

PROJECT SUBMISSION

Advanced Software Engineering Project

**Program: CESS**

***Course Code: CSE 232***

***Course Name: Advanced Software Engineering***

***Examination Committee***

**Dr. Gamal A. Ebrahim**

**Ain Shams University**

**Faculty of Engineering**

**Fall 2020 Semester**

Contents

[I. Abstract 1](#_Toc60310679)

[II. Introduction 1](#_Toc60310680)

[A. Purpose 1](#_Toc60310681)

[B. List of Definitions 1](#_Toc60310682)

[C. Scope 1](#_Toc60310683)

[D. List of References 2](#_Toc60310684)

[E. Overview 2](#_Toc60310685)

[III. General Description 2](#_Toc60310686)

[A. Product Perspective 2](#_Toc60310687)

[B. General Capabilities 3](#_Toc60310688)

[C. General Constraints 3](#_Toc60310689)

[D. User Characteristics 3](#_Toc60310690)

[E. Environment Description 3](#_Toc60310691)

[F. Assumptions and Dependencies 4](#_Toc60310692)

[IV. Specific Requirements 4](#_Toc60310693)

[A. Capability Requirements 4](#_Toc60310694)

[B. Constraint Requirements 5](#_Toc60310695)

[V. Illustrations and Diagrams 5](#_Toc60310696)

[A. Use Case Diagram and Narrative Descriptions 5](#_Toc60310697)

[B. Swimlane Activity Diagram of all use cases 5](#_Toc60310698)

[C. Interaction Diagram 5](#_Toc60310699)

[D. Noun Extraction and CRC Cards 5](#_Toc60310700)

[E. Class Diagram 7](#_Toc60310701)

[F. State Diagram 7](#_Toc60310702)

[G. Client-Object Relation Diagram 8](#_Toc60310703)

[H. Component Diagram 8](#_Toc60310704)

[I. DFD (level 1) 8](#_Toc60310705)

[VI. OOAD Methologies 8](#_Toc60310706)

[A. Object-Modeling Technique (OMT) by Rambaugh 1991 8](#_Toc60310707)

[B. Jacobson et al. Methodologies 9](#_Toc60310708)

[VII. Comparative Analysis of the Output of the Adopted Methodologies 11](#_Toc60310709)

[VIII. Architectural Model 11](#_Toc60310710)

[A. Implicit Invocation Architecture 11](#_Toc60310711)

[B. Layered Architecture 12](#_Toc60310712)

[C. Model-View-Controller Architecture Figure 12 13](#_Toc60310713)

[IX. Testing 14](#_Toc60310714)

[X. Timeplan 16](#_Toc60310715)

[XI. User Guide 18](#_Toc60310716)

[A. Login Page 18](#_Toc60310717)

[B. Client Account 18](#_Toc60310718)

[C. Employee 19](#_Toc60310719)

[D. Employee Loan 19](#_Toc60310720)

[E. Employee Meeting 19](#_Toc60310721)

[F. Loan History 19](#_Toc60310722)

[G. Loan Request 19](#_Toc60310723)

[H. Meeting Request 20](#_Toc60310724)

[I. Set Bill 20](#_Toc60310725)

[J. Transaction History 20](#_Toc60310726)

[K. Transfer Money 20](#_Toc60310727)

[L. Money Withdrawal 20](#_Toc60310728)

[XII. Figures and Tables 21](#_Toc60310729)

[A. Diagrams 21](#_Toc60310730)

[B. GUI 30](#_Toc60310731)

[C.Tables 36](#_Toc60310732)

Table of Figures

Figure 1: Use Case Diagram 21

Figure 2: Swimlane Activity Diagram 22

Figure 3: Sequence Diagram 23

Figure 4: CRC Diagram 23

Figure 5: Class Diagram 24

Figure 6: State Diagram 25

Figure 7: Component Diagram 26

Figure 8: DFD Diagram 27

Figure 9: OOSE Diagram 28

Figure 10: Client-Object Diagram 28

Figure 11: MVC Architecture Diagram 29

Figure 12:The Gantt Chart 29

Figure 13: The Login Screen 30

Figure 14: The Client Landing Screen 30

Figure 15: The Employee Landing Screen 31

Figure 16: The Employee Loan Screen 31

Figure 17: The Employee Meeting Screen 32

Figure 18: The Loan History Screen 32

Figure 19: The Loan Request Screen 33

Figure 20: The Meeting Request Screen 33

Figure 21 : The Set Bill Screen 34

Figure 22:The Client Transaction History Screen 34

Figure 23:The Transfer Screen 35

Figure 24: The Withdraw Screen 35

Table of Tables

Table 1 : Login Use Case Description 37

Table 2: Check Balance Use Case Description 37

Table 3: Request Loan Use Case Description 38

Table 4: Money Withdrawal Use Case Description 39

Table 5: Transfer Funds Use Case Diagram 40

Table 6: Make Payment Use Case Description 41

# Abstract

This project is for a banking system web-based application that manages the users’ account through different functionalities as described later.

# Introduction

## Purpose

The purpose of this document is detailed documentation of the project/system along with a description of how to use the system. The intended readership of this document is dissected into two parts as parts from 2 to 16 are intended for the project managers and different software engineers working on the project. While part 17 (user guide) is intended for the users after the delivery of this product.

## List of Definitions

* CRC: Class Responsibility Collaboration
* DFD: Data Flow Diagram
* OOAD: Object Oriented Analysis and Design
* OMT: Object Modeling Technique
* MVC: Model-View-Controller
* GUI: Graphical User Interface

## Scope

A banking system web-based application that allows existing clients of the bank to access their account information, balance, transactions, can conduct different transactions (withdrawal, transfers), make inquiries, request loans, and be able to request meetings. Also, the bank employees have different functionalities such as responding to such inquiries and meetings, accessing clients’ accounts for different purposes, and accepting loans or checking transaction history. It follows the Central bank rules, the international commerce rules and keeps check of all the details and information of the user.

## List of References

1. Pressman, R. S., & Maxim, B. R. (2015). Software engineering: a practitioners approach (Eighth). New York, NY: McGraw-Hill Education.
2. Schach, S. R. (2009). Object-Oriented Software Engineering. Beijing: China Machine Press.
3. Sommerville, I. (2016). Software engineering (Tenth). Boston: Pearson.
4. Rumbaugh, James, et al. The Unified Modeling Language Reference Manual (Second). Addison-Wesley, 2010.
5. Kellner, Marc I., and Watts S. Humphrey. Software Process Modeling: Principles of Entity Process Models. Carnegie Mellon University, Software Engineering Institute, 1989.

## Overview

The document includes:

* The general description of the product: Description of the product design and process.
* Its specific requirements - The capability and constraint requirements.
* The diagrams needed for the development of the product.
* The used OOAD methodologies and a comparative study between them.
* The architectural model of the system with the justification of the design decision.
* Testing: The detailed testing process
* Time plan: A time plan of all the phases of the project
* User guide

# General Description

## Product Perspective

The product is a web-based application software for a bank system which will help to reduce the paper filling and handwritten work which will then reduce the environmental impact and accelerate services of the banks.

## General Capabilities

* Clients will have a password protected account accessed by them and by the bank teller when the client goes to the bank personally.
* Clients will be able to access their account balance and check it, and the bank will send the right information to them.
* Clients can send a request for a money loan which will be send to the bank manager and then the bank manager will have the option to review the request and either reject or accept it.
* Payments can be made by the clients if they have the sufficient funds and they can choose either to pay from savings or checking.
* Clients will be able to transfer money from their account to other accounts and vice versa, and the bank will follow the transferring process.
* Clients can withdraw money from their accounts if they have the sufficient funds for that.

