Software Requirements Specification

for

<Bank Management System>

Prepared by : Moamen Mohmed Abd Elrazik

Table of Contents

1.0. Introduction

1.1. Purpose

The purpose of this document is to present a detailed description of the bank system software project. It aims to provide a comprehensive outline of the system's purpose, features, interfaces, functionalities, constraints, and responses to external stimuli. This document serves as a guide for stakeholders, developers, and end-users, detailing the scope and objectives of the project to ensure a clear understanding and smooth implementation.

1.2. Project Proposal

In today's fast-paced and technologically driven world, the banking sector is increasingly dependent on advanced software systems to manage a multitude of financial operations, customer interactions, and regulatory compliance requirements. The proposed project seeks to develop a comprehensive, reliable, and secure bank system using Java. This system is intended to streamline banking operations, enhance customer service, and ensure robust security measures.

Bank System Project is envisioned to be an all-encompassing platform that supports a wide range of banking activities. The primary objectives of the system include:

Efficient Account Management: The system will enable customers to effortlessly open, close, and manage their bank accounts. This includes features for updating personal information, viewing account balances, and managing multiple accounts.

Seamless Transaction Processing: The system will support various types of financial transactions such as deposits, withdrawals, transfers, bill payments, and direct debits. It will ensure that transactions are processed quickly, accurately, and securely.

Enhanced Customer Service: The system will include tools for customer support, allowing bank staff to handle inquiries, resolve issues, and provide assistance efficiently. Features such as live chat support, FAQs, and automated responses will be integrated to enhance the customer experience.

Robust Security Measures: Given the sensitivity of financial data, the system will incorporate strong security protocols to protect against unauthorized access, fraud, and cyber threats. This will include encryption, multi-factor authentication, and regular security audits.

Regulatory Compliance: The system will be designed to comply with all relevant financial regulations and standards. This includes anti-money laundering (AML) policies, know your customer (KYC) procedures, and data protection laws.

User-Friendly Interface: The system will feature an intuitive and easy-to-navigate interface, ensuring that users of all technical abilities can operate it with ease. This includes both the customer-facing portal and the administrative backend for bank staff.

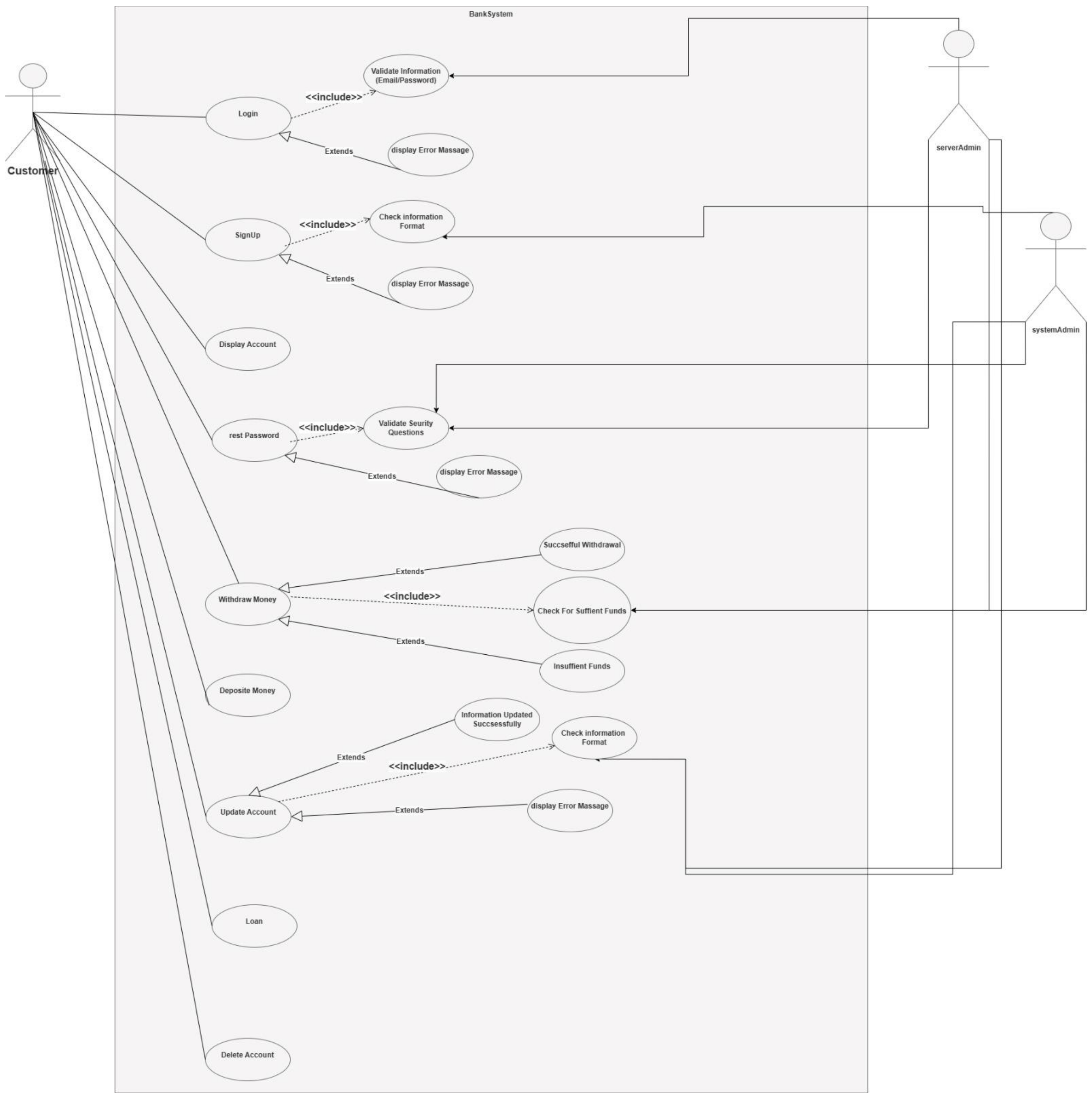
Scalability and Flexibility: The system will be scalable to accommodate the growth of the bank, whether it be in terms of user base, transaction volume, or new services. It will also be flexible enough to integrate future technological advancements and new features as needed.

