:

Register Customer

Deposit Funds

Withdraw Funds

Check Balance

View Transaction History

Authenticate User

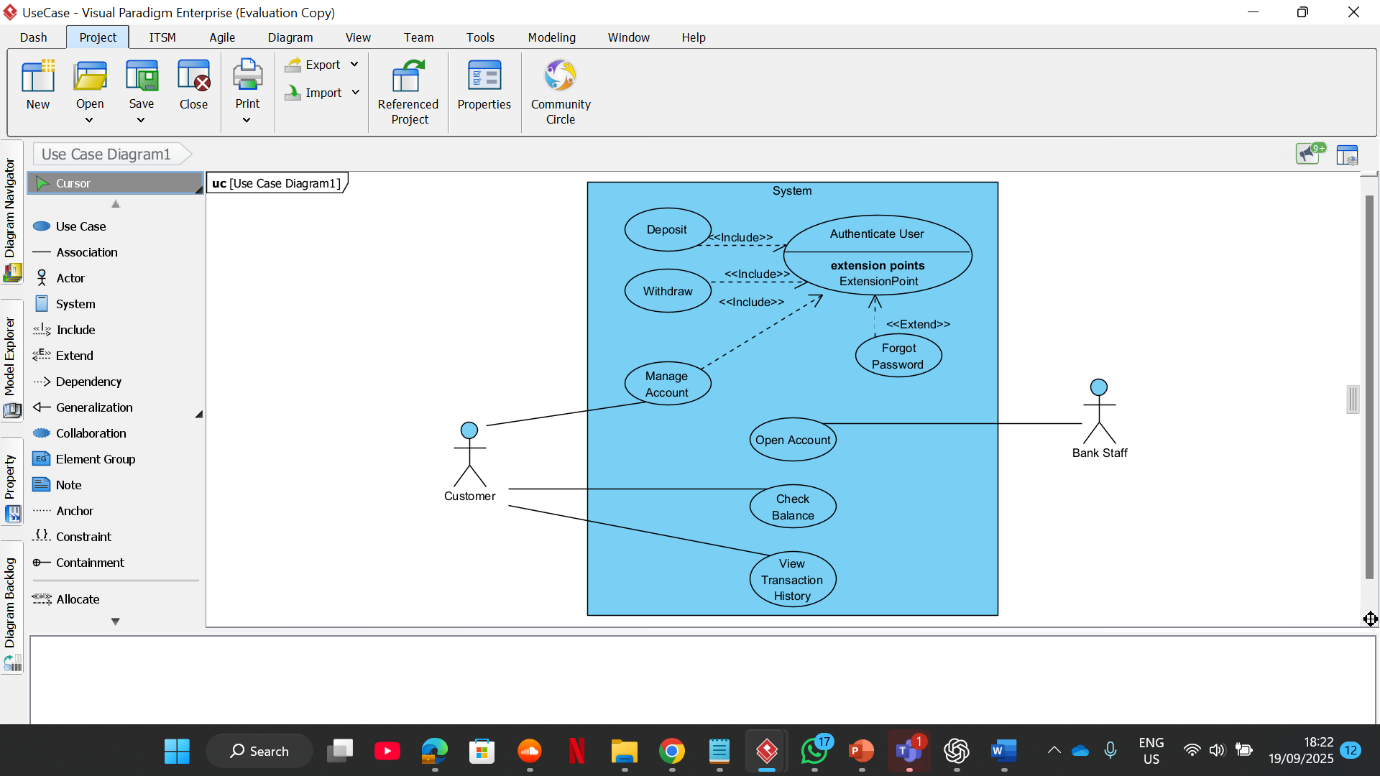
Forgot Password

3. Relationships

We have used two key relationships to show the connections between our use cases:

<<include>>: This relationship is used when one use case requires the functionality of another. For example, Deposit Funds and Withdraw Funds both <<include>> Authenticate User, as a user must be authenticated to perform a transaction.

<<extend>>: This relationship is used to add optional functionality to a use case. For example, the Forgot Password use case <<extends>> Authenticate User because it is an optional flow that only occurs if a user cannot log in normally.



Section: Class Diagram Explanation

The Class Diagram for the Banking System models the static structure of the system. It shows the classes, their attributes (data), their methods (behavior), and the relationships between them. This diagram is crucial for laying the groundwork for our code, as it directly demonstrates key Object-Oriented Programming (OOP) principles.

1. Abstraction

"Abstraction is the process of hiding complex details and showing only the essential features. Our Account class is an abstract class that serves as a general blueprint for all account types. It contains common attributes like accountNumber and balance, but cannot be directly instantiated. Instead, we create concrete classes like Savings Account, Investment Account, and Cheque Account that inherit from it."

2. Inheritance

"Inheritance allows us to reuse code and create a hierarchy of classes. Our diagram shows this with the generalization lines pointing from Savings Account, Investment Account, and Cheque Account to the parent Account class. This means they inherit all the attributes and methods of Account, and we only need to add their unique features."

3. Encapsulation

"Encapsulation is the bundling of data and methods that operate on that data. We have shown this by marking our attributes as private (-) and our methods as public (+). For example, the balance attribute in the Account class is private, so it can only be modified by public methods like +deposit() and +withdraw(), which ensures data integrity."

4. Using an Interface

"An interface is a contract that defines a set of methods that a class must implement. We created an Interest Payable interface with a calculateMonthlyInterest() method. We have shown that the Savings Account and Investment Account classes both implement this interface using a dashed line. This is the correct way to model a shared behavior that is not part of the inheritance hierarchy."

5. Relationships

"The diagram also shows the key associations between classes. We have modeled a relationship between the Customer and Account classes with a multiplicity of 1..\* on the Account side. This correctly shows that a single customer can have one or more accounts, as required by the assignment.