## General Constraints

* The program should require less than 0.2 GB of RAM to run.
* The website should support mobile application for Android and IOS.

## User Characteristics

* Client: login to his account, check balance, withdraw money, transfer money, make money payments, request loan.
* Bank teller: when client goes to bank, the bank teller can do payments, transfer money, request loans, and check balance for the client.
* Bank manager: review the requests for loans that have been submitted by the clients and choose whether to accept them or not.

## Environment Description

* The software is a web-based application.
* The programming language used to make this software is Java.
* Client’s information is stored in a password protected database (Microsoft Azure or MySQL) with Java code extracting and editing the protected information.

## Assumptions and Dependencies

* The website should work on different browsers such as chrome and safari.
* The website should have the ability to adjust to different formats such as on mobile or tablet.
* The website should support IOS and Android use (preferably a supported application).
* The application should be no more than 60 MB in size.
* The password should follow the NIST standards that state that a password should be at least six characters and could contain numbers only.
* The website should have high security and privacy to protect credit card details.
* It should also have online payment support.

# Specific Requirements

## Capability Requirements

* Clients will be able to make account protected with a password and access them anytime.
* Clients will be able to check their balance.
* Clients will be able to withdraw specific amount of money.
* Clients will be able to transfer funds.
* Clients will be able to make payments either from checking or from savings.
* Clients will be able to request loan.
* Bank manager will be able to review client’s loan requests.
* Bank manager will be able to either accept or decline client’s loan requests.
* Bank teller will be able to access client’s account when client approach them in the bank.
* Bank teller will be able to look into client’s account balance on behalf of client.
* Bank teller will be able to transfer money on behalf of client.
* Bank teller will be able to make payments on behalf of client.
* Bank teller will be able to request loans on behalf of client.

## Constraint Requirements

* Bank tellers will not be able to see client’s information without their permission.
* Bank manager can only access part of client’s information when they request a loan.
* Client will not be able to see other client’s information.
* The maximum data stored for each client is provided by the database manager systems.
* Not updated Information should have a limit while staying on the database.
* The password should follow the NIST standards that state that a password should be at least six characters and could contain numbers only.

# Illustrations and Diagrams

## Use Case Diagram and Narrative Descriptions

The goal of the use case diagram is to show to the stakeholders the system from the perspective of the end-users, or actors. It represents the different functionalities of the software indicated in Figure 1. Also, a description of all the uses cases is added to further explain the use case (view Table 1 through Table 6).

## Swimlane Activity Diagram of all use cases

The activity diagram, as demonstrated in Figure 2, explains how the system flows from a behavior perspective, from beginning to the end, while indicating the user responsible for each action.

## Interaction Diagram

The sequence diagram shows how the actions of the system flow in addition to the software interactions with the real world, which is visible in Figure 3.

## Noun Extraction and CRC Cards

1. *General description of the System:*

The banking system allows its clients to have accounts that can be checked, where banking transactions are available such as withdrawal, money transfer and balance inquiry. Clients can also request loans, but they must get approval, and the system will track all loans and payments. Clients can also associate their account with monthly payments or bills that can be automatically deducted or deposited in their account. This software system can also be accessed by bank tellers to be able to make banking transactions on behalf of the clients who visit the different branches of the bank. The banking system is accessed through a username and a password associated with a card number; hence, the system must be able to identify the user, either client or bank employee.

1. *Functional Requirements:*

* Login
* Check the balance.
* Ask for money withdrawal.
* Make money transfer.
* Request/approve loans.
* Track loans and payments.
* Associate bank account with monthly payments/bills.

1. *Non-Functional Requirements:*

* The website should work on different browsers such as chrome and safari.
* The website should have the ability to adjust to different formats such as on mobile or tablet.
* The website should support IOS and Android use (preferably a supported application).
* The application should be no more than 60 MB in size.
* The website should be supported for both IOS and Android.
* The website should be written in Java, HTML, CSS, JavaScript, or Python.
* The password should follow the NIST standards that state that a password should be at least six characters and could contain numbers only.
* The website should have high security and privacy to protect credit card details.
* It should also have online payment support.

1. *Noun Extraction Process:*

* Nouns from the description: System, Client, Account, Transaction, Withdrawal, Money Transfer, Balance Inquiry, Loans, Payments, Bills, Bank tellers, Branches, Bank, Username, Password, Card Number (ID), User, Bank Employee
* Irrelevant nouns: System, Branches, Bank, Bank tellers, Balance Inquiry, Loans, Payments, Bills
* Nouns selected: Client, Account, User, Transactions, Employee, Username, Password, Withdrawal, Money transfer, Card number, Bills, Loans.
* Classes: Client, Employee, Bill, Transaction, Loan

1. *CRC Cards:*

The class responsibility collaboration cards (CRC) are used to clearly digest the object-oriented mechanisms into more common concepts for the sake of the users to understand, implemented in Figure 4.

## Class Diagram

The class diagram, one of the main UML diagrams, depicts the organization of the code which is to be delivered by the programmers. This is done by abstraction and encapsulation of individual entities, where each entity of the real world becomes a class with its corresponding attributes and methods, as Figure 5 shows.

## State Diagram

The state diagram, described visually in Figure 6, is used to portray a software system through its dynamic behavior by the concept of states, where transfer between states representing the flow of the system caused by a specific action.

## Client-Object Relation Diagram

The Client-Object Relation diagram, delivered in Figure 11, displays the relation between the different parts of the software and the clients interacting with it, it also indicates which objects should be in the main part of the software and which parts should be on their own. The classes Client, Bill and Transaction should be instantiated in the main().

## Component Diagram

The component diagram explains the software in large from a deeper viewpoint, where different modules of the system communicate through provided and required interfaces. This diagram, as Figure 7 portrays, is a way to represent different parts of the software that will be handled by different teams of programmers.

## DFD (level 1)

The DFD, which is short for the data flow diagram, is an illustration to explain profoundly how data and information can immerse through the system and travel between different parts of the system. This diagram aids in the conversion to a layered architecture for the system, which is depicted in Figure 8.

# OOAD Methologies

## Object-Modeling Technique (OMT) by Rambaugh 1991

The object-modeling methodology is a technique for software creation and design. Invented by Rambaugh, Blaha, Premerlani, Eddy and Lorensen, this technique was used as a way to develop systems based on object-oriented ideology and to support object-oriented programming. OMT illustrates the static structure of the software system.

OMT was developed for:

* Test entities before their construction or simulation.
* Improve communication with stakeholders.
* Help with visualization.
* Reduce complexity.

OMT is divided into phases, which can be performed iteratively:

1. Analysis: A model of real-world problems is prepared at this stage. Analysis starts with setting a goal i.e., finding the problem statement. The problem statement is then divided into object, dynamic and functional model.
2. System design: It decides all system architecture and data storage. Prominent level architecture of the system is designed during this phase that decomposes the system into a set of sub-systems and/or packages Figure 11.
3. Object design: This phase creates a design document, that contains the objects in detail along with the dynamic and functional models. It focuses on the separation of objects into different classes with their attributes and necessary operations and functions.
4. Implementation: During this phase, the design is translated into the software by development.
5. OMT separates analysis phase into three distinct parts:

* Object model: presented by the object model shown in Figure 9 and the data dictionary.
* Dynamic model: presented by the state diagrams and event flow diagrams. It is displayed in Figure 2, Figure 3, and Figure 6
* Functional model: Functional Model in OMT describes all the processes and their actions using data flow diagram (DFD). It focuses on the flow of the data through the system, where data is stored and how it is processed by the different processes. This is shown in Figure 8.