Comprehensive Reporting and Analytics: The system will include powerful reporting and analytics tools to help bank management make informed decisions. This includes financial reporting, customer behavior analysis, and performance metrics.

By developing this bank system, we aim to provide a platform that not only meets the current needs of the bank and its customers but also positions the bank for future growth and technological evolution. The overarching goal is to enhance the overall banking experience, improve operational efficiency, and maintain the highest standards of security and compliance.

2.0. Overall Description

2.1 System Environment



2.2 Functional Requirements Specification

This section outlines the functional requirements of the bank system by detailing the use cases for each activity that the system will support. The use cases describe the interactions between the users (actors) and the system, specifying how the system should respond to different inputs and actions.

2.2.1 Register users Use Case

Use Case: Account Create

Brief Description: To utilize the bank system, a user must first register by creating an account. This process ensures that only authorized users can access and benefit from the system's features.

Initial Step-By-Step Description: Before this use case can be initiated, the user must have already installed or accessed the bank system application.

Steps:

Create Account:

Actor: User

Description: The user initiates the account creation process by providing necessary personal information such as name, email, and password.

System Response: The system validates the entered information to ensure it meets the required format and completeness. If the information is valid, the system proceeds to the next step; otherwise, it prompts the user to correct the errors.

Activate Account:

Actor: System

Description: The system sends an account activation link or code to the user's registered email address.

System Response: The user must follow the instructions in the email to activate their account. This often involves clicking a link or entering a code back into the system. Upon successful activation, the account becomes active and ready for use.

Reset Password:

Actor: User

Description: If the user forgets their password, they can initiate a password reset.

System Response: The system sends a password reset link or code to the user's registered email address. The user follows the instructions to set a new password. The system validates the new password and updates the user's account accordingly.

Access System:

Actor: User

Description: Once the account is created and activated, the user can log in using their credentials.

System Response: The system validates the credentials and grants access to the user. The user can then utilize the various features of the bank system.

Detailed Use Case Description

Title Account Create

Primary Actor User

Goal To create and activate a new user account in the bank system.

Preconditions

The user has access to the bank system application.

The user has a valid email address.

Post conditions

The user has an active account in the bank system.

The user can log in and use the system's features.

Main Success Scenario:

The user opens the bank system application and navigates to the registration page.

The user enters the required personal information (name, email, password).

The system validates the entered information.

The system sends an activation email to the user.

The user receives the email and follows the activation instructions.

The system confirms the activation and enables the user account.

The user logs in using the registered email and password.

The system grants access, and the user can now use the system.

Extensions

Invalid Information If the user enters invalid information during registration (e.g., incorrect email format), the system displays an error message and prompts the user to correct the information.

Activation Email Not Received If the user does not receive the activation email, they can request the system to resend it.

Forgotten Password: If the user forgets their password, they can initiate the password reset process to regain access.

Special Requirements:

The system should use secure methods to validate user information.

The activation email should be sent promptly and contain clear instructions.

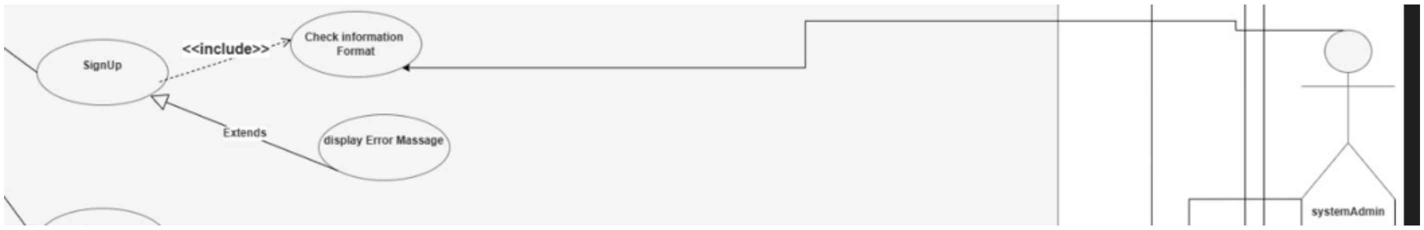
Passwords should be stored securely using encryption.

Frequency of Use: As needed, whenever a new user wants to register.

Assumptions:

Users have internet access to receive the activation email.

Users will provide accurate and valid information during registration.



2.2.2 User Use Case

Brief Description Once the user has registered and created an account in the bank system, they gain access to several functionalities. This use case outlines the various interactions a registered user can have with the system. These interactions include managing their personal information, accessing account details, and utilizing the services provided by the bank system.