## Jacobson et al. Methodologies

One of the many OOAD methodologies that established the UML to the software engineering world.

It was developed by Ivar Jacobson in 1992 and called Object Oriented Software Engineering (OOSE) shown in Figure 10 where it covers the entire life cycle of the software system using four main steps:

1. Requirements: the developers review the system description and extract the divergent functions and requirements needed to start the analysis process. This is discussed in ‎Section IV-D above.
2. Analysis: the developers analyze the extracted requirements to decided what can be removed and what else is missing to refine the design process. This is discussed in ‎Section IV-D above.
3. Design: the developers start the designing of the system and the outer view of every function and requirements that will be implemented. This is discussed in ‎Section IV-D above.
4. Implementation: where the developers start implementing the approved design.
5. Testing: where the implemented functionalities are tested to make sure they are according to standards and achieve the desired output and function. This is discussed in ‎Section IX below

The different diagrams needed in this figure are similar to the other methodologies such as interaction diagram represented by Figure 3 and state transition diagram represented by Figure 6.

It also stresses and focuses on traceability between the various phases. This method is better presented by use case diagram shown in Figure 1 that is considered its main diagram and specialty that shows the scenarios needed to understand the system requirements showing the interactions between users and the system while capturing the goal of the user and the various responsibilities of the system.

# Comparative Analysis of the Output of the Adopted Methodologies

From the use of both methodologies, it was concluded that the output is like each other, but they have some different diagrams and analysis steps.

OMT consists of four steps as mentioned above analysis, system design, object design, and implementation. These steps result in the production of class, state, activity, data flow and sequence diagrams also the selection of a certain architecture style and the reason why it is used is stated.

Jacobson et al. Methodology consists of five steps as mentioned above requirements, analysis, design, and implementation. This methodology focuses on the entire life cycle of the system as it bisects the system from the beginning of the development life cycle as it extracts all the requirements and functionalities then they are analyzed in the next phase then the design of the system starts according to the analysis, implementation of the approved design is the next phase. Testing is the next phase as to make sure everything is in order as shown in ‎IX below. These steps result in the production of CRC cards, class, sequence, and state diagrams.

# Architectural Model

## Implicit Invocation Architecture

1. *Problem:*

The GUI is composed of loose components that function semi-independently. Buttons, forms, and scripts must be responsive and reactive to user interactions.

1. *Context:*

An event handling programming language is needed to interact with the user.

1. *Solution:*

We will use JavaFX as our programming language as it is event driven. This is shown by implementing the JavaFX event handlers for any buttons, forms, combo boxes and any other user interactive object. JavaFX allows us to make certain components listen for specified signals and, depending on the signal received, activate an event. The architecture chosen to aid in the development of the GUI is the Implicit Invocation.

## Layered Architecture

1. *Problem:*

The program should have some organizational structure to protect data integrity and have a more organized complex program. It should also support component reuse and enhancement.

1. *Context:*

Often, software programs get larger as more time passes. These programs, that are composed of various components, reach a level where inter-process communication becomes unmanageable. This causes problems in unit and integration testing between the modules; therefore, we should look to keep component communication well organized and to a minimum.

This is done by dividing the program into layers where each layer can only interact with the 2 layers surrounding it (above or below it). This allows our system to be simpler and have a better, cleaner implementation. These layers can also be tested individually as they should be able to be used as standalone layers. The layers communicate through the implementation of an interface to avoid revealing sensitive data to upper layers, simplify the communication process, and make the layer more portable.

1. *Solution:*

This layered architecture will be implemented as follows:

* The lowest layer will contain the database manager as it has the most sensitive information, so it needs to be the furthest from any user interaction to avoid unwanted data manipulation. The database manager is responsible for fetching data from the database and updating the database with any data updates.
* The layer that communicates with the database manager layer is the Class layer where only the relevant data for the processing is extracted from the database into its specified classes to be processed or updated.
* The Transaction and Account Manager layer communicates with the Class layer to process the data and do execute the processing that is required on it. This layer is responsible for processing validated user input and updating values inside classes in the Class layer.
* The API layer communicates with the Transaction and Account Manager layer to send validated user input for processing. This layer is responsible for validating all user input.

## Model-View-Controller Architecture Figure 12

1. *Problem:*

The GUI should become more independent and flexible from the rest of the software.

1. *Context:*

The architecture should allow the user to communicate with the data in applications and modify it, in turn, changing the set of data to a new up to date one. For the sake of modularity, the data should be divided into modules by the chosen architecture. Also, it is best if data manipulation and viewing are presented to the user as a type of interaction; therefore, allowing us to expand the program easily by changing the correct subsystem.

Ideally, the program focuses on the data itself and any changes surrounding it. Several users can interact with the same set of data simultaneously, causing problems like those in Repository Style Architectures, such as those associated with handling conflicting data changes. More interestingly, a single client might like to have several different views with some connection between them.

1. *Solution:*

We implemented the MVC architecture as follows:

* The View is all the GUI components of the system developed in the Implicit Invocation architecture style. This is where all components of the GUI reside, and all user input is handled there. After handling input, the data is sent to the controller for further validation and computation.
* The Model is the system’s database where all system’s data is stored. This data includes all the details about each client’s bank account, and all employee records. This data is sent to the controller model to be processed and edited depending on the user input. This data is also sent to the view for the GUI components to display any changes or the original data to the user.
* The Controller is where most of the backend processing occurs. It is where the system computes any transactions needed, validates user input, and updates data inside the database.

# Testing

While composing the software at hand, testing along the way is a particularly important matter to ensure that different components are working, and that the system can compute all its functionalities without any problems. Software testing is a method that evaluates each system requirement by checking if any problems arise, while verifying the components under the test. Once a problem occurs, it should be solved on the spot, this method highlights the advantages of early testing, which leads to a nearly bug free software product, thoroughly tested, ready for delivery. A test product ensures high reliability, performance, and security; hence, going through software testing leads to lower costs, best quality, and customer satisfaction. Considering the software system implemented, the banking software system was divided into five different components complementary to one another. The components are Data Manager, Graphical User Interface, Transaction Manager, Account Information Formatter, and Front-End Processing. (in Figure 7)

The software system at hand must be tested partially first, by applying unit testing strategy. Once one of the components is ready, unit testing is done to check that a design meets the specifications and behaves as expected. These tests are usually done automatically and run via numerous applications, ensuring each unit fulfils its duties. Modul testing is added to be sure of the component’s performance, with the help of two modules: the stub and the driver. Both modules ease the testing process as the stub supplies any missing method usually provided by another component. This method is replaced with a more simplified version of itself, holding the same signature but more compact behavior. As for the driver, this module implements the call for the component under the test. In case of facing any problem, we trace the code and check what causes the problem to be able to deal with the errors. This is done repeatedly till each component is bug free. We performed unit testing successfully on Account Information Formatter and Transaction Manager.

Once the unit testing has been completed, we move on to the remaining modules through integration testing. Integration testing is a method where one component is tested individually, then, logically related components are tested together. Here, we applied the incremental testing method as it facilitates finding and repairing any errors found throughout the way. This method also reduces the time lost and the cost needed in creating stubs and drivers for each module as the modules tested become stubs and drivers to the components under the test. Incremental testing also allows a view of how a part of the system, made up of two or more components, can work interactively. We performed integration testing on Front-End Processing, together with Transaction Manager and Account Information Formatter, to uncover any defects in the Front-End Processing component, or in general. Moreover, we added the component Data Manager, to check for any bugs while ensuring it is integrated well with the rest of the components. This made us ensure that these four components not only work seamlessly individually, but that they work together without any problems.