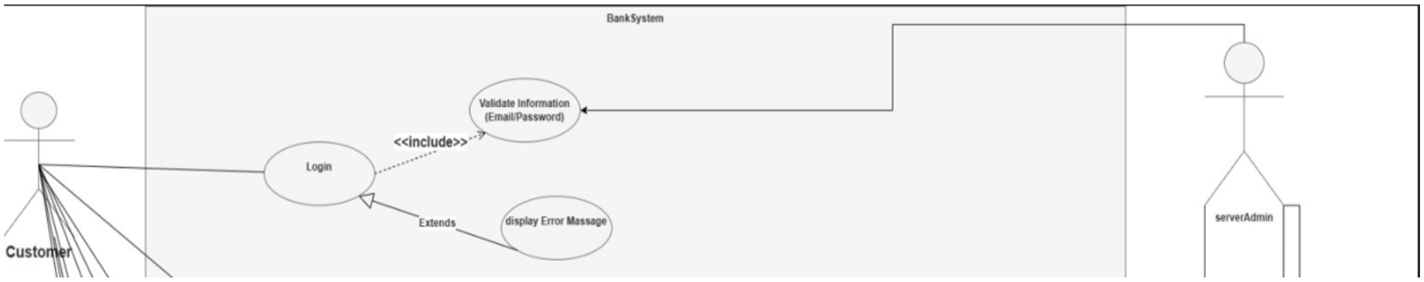
User Use Case Diagram

Elements

email: The primary actor interacting with the system.

Status: The current status of the user account (e.g., active, inactive, pending verification).

Account Info: Detailed information about the user's bank accounts, including balances, transaction history, and account settings.



Detailed Use Case Descriptions

Use Case: Manage Account

Brief Description The user can manage their account settings, update personal information, reset their password, or delete their account.

Primary Actor: User

Steps:

Update Information:

Description: The user updates their personal information (e.g., address, phone number).

System Response: The system validates the new information and updates the user's profile.

Reset Password:

Description: The user initiates a password reset.

System Response: The system sends a reset link to the user's email. The user follows the link to set a new password.

Delete Account:

Description: The user requests to delete their account.

System Response: The system confirms the request and permanently deletes the account after verification.

Use Case: Access Account Info

Brief Description: The user can view their account details, including balances, transaction history, and account settings.

Primary Actor: User

Steps:

View Balance:

Description: The user checks their current account balance.

System Response: The system retrieves and displays the balance information.

View Transactions:

Description: The user views their transaction history.

System Response: The system retrieves and displays recent transactions.

Account Settings:

Description: The user accesses their account settings to make changes or review details.

System Response: The system displays the settings, allowing the user to make adjustments as needed.

Use Case: Mail Box Access

Brief Description: The user can access their mailbox within the system to read notifications and messages from the bank.

Primary Actor: User

Steps:

Read Notifications:

Description: The user reads notifications about account activity or system updates.

System Response: The system displays the notifications.

View Messages:

Description: The user views messages sent by the bank, such as alerts or promotional offers.

System Response: The system displays the messages in the user's mailbox.

Special Requirements

Security: Ensure all personal and financial data is encrypted and protected against unauthorized access.

Usability: The user interface should be intuitive and user-friendly to facilitate easy navigation and interaction.

Accessibility: The system should be accessible to users with disabilities, adhering to relevant accessibility standards.

Performance: The system should be responsive, with minimal latency in retrieving and displaying information.

Frequency of Use

These use cases are expected to be utilized frequently by users as part of their regular interactions with the bank system. Managing account information, accessing account details, and reading notifications are routine activities essential to the user experience.

Assumptions

Users have a registered account and are logged into the system.

Users have a valid email address for receiving notifications and password resets.

The system is operational and accessible to users.

2.2.3 Use Case: User volunteer

Brief Description: This use case allows a user to volunteer to help other users within the bank system. Before this use case can be initiated, the user must have accessed the main page of the application.

Use Case Diagram

Elements:

User: The primary actor who is logged into the system and wants to volunteer.

Volunteer: The action of offering help or services to other users.

Detailed Use Case Description

Use Case: Volunteer

Primary Actor: User

Goal: To enable users to volunteer and help other users within the system.

Preconditions:

The user is logged into the system.

The user has accessed the main page of the application.

Post conditions:

The user has volunteered to help and can view available tasks or requests from other users.

The user can provide assistance and mark tasks as complete once the help is provided.

Main Success Scenario:

Access Main Page:

Description: The user accesses the main page of the bank system application.

System Response: The system displays the main page with options to volunteer.

Volunteer:

Description: The user selects the option to volunteer to help other users.

System Response: The system confirms the user's intent to volunteer and displays available tasks or requests for help.

View Volunteer Tasks:

Description: The user browses through a list of tasks or requests for help from other users.

System Response: The system displays the tasks, including details and requirements.

Select Task to Help:

Description: The user selects a task they wish to help with.

System Response: The system assigns the task to the user and provides necessary details for assistance.

Assist Other Users:

Description: The user provides the requested support or help to the other user.

System Response: The system tracks the assistance provided and allows the user to mark the task as complete once done.

Extensions:

No Available Tasks: If there are no available tasks or requests for help, the system notifies the user and suggests they check back later.

Task Already Taken: If another user has already selected the task, the system informs the user and updates the task list.

Special Requirements:

Notification System: Notify users when new tasks or help requests are posted.

Feedback Mechanism: Allow users to provide feedback on the help they received or provided.

User Verification: Ensure that users volunteering are verified and trusted within the system.

Frequency of Use: Users may volunteer as often as they wish, depending on the availability of tasks and their willingness to help.

Assumptions:

Users are motivated to help others and will volunteer in good faith.

The system can accurately track and manage volunteer tasks and assistance provided.

2.2.4 Help CLASS Diagram

Brief Description: In the context of the bank system, this CLASS diagram outlines the main entities and their relationships involved in the system. The diagram includes details about clients, employees, and the roles within the bank system.

Class Diagram: Help Asking

Entities and Relationships:

Client: Represents a customer who uses the banking services.