Following the incremental integration testing, we developed the last touches in the last component, which is the Graphical User Interface, and started system testing. This type of testing checks systematically if the entire system performs its required functionalities or not. It is a higher level than the previous tests as it looks at the software system from a large overview. It validates that our banking system meets the specifications. We performed this level of testing to ensure that the Graphical User Interface was able to integrate peacefully with the rest of the system without any problems arising from the incorporation. Once we fixed all errors and ensured the stability of the system, we were ready for the next and ultimate step of testing. Nonetheless, while adding a new component and testing the system for faults, we also used sanity testing to ensure that any error resolution or changes in code does not affect the functionalities of the system or the past components that were already tested.

Finally, once the software system was completed and ready, to verify stability and reliability, we ran some stress tests with the help of the staff and volunteers, to ensure that the system can work under extreme conditions. Afterwards, we went through acceptance testing through both its phases (alpha and beta). This part of testing checks customer satisfaction by evaluating the system in terms of the requirements and specifications that were highlighted and documented in the contract. The system is tested for acceptability to inspect if it is ready for delivery. This last branch of testing was completed successfully, and the customers’ needs were met; hence, the software system was accepted.

# Timeplan

Gantt chart (displayed in Figure 11) is a genre of bar chart that displays a project schedule. Our time plan is divided into 4 phases which are Documentation, Design, Creating the program, Testing, and finalizing.

First in the documentation phase, which contains three tasks:

* Interviewing the clients, where we meet our clients and gather the information and requirements that our client’s needs for the software project.
* Then we write down the general description of the project using the information we gathered from the clients.
* Then this general description is turned into formal requirements so we can use it to build the project.

Second in the design phase, the team starts Designing all the diagrams required to display and show all the ideas for the program using the requirements gathered in the documentation phase.

The design phase has three tasks which are

* UML design phase 1 which includes creating the Use Case diagram with all its narrative descriptions, Swimlane diagrams for all use cases, and State diagram for the program.
* UML design phase 2 which includes Sequence diagram, and OOAD steps. The OOAD steps will start with noun extraction, then CRC Cards, and finally creating Class diagram.
* Architectural Design which includes creating a component diagram and then turning it into an Architectural Style which we chose to be MVC.

Third in the creating the program phase, it was time to step from idea creation into idea realization. We started to work on the actual program we were assigned to create. All three tasks were worked on in parallel.

* Creating the model service which contains all the code classes with the attributes and methods that will work on the backend of the code.
* Creating the view which is the GUI Design with all the .fxml files that will contain all the screens and such.
* Creating the controller which connects the model with the GUI (they are the code classes corresponding to the .fxml classes) also created.

Finally, in the testing and finalizing phase, it includes four tasks worked on in chronological order and one task which is finalizing the documentation being worked on in parallel.

* Unit testing included testing the individual classes alone.
* Integration testing tested which tests the problems arising from the different modules interacting together (in this case the Model View and Controller shown in Figure 12).
* Alpha and beta testing which tested the system as a whole and gets it ready for the public.
* Finalizing all the papers including the user guide and all the technicalities for the program was finished.

# User Guide

## Login Page

As the user opens the application, the login page is the first page to appear. The user must enter a valid username and password that will be verified. If the username and password are incorrect, the user cannot get access to their account. Otherwise, the application allows the user to get access to different pages such as Account, Meeting, Loan and Transactions (Figure 13).

## Client Account

Once the user logs in, their Account Name, Number, Type, and current balance appear in the empty fields below. Form here, The Client Can access the tabs on the top of the screen to Request meetings, loans or see their transaction history (Figure 14).

## Employee

The bank employee can also have access to the application. Through their account, an employee can access a client’s name, view the account type, and their current balance. An employee can also create a new bank account by adding the client’s name, specifying the account type and the balance that they deposited into that account with their visit to the bank’s branch (Figure 15).

## Employee Loan

Here the Employee can view all loan requests and they can change the loan status from pending to either approved or declined. To approve or decline a loan request, the employee must first select the desired loan request appearing inside the table and then click on either approve or decline (Figure 16).

## Employee Meeting

Here the Employee can view all meeting requests and they can change the meeting status from pending to either approved or declined. To approve or decline a meeting request, the employee must first select the desired meeting request appearing inside the table and then click on either approve or decline (Figure 17).

## Loan History

This screen appears for all Clients under the Loan tab. This screen shows details about the loan at hand such as the loan date, the loan reason specified by the user previously, and the loan status. If the account has declined, approved, or pending loans, they appear here automatically (Figure 18).

## Loan Request

To request a loan, the client must click on the “Loan” tab above. After that, another tab will appear containing either “Request Loan” or “Request History/Status”. The user must click on “Request Loan”. The form will contain a loan reason which needs to be filled out by the client. After the Client is done filling out the form, the “Request Loan” button should be pressed (Figure 19).

## Meeting Request

To request a meeting with the Employee, the client should press the “Meeting” tab that appears on top of the screen. Then, the client should choose the desired date from the date box on the right and write their reason for wanting to have the meeting (Figure 20).

## Set Bill

To set a monthly bill, the client must click on the “Transactions” tab on the top of the screen. Then, there will appear a sub menu where the client should select the “Set Bill” tab. There, the client should type inside the form the required bill name and amount that should be deducted monthly. After entering the required information, the client presses the “Set New Bill” button, and the bill is created (Figure 21).

## Transaction History

This tab helps the user to keep track of all their previous transactions. If a transaction was completed successfully, the user can view the transaction name, date, specific time, and the transaction amount. This tab will be updated automatically with each new transaction (Figure 22).

## Transfer Money

A bank’s client can make a transaction from their valid account such as money transfer. The user should first choose the tab labeled “Transactions” then press on “Transfer”. Here, the user will need to specify the amount of money to transfer, the account number to which the money should be transferred, and the currency. This transaction is only valid if the user’s current balance is not less than the amount desired to be transferred (Figure 23).

## Money Withdrawal

A bank’s client can make a transaction from their valid account such as money withdrawal which is the act of removing funds from a bank account. The user should first choose the tab labeled “Transactions” then press on “Withdraw”. Afterwards, the user must then specify the amount that they desire to withdraw and the currency they need. The money withdrawal will only be successful if and only if the client’s current balance is greater than or equals the amount to be withdrawn (Figure 24).