Bank: Represents the bank institution.

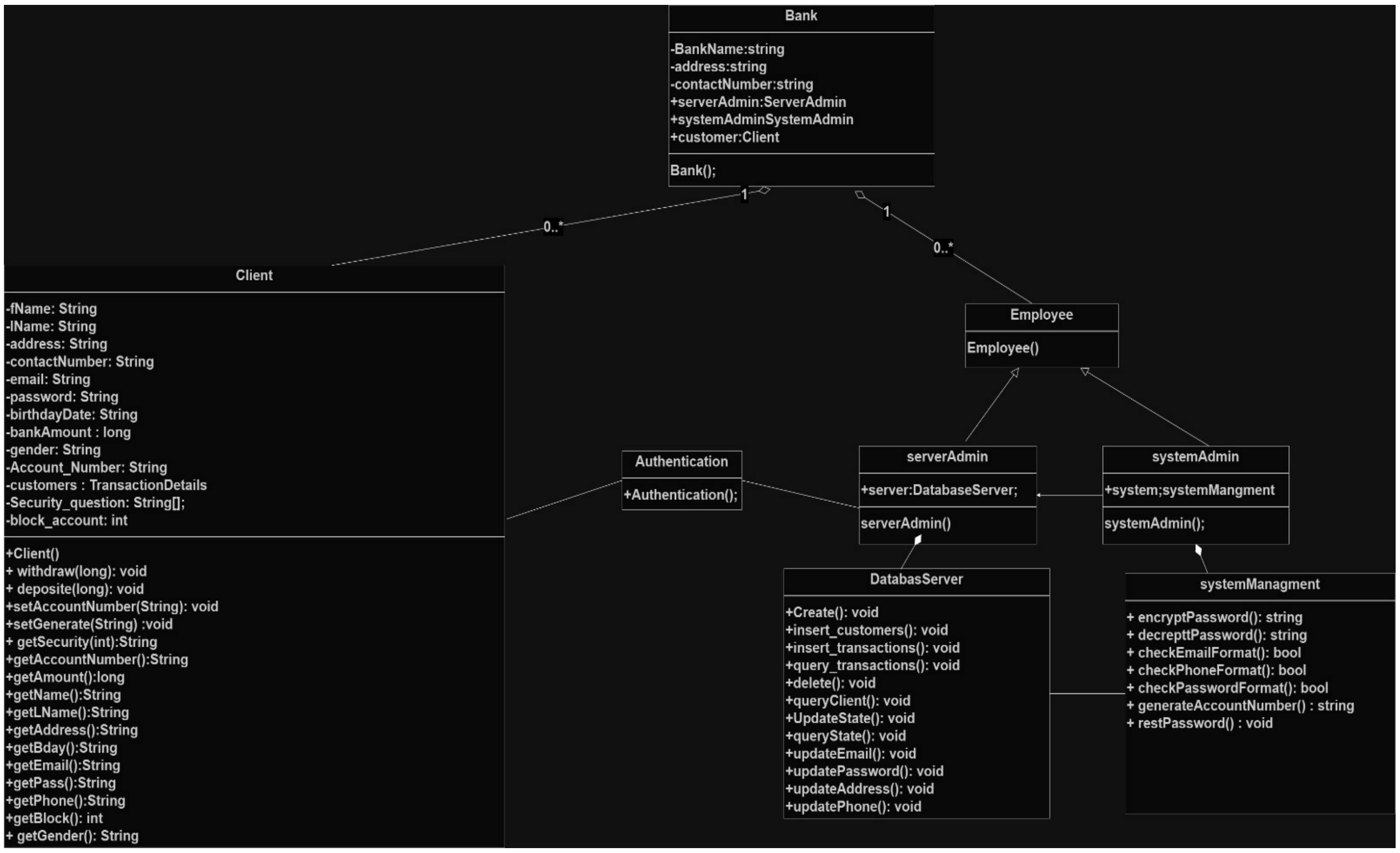
Employee: Represents the staff working for the bank, which includes server administrators and system administrators.

Authentication: Manages the login and verification process.

Database Server: Handles database-related operations.

System Management: Manages system-wide tasks and operations.

Figure 5 - Help CLASS Diagram:



Detailed Description of Entities

Client:

Attributes:

fName: First name of the client.

lName: Last name of the client.

address: Address of the client.

contactNumber: Contact number of the client.

email: Email address of the client.

password: Password for client authentication.

birthdayDate: Birthday of the client.

bankAmount: Current bank balance of the client.

gender: Gender of the client.

Account_Number: Account number of the client.

customers: Transaction details associated with the client.

Security_question: Array of security questions for password recovery.

block_account: Indicator if the account is blocked (1 for blocked, 0 for active).

Methods:

Client(): Constructor to initialize the client.

withdraw(long): Method to withdraw money.

deposite(long): Method to deposit money.

setAccountNumber(String): Set the account number.

setGenerate(String): Generate necessary information.

getSecurity(int): Get security details.

getAccountNumber(): Get the account number.

getAmount(): Get the amount.

getName(): Get the name.

getAddress(): Get the address.

getEmail(): Get the email.

getPhone(): Get the phone number.

getBday(): Get the birthday.

getPass(): Get the password.

getBlock(): Get block status.

getGender(): Get the gender.

Bank:

Attributes:

BankName: Name of the bank.

address: Address of the bank.

contactNumber: Contact number of the bank.

serverAdmin: Association with ServerAdmin.

systemAdmin: Association with SystemAdmin.

customer: Association with Client.

Methods:

Bank(): Constructor to initialize the bank.

Employee:

Methods:

Employee(): Constructor to initialize the employee.

ServerAdmin (inherits from Employee):

Attributes:

server: Association with DatabaseServer.

Methods:

serverAdmin(): Constructor to initialize the server admin.

SystemAdmin (inherits from Employee):

Attributes:

system: Association with SystemManagement.

Methods:

systemAdmin(): Constructor to initialize the system admin.

Authentication:

Methods:

Authentication(): Constructor to initialize the authentication process.

DatabaseServer:

Methods:

Create(): Create database entries.

insert_customers(): Insert customer details.

insert_transactions(): Insert transaction details.

query_transactions(): Query transactions.

delete(): Delete entries.

queryClient(): Query client details.

UpdateState(): Update the state of the database.

updateEmail(): Update email details.

updatePassword(): Update password.

updateAddress(): Update address.

updatePhone(): Update phone number.

SystemManagement:

Methods:

encryptPassword(): Encrypt passwords.

decryptPassword(): Decrypt passwords.

checkEmailFormat(): Check email format.

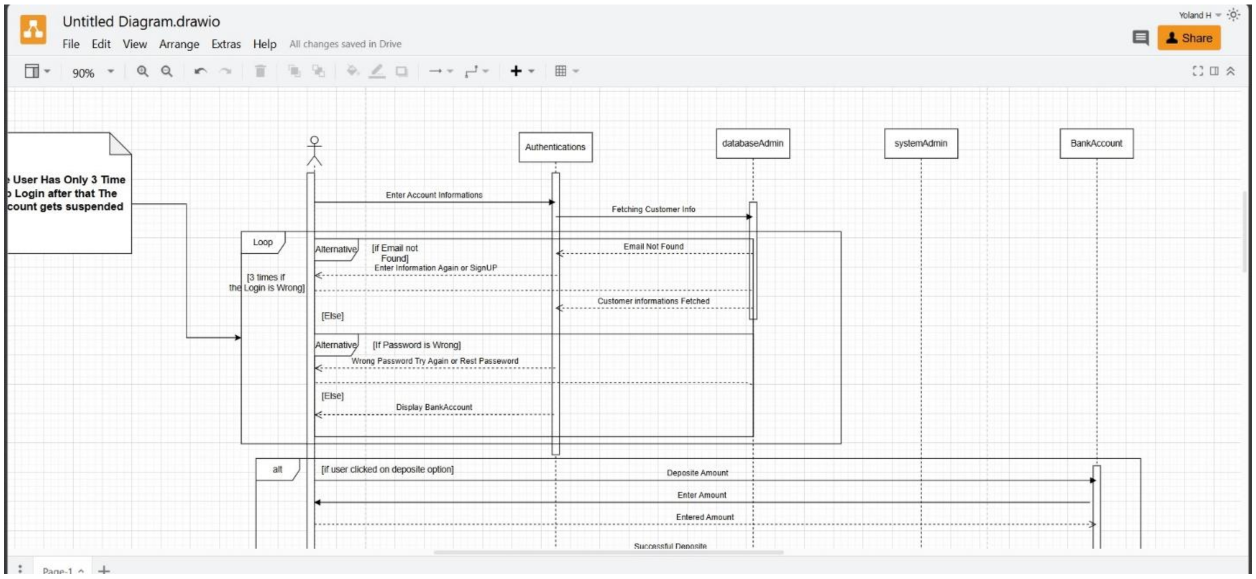
checkPhoneFormat(): Check phone format.

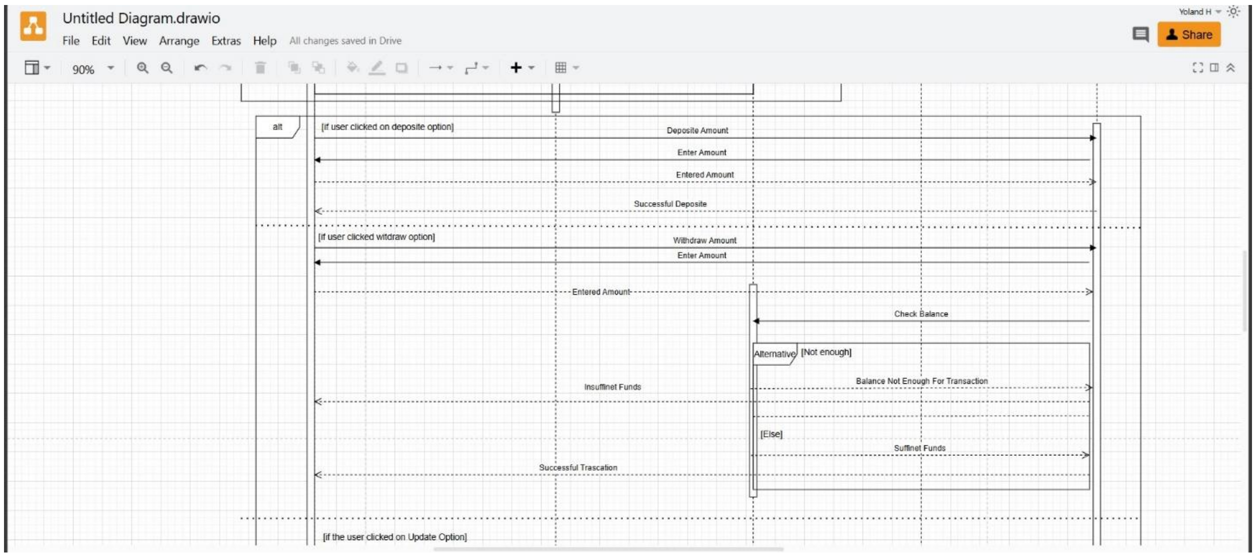
checkPasswordFormat(): Check password format.