# Figures and Tables

## Diagram Description automatically generatedDiagrams

Figure 1: Use Case Diagram

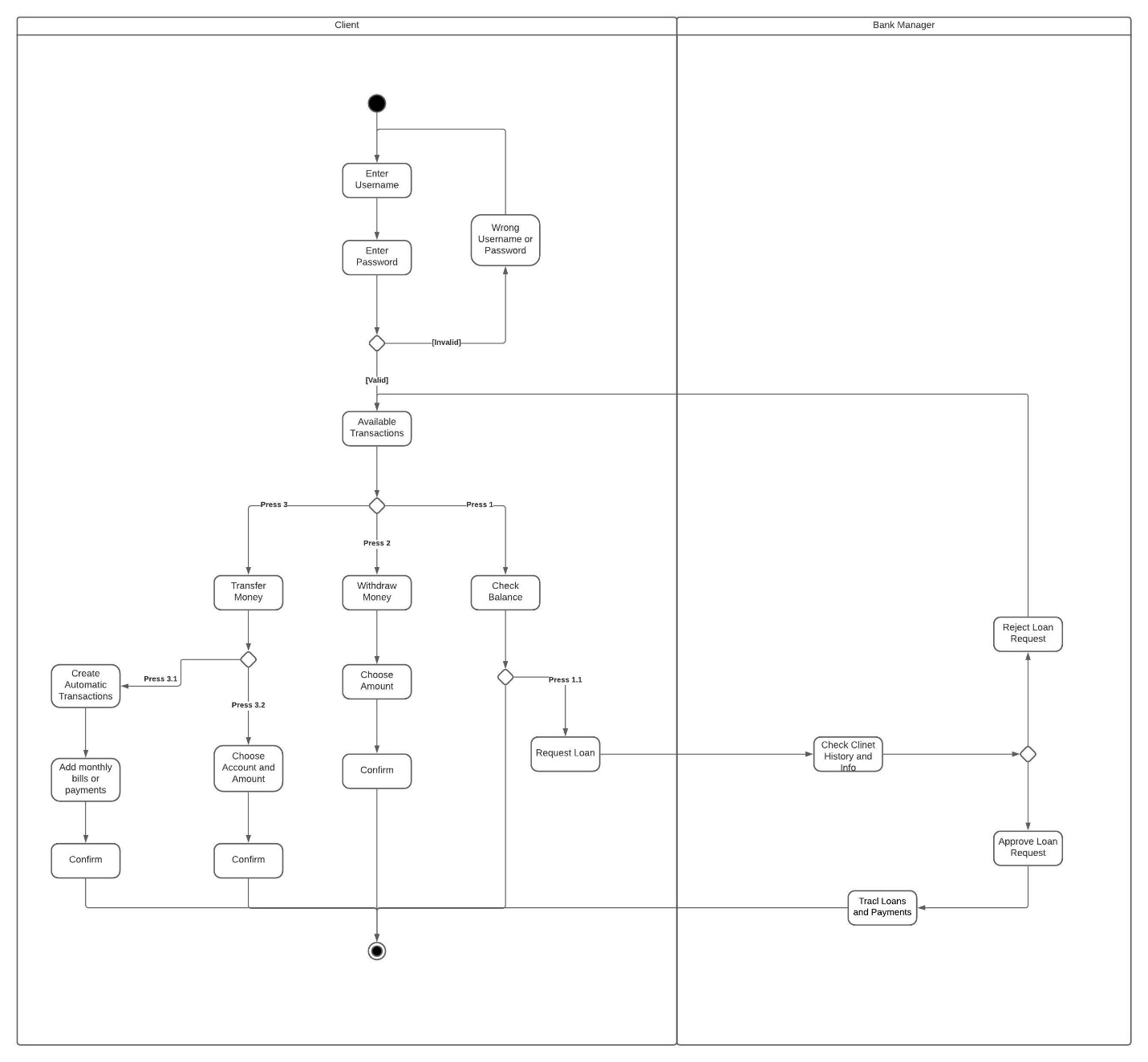


Figure 2: Swimlane Activity Diagram

Diagram

Description automatically generatedDiagram

Description automatically generated

Figure 3: Sequence Diagram

Figure 4: CRC Diagram

Diagram

Description automatically generated

Figure 5: Class Diagram

Diagram, schematic

Description automatically generated

Figure 6: State Diagram

Diagram

Description automatically generated

Figure 7: Component Diagram

Diagram

Description automatically generated

Figure 8:DFD Diagram

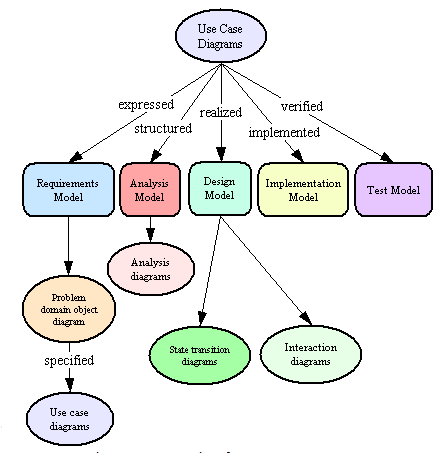


Figure 9: OOSE Diagram

Diagram

Description automatically generated

Figure 10: Client-Object Diagram

Diagram, engineering drawing

Description automatically generated

Figure 11: MVC Architecture Diagram

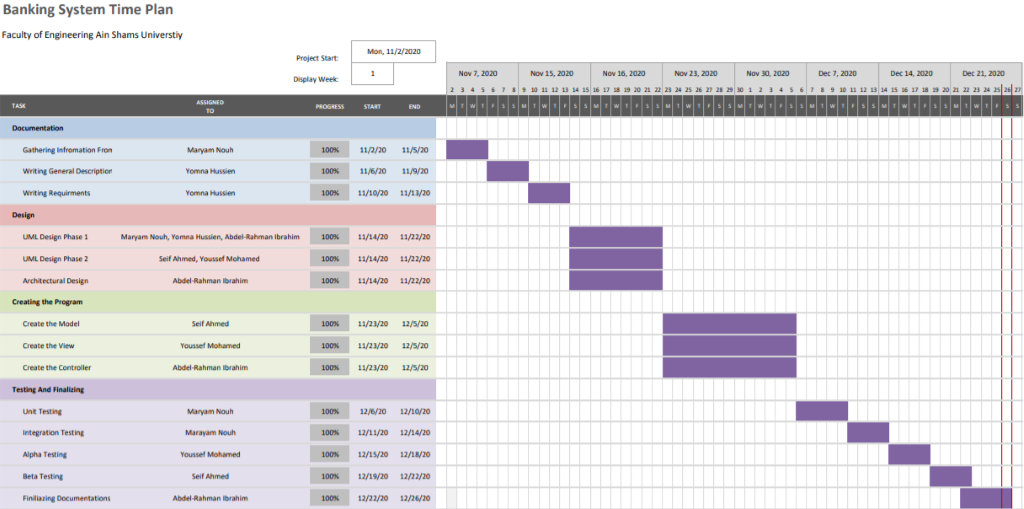
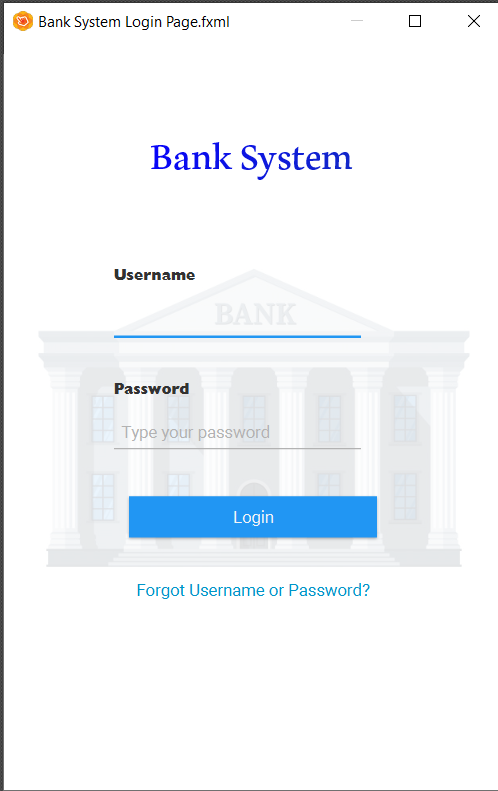