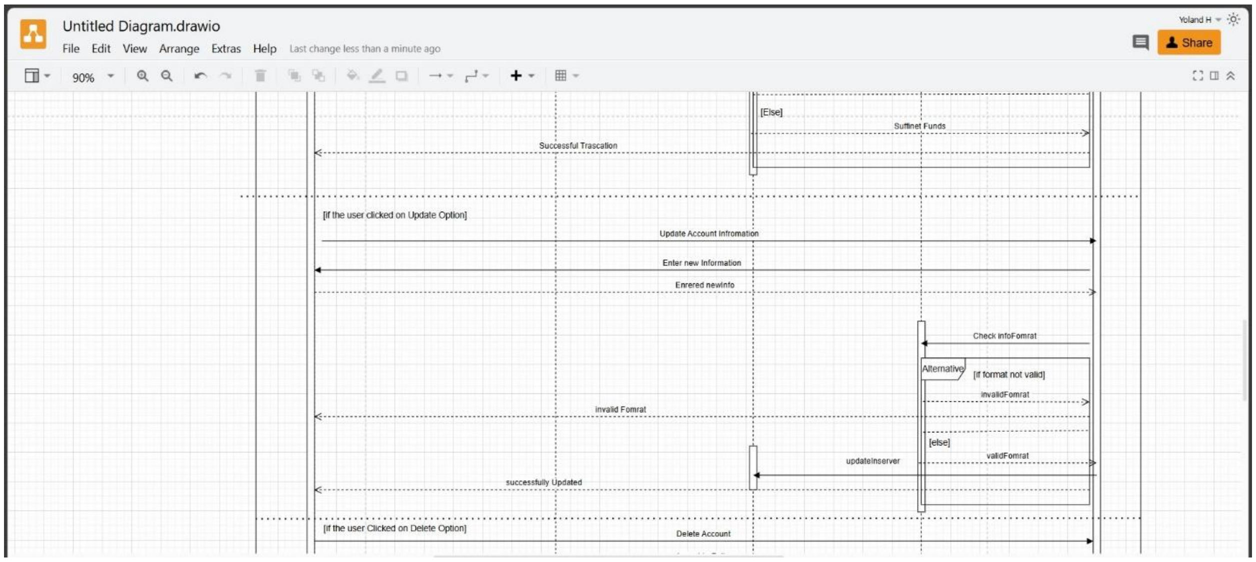
generateAccountNumber(): Generate account numbers.

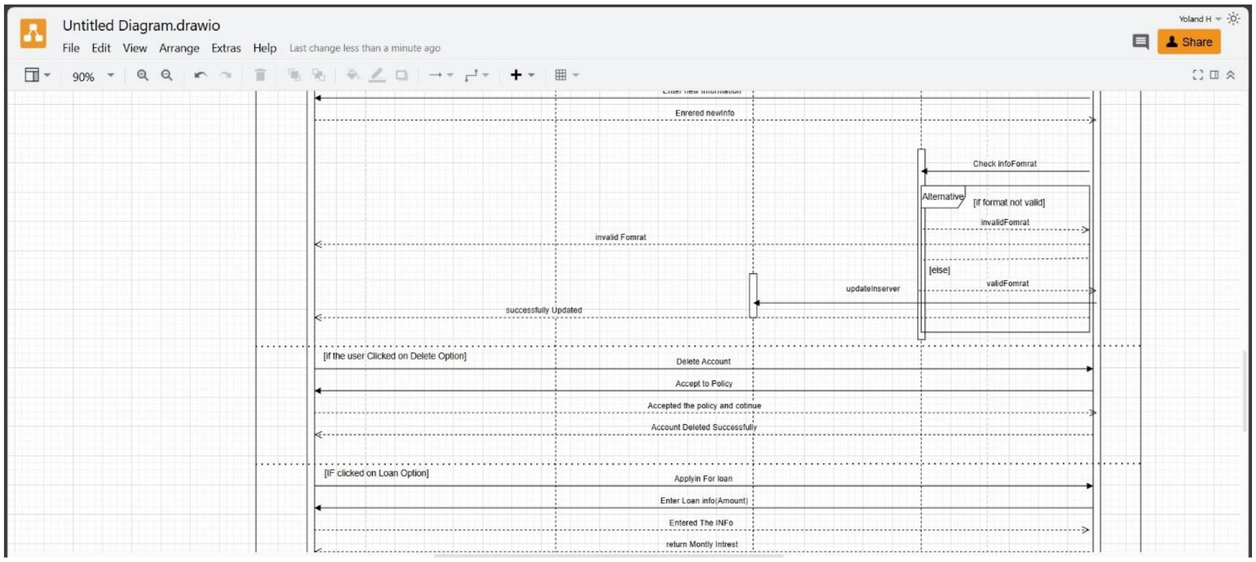
resetPassword(): Reset passwords.

2.2.5 System Work Flow (Sequence diagram)









The sequence diagram provides a graphical representation of the flow of data through the bank system, illustrating how data is processed by the system in terms of inputs and outputs. This helps to understand the movement and transformation of data within the system.

Entities:

Customer: Interacts with the system to perform various banking operations.

Bank System: Central process handling all operations.

Server Admin: Manages server-related tasks.

System Admin: Manages system-wide configurations.

Level 1: Detailed Sequence Flow Diagram

The Level 1 breaks down the main process into subprocesses, showing more detail about how data flows between the different parts of the system.