Figure 12:The Gantt Chart

## GUI



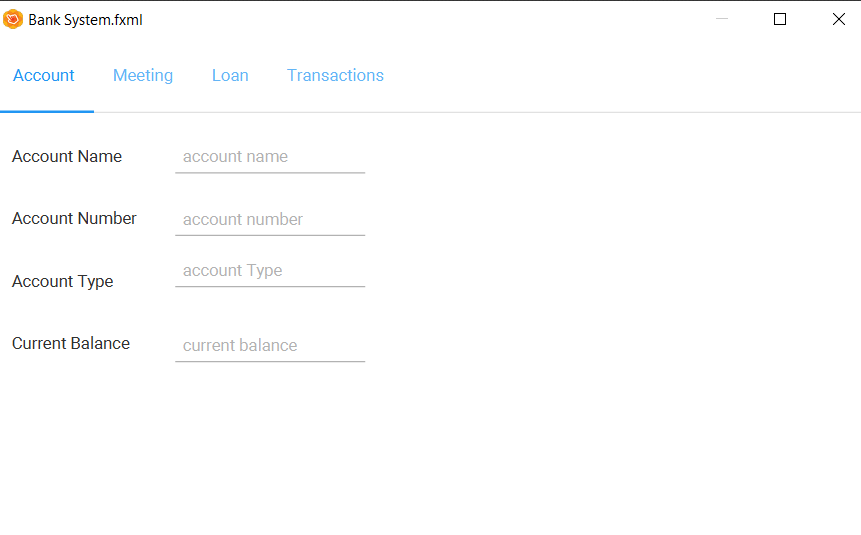


Figure 13: The Login Screen

Figure 14: The Client Landing Screen

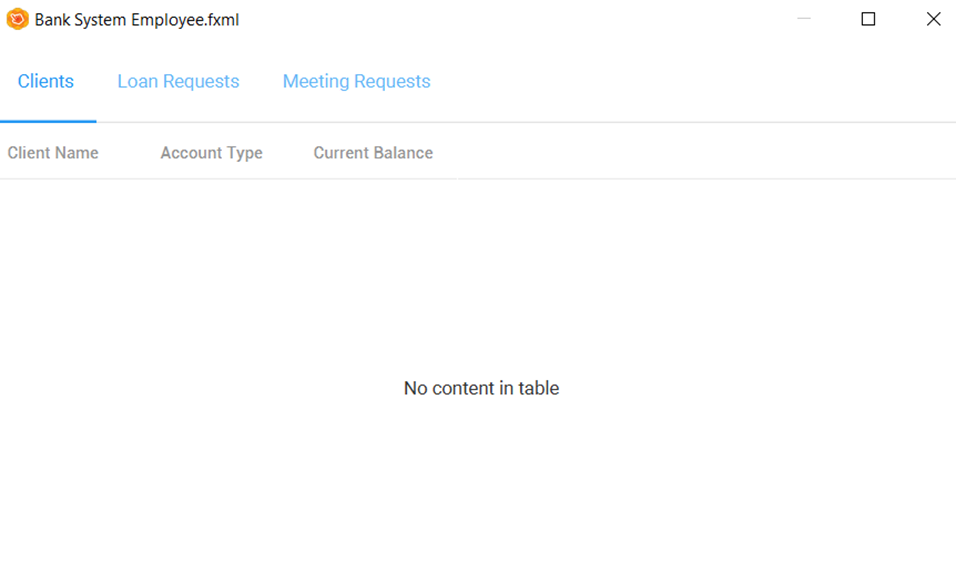
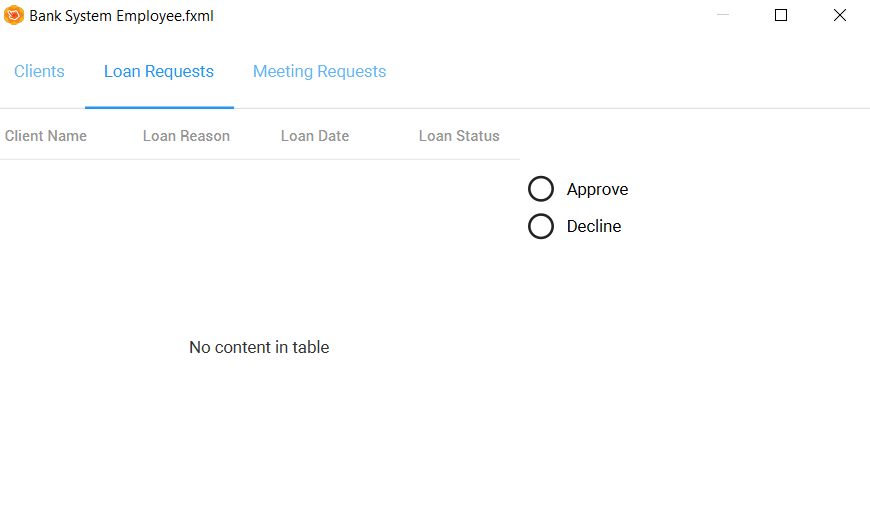


Figure 15: The Employee Landing Screen

Figure 16: The Employee Loan Screen

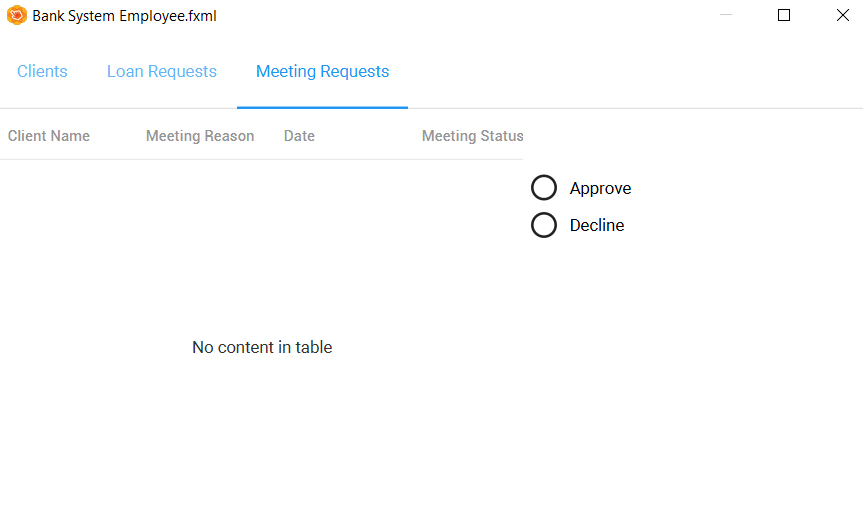
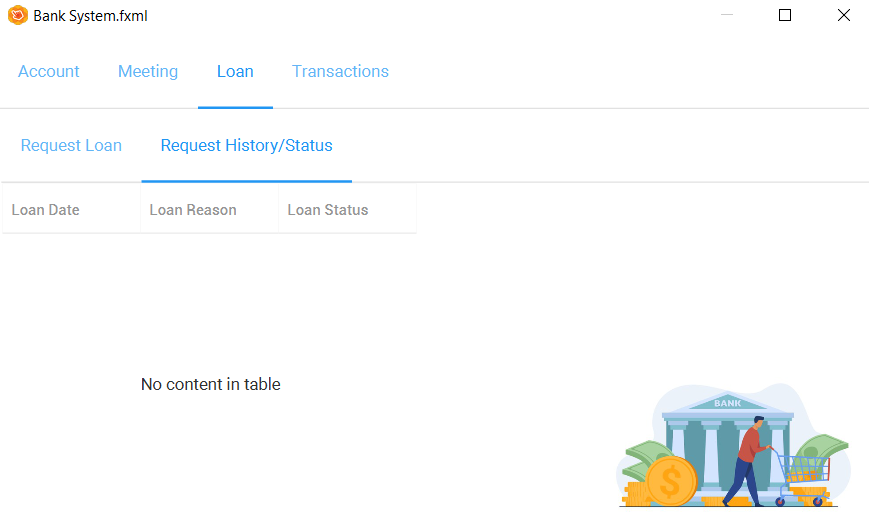