Processes:

P1.0: User Authentication

P2.0: Account Management

P3.0: Transaction Processing

P4.0: System Management

Data Stores:

D1: User Database

D2: Transaction Database

Details:

Process P1.0: User Authentication

Inputs: User credentials

Outputs: Authentication token, error messages

Data Stores Accessed: D1 (User Database)

Process P2.0: Account Management

Inputs: User details, account update requests

Outputs: Account details, confirmation messages

Data Stores Accessed: D1 (User Database)

Process P3.0: Transaction Processing

Inputs: Transaction requests (withdrawal, deposit)

Outputs: Transaction confirmation, account balance updates

Data Stores Accessed: D2 (Transaction Database)

Process P4.0: System Management

Inputs: Administrative commands

Outputs: System configuration updates, audit logs

Data Stores Accessed: D1 (User Database), D2 (Transaction Database)

By breaking down the system into these detailed processes and data flows, the sequence provides a clear and comprehensive view of how the bank system operates and handles data. This helps in understanding the system's architecture and identifying areas for optimization and improvement.

2.3 Non-Functional Requirements

Non-functional requirements define the overall qualities or attributes of the system. These requirements specify criteria that can be used to judge the operation of the system, rather than specific behaviors or functions.

2.3.1 Performance Requirements

Response Time: The system must respond to user inputs within 2 seconds for 95% of transactions.

Throughput: The system should handle up to 1000 transactions per second to accommodate peak load times.

Scalability: The system must be able to scale horizontally to support an increasing number of users without degradation in performance.

Availability: The system should have an uptime of 99.99%, ensuring high availability for users.

2.3.2 Reliability Requirements

Data Integrity: The system must ensure the accuracy and consistency of stored data. Transactions should be processed in a manner that maintains data integrity.

Fault Tolerance: The system should be able to recover from hardware failures and continue operation without loss of data or transactions.

Error Handling: The system must gracefully handle errors and provide meaningful error messages to users without exposing sensitive information.

2.3.3 Security Requirements

Authentication: The system must ensure secure authentication mechanisms, including two-factor authentication for sensitive transactions.

Authorization: Only authorized users should have access to certain functionalities and data within the system. Role-based access control should be implemented.

Data Encryption: All sensitive data, including user credentials and transaction details, should be encrypted both in transit and at rest.

Audit Logging: The system must log all user activities, especially those related to financial transactions, for auditing and compliance purposes.

2.3.4 Usability Requirements

User Interface: The system should provide an intuitive and user-friendly interface that is easy to navigate for users with varying levels of technical proficiency.

Accessibility: The system must comply with accessibility standards to ensure that users with disabilities can access and use the application.

Documentation: Comprehensive user documentation and help resources should be available to assist users in understanding and utilizing the system's features.

2.3.5 Maintainability Requirements

Modularity: The system should be designed in a modular way to facilitate easier maintenance and upgrades.

Code Quality: The system must adhere to coding standards and best practices to ensure that the codebase is clean, well-documented, and maintainable.

Error Reporting: The system should include robust error reporting mechanisms to help developers quickly identify and resolve issues.

2.3.6 Compliance Requirements

Regulatory Compliance: The system must comply with relevant banking regulations and standards, including data protection laws such as GDPR and PCI-DSS.

Audit Compliance: The system should provide features that support internal and external audits, ensuring that all financial activities can be tracked and verified.

2.3.7 Portability Requirements

Platform Independence: The system should be able to run on various operating systems (e.g., Windows, Linux) without requiring significant modifications.

Browser Compatibility: The web-based components of the system must be compatible with major web browsers (e.g., Chrome, Firefox, Safari, Edge).

2.3.8 Interoperability Requirements

API Integration: The system must provide APIs for integration with other systems and services, such as third-party payment gateways and financial institutions.

Data Exchange: The system should support standard data exchange formats (e.g., JSON, XML) to facilitate communication with external systems.

By meeting these non-functional requirements, the bank system will ensure a high level of performance, security, usability, and maintainability, thereby providing a reliable and efficient service to its users.

3.1 Description of Procedures and Functions

3.1.1 User Management

Procedure: User Registration

Function: Allow users to create an account.

Steps:

User submits registration form with required details (username, password, email, phone number).

System validates the input data.

System hashes the password.

System stores user details in the database.

System sends a verification email to the user.

Outcome: New user account is created and email verification is initiated.

Procedure: User Authentication

Function: Authenticate users to access the system.

Steps:

User submits login credentials (username and password).

System validates the credentials.

System verifies the password hash.

System generates a JWT token for the session.

System returns the token to the client.

Outcome: User is logged in and receives an authentication token.

Procedure: Profile Management

Function: Enable users to update their profile information.

Steps:

User accesses the profile management section.

User submits updates to profile details.

System validates the input data.

System updates the user profile in the database.

Outcome: User profile is updated.

Procedure: Password Reset

Function: Allow users to reset their password.

Steps:

User requests a password reset.

System sends a password reset link to the user's email.

User clicks the link and submits a new password.

System validates the new password.

System hashes the new password and updates the database.

Outcome: User password is reset.

3.1.2 Account Management

Procedure: Account Information Retrieval

Function: Display user account details.

Steps:

User requests account information.

System retrieves account details from the database.

System returns account details to the user.

Outcome: User views account information.

Procedure: Fund Deposit

Function: Allow users to deposit funds into their account.

Steps:

User initiates a deposit transaction.

System validates the deposit amount.

System updates the account balance in the database.

System records the transaction in the transaction history.

Outcome: Funds are deposited into the user's account.

Procedure: Fund Withdrawal

Function: Allow users to withdraw funds from their account.