Figure 17: The Employee Meeting Screen

Figure 18: The Loan History Screen

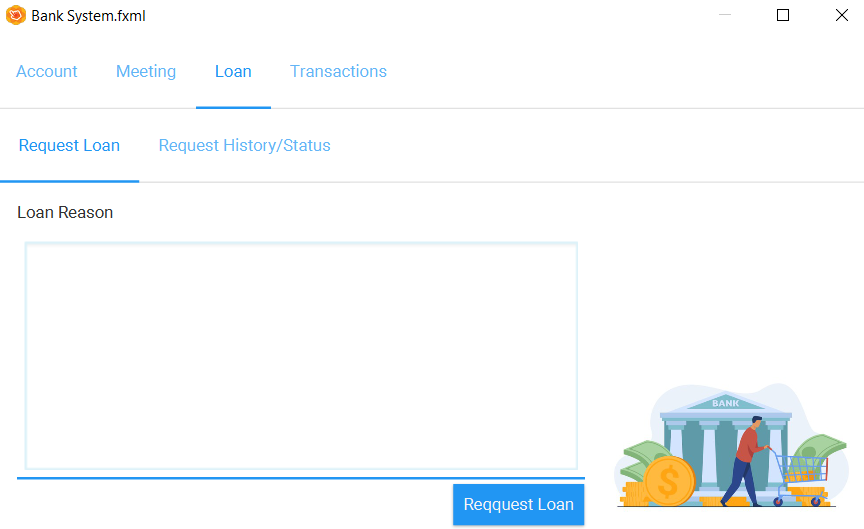
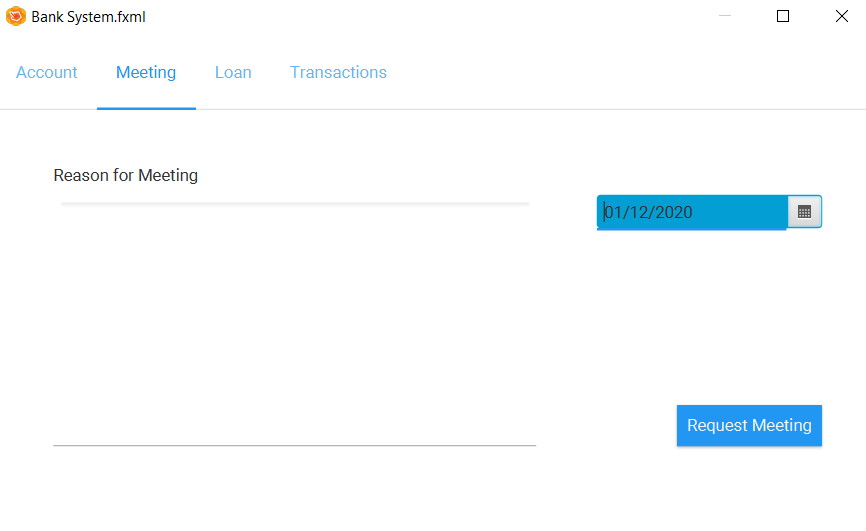


Figure 19: The Loan Request Screen

Figure 20: The Meeting Request Screen

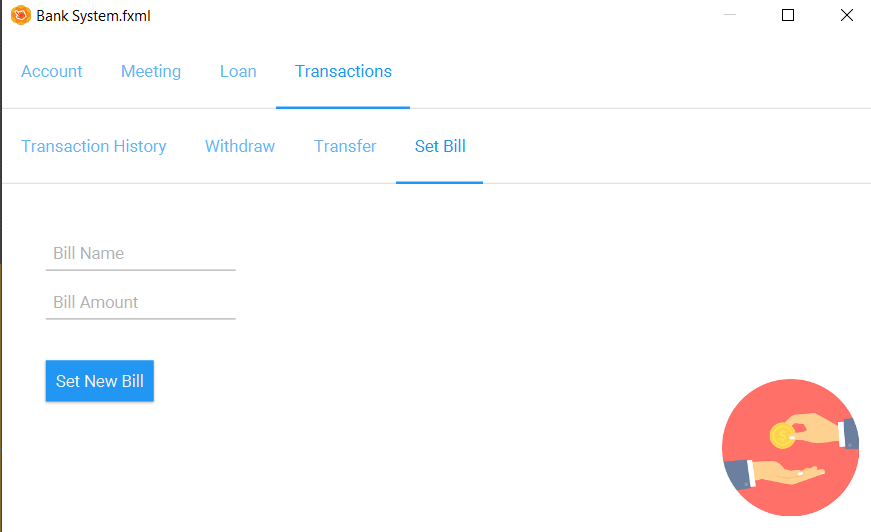
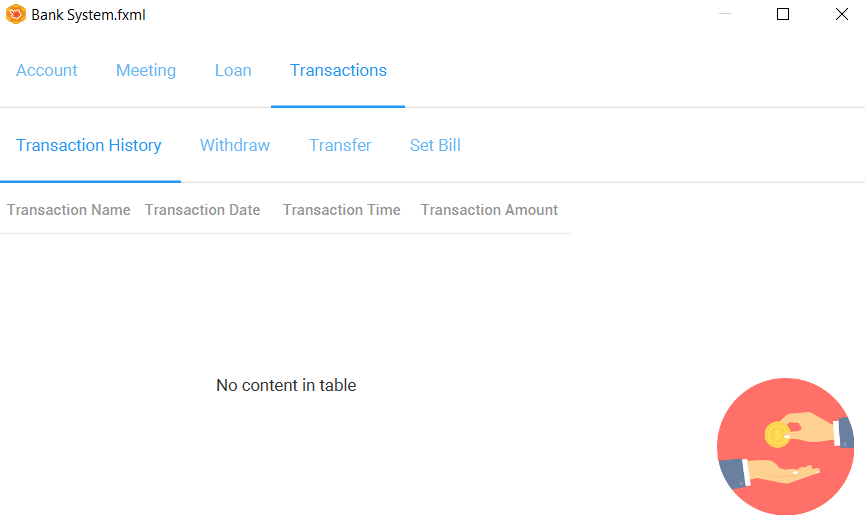


Figure 21 : The Set Bill Screen

Figure 22:The Client Transaction History Screen

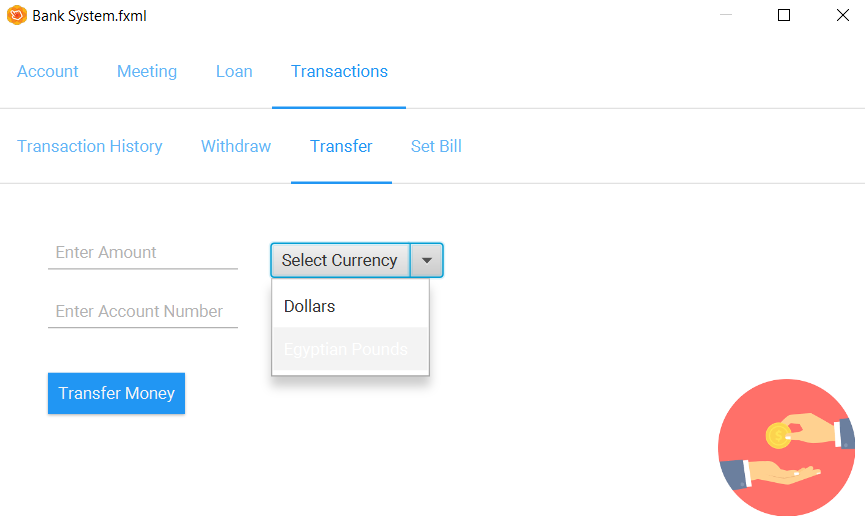
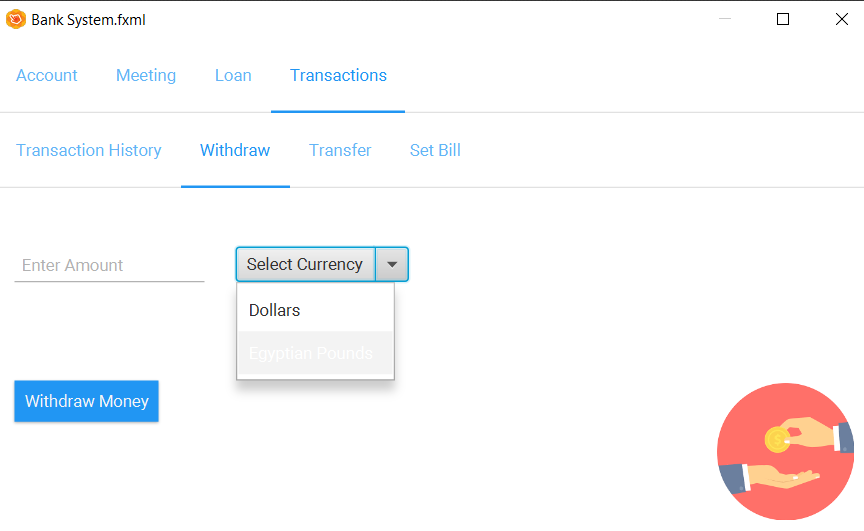


Figure 23: The Withdraw Screen

Figure 24:The Transfer Screen

## C.Tables

Table 1 : Login Use Case Description

|  |  |  |
| --- | --- | --- |
| Use Case Name | Login | |
| Goal in Context | A user requests to access their existing account | |
| Preconditions | None | |
| Successful End Conditions | User logs in successfully | |
| Failed End Conditions | User fails to login | |
| Primary Actors | User | |
| Secondary Actors | None | |
| Trigger | User presses the login button | |
| Included Cases | Verify Password | |
| Main Flow | Step | Action |
|  | 1 | User opens the application |
|  | 2 | Application requests valid username and password |
|  | 3 | User enters their username and password |
|  | 4 | Validation of username and password |
|  | Include :: Verify Password |  |
|  | 5 | User gets access to their account |
| Extensions | 4.1 | Password is invalid |
|  | 4.2 | Message displayed on screen entitled try again |

Table 2: Check Balance Use Case Description

|  |  |  |
| --- | --- | --- |
| Use Case Name | Check Balance | |
| Goal in Context | An existing user requests to check their current balance | |
| Preconditions | Login | |
| Successful End Conditions | User checks their balance successfully | |
| Failed End Conditions | User fails to check their balance | |
| Primary Actors | User | |
| Secondary Actors | Bank | |
| Trigger | User presses check balance tab | |
| Included Cases | None | |
| Main Flow | Step | Action |
|  | 1 | User opens application |
|  | 2 | User logs in successfully |
|  | 3 | User presses on the tab “check balance” |
|  | 4 | Balance is displayed |
|  | 5 | User accesses their balance |

Table 3: Request Loan Use Case Description

|  |  |  |
| --- | --- | --- |
| Use Case Name | Request Loan | |
| Goal in Context | An existing user requests to get a loan | |
| Preconditions | Login | |
| Successful End Conditions | User gets loan successfully | |
| Failed End Conditions | User fails to get loan | |
| Primary Actors | User | |
| Secondary Actors | Bank Manager | |
| Trigger | User presses request loan | |
| Included Cases | None | |
| Main Flow | Step | Action |
|  | 1 | User opens application |
|  | 2 | User logs in successfully |
|  | 3 | User presses “request loan” |
|  | 4 | User enters desired value of the loan |
|  | 5 | User acquires the loan |
| Extensions | 4.1 | Bank manager denies the loan request |

Table 4: Money Withdrawal Use Case Description

|  |  |  |
| --- | --- | --- |
| Use Case Name | Money Withdrawal | |
| Base Use Case | Money Transaction | |
| Goal in Context | An existing user requests to withdraw specific amount of money | |
| Preconditions | Login | |
| Successful End Conditions | User withdraws money  successfully | |
| Failed End Conditions | User fails to withdraw money | |
| Primary Actors | User | |
| Secondary Actors | Bank | |
| Trigger | User presses money withdrawal tab | |
| Included Cases | -Verify Sufficient Funds  -SMS Notification Sent | |
| Main Flow | Step | Action |
|  | 1 | User opens application |
|  | 2 | User logs in successfully |
|  | 3 | User presses on the tab “Money Withdrawal” |
|  | Include :: Verify Sufficient Fund |  |
|  | 4 | User specifies the amount of money to withdraw |
|  | Include :: SMS Notification Sent |  |
|  | 5 | User takes the money they withdrew |
| Extensions | 4.1 | Display insufficient funds |

Table 5: Transfer Funds Use Case Diagram

|  |  |  |
| --- | --- | --- |
| Use Case Name | Transfer Funds | |
| Base Use Case | Money Transaction | |
| Goal in Context | An existing user requests to transfer specific amount of money | |
| Preconditions | Login | |
| Successful End Conditions | User transfer money  successfully | |
| Failed End Conditions | User fails to transfer money | |
| Primary Actors | User | |
| Secondary Actors | Bank | |
| Trigger | User presses transfer funds tab | |
| Included Cases | -Verify Sufficient Funds  -SMS Notification Sent | |
| Main Flow | Step | Action |
|  | 1 | User opens application |
|  | 2 | User logs in successfully |
|  | 3 | User presses on the tab “Transfer Funds” |
|  | Include :: Verify Sufficient Fund |  |
|  | 4 | User specifies the amount of money to transfer |
|  | Include :: SMS Notification Sent |  |
|  | 5 | Money is transferred |
| Extensions | 4.1 | Display insufficient funds |

Table 6: Make Payment Use Case Description

|  |  |  |
| --- | --- | --- |
| Use Case Name | Make Payment | |
| Base Use Case | Money Transaction | |
| Goal in Context | An existing user requests to make payment with specific amount of money | |
| Preconditions | Login | |
| Successful End Conditions | User makes the payment successfully | |
| Failed End Conditions | User fails to make payment | |
| Primary Actors | User | |
| Secondary Actors | Bank | |
| Trigger | User presses make payment tab | |
| Included Cases | -Verify Sufficient Funds  -SMS Notification Sent | |
| Main Flow | Step | Action |
|  | 1 | User opens application |
|  | 2 | User logs in successfully |
|  | 3 | User presses on the tab “Make Payment” |
|  | Include :: Verify Sufficient Fund |  |
|  | 4 | User specifies the amount of money to make payment with |
|  | Include :: SMS Notification Sent |  |
|  | 5 | Payment is done |
| Extensions | 4.1 | Display insufficient funds |