Steps:

User initiates a withdrawal transaction.

System validates the withdrawal amount and checks for sufficient funds.

System updates the account balance in the database.

System records the transaction in the transaction history.

Outcome: Funds are withdrawn from the user's account.

Procedure: Fund Transfer

Function: Enable users to transfer funds between accounts.

Steps:

User initiates a fund transfer.

System validates the transfer details (source account, destination account, amount).

System checks for sufficient funds in the source account.

System updates the balances of the source and destination accounts in the database.

System records the transaction in the transaction history.

Outcome: Funds are transferred between accounts.

3.1.3 Transaction Management

Procedure: Transaction Processing

Function: Process user transactions in real-time.

Steps:

User initiates a transaction (deposit, withdrawal, transfer).

System validates the transaction details.

System performs the transaction and updates account balances.

System records the transaction in the transaction history.

Outcome: Transaction is processed and recorded.

Procedure: Transaction History Retrieval

Function: Allow users to view their transaction history.

Steps:

User requests transaction history.

System retrieves transaction records from the database.

System returns the transaction history to the user.

Outcome: User views transaction history.

3.1.4 Notification System

Procedure: Sending Notifications

Function: Send alerts and notifications to users.

Steps:

Event triggers a notification (e.g., successful transaction, low balance).

System generates the notification message.

System sends the notification via the configured channels (email, SMS, in-app).

Outcome: User receives the notification.

Procedure: Notification Preferences Management

Function: Allow users to manage their notification preferences.

Steps:

User accesses the notification preferences section.

User updates preferences (e.g., email notifications, SMS notifications).

System stores the updated preferences in the database.

Outcome: User notification preferences are updated.

3.2 Hardware and Software Requirements

3.2.1 Hardware Requirements

Server: A reliable server to host the application. Specifications depend on the number of users but typically include:

Processor: Multi-core CPU

RAM: 8 GB or higher

Storage: SSD with at least 100 GB of space

Client Devices: Users can access the application on any modern device with internet access, including desktops, laptops, tablets, and smartphones.

3.2.2 Software Requirements

Java Development Kit (JDK): Version 8 or higher for developing and running Java applications.

Integrated Development Environment (IDE): IntelliJ IDEA or Eclipse for coding and debugging.

XML Parser: For reading and writing XML configuration files.

Build Tools: Apache Ant or Maven for managing project builds and dependencies.

Version Control System: Git for source code management and collaboration.

Operating System: Any modern operating system such as Windows, macOS, or Linux for development and deployment.

3.3 System Interfaces

The system interfaces for the banking application include a graphical user interface (GUI) for user interaction and XML files for data storage and configuration.

3.3.1 Graphical User Interface (GUI)

Login Screen: Allows users to enter their credentials to access their accounts.

Dashboard: Displays user and account information, transaction history, and options for various actions such as deposit, withdrawal, and loan applications.

Profile Management: Provides forms for updating user information such as email, password, address, and phone number.

Transaction History: Shows a detailed list of all transactions with relevant details.

3.3.2 XML Configuration

User Data: Stored in XML files, including personal information, account details, and transaction history.

Application Settings: Configuration settings for the application, such as security parameters and system preferences.

4.0 Implementation and Testing

The implementation and testing phase is crucial for ensuring that the banking application is developed according to the specified requirements and functions correctly in various scenarios. This section outlines the approach, strategies, and tools used for the implementation and testing of the system.

4.1 Introduction

The implementation phase involves translating the system design into actual code and integrating various components to build a functional application. The primary programming language used for this project is Java, with XML used for configuration and data representation. The testing phase involves verifying that the system meets the specified requirements and functions correctly under various conditions. Testing includes unit testing, integration testing, system testing, and user acceptance testing (UAT). Automated testing tools such as JUnit are used for unit tests, while manual testing is conducted for system and UAT.

4.2 Implementation

4.2.1 Development Environment

Programming Language: Java

Configuration Language: XML

Build Tool: Apache Ant / Maven

Version Control: Git (GitHub/GitLab)

IDE: IntelliJ IDEA / Eclipse

4.2.2 Steps in Implementation

Setup Development Environment:

Install JDK and configure the development environment.

Set up the project structure and configure build tools such as Apache Ant or Maven.

Define the XML schema for data representation.

Develop Core Modules:

User Management: Implement user registration, authentication, profile management, and password reset functionalities.

Account Management: Develop modules for account information retrieval, fund deposit, withdrawal, and transfer.

XML Configuration:

Use XML files for configuring application settings, user data, and account information.

Ensure that XML parsing is handled efficiently in the application.

Integration:

Integrate various modules and ensure seamless data flow using XML for configuration and data representation.

Implement security features such as input validation to prevent unauthorized access.

Deployment:

Package the application into a deployable format (e.g., JAR file).

Deploy the application on the target environment.

4.3 Testing

4.3.1 Testing Strategies

Unit Testing:

Objective: Test individual components or modules in isolation.

Tools: JUnit

Scope: Test services, utilities, and classes to ensure they function as expected.

Integration Testing:

Objective: Test the interaction between different modules or components.

Tools: JUnit, XML-based test data

Scope: Ensure that components integrate correctly and data flow between modules is accurate.

System Testing:

Objective: Test the complete system to verify that it meets the specified requirements.

Tools: Manual testing, JUnit for automated tests

Scope: Validate the entire application workflow from end to end.

User Acceptance Testing (UAT):

Objective: Ensure the system meets the business requirements and is ready for production use.

Tools: Manual testing

Scope: Conducted by end-users or stakeholders to validate real-world usage scenarios.

4.3.2 Test Cases and Scenarios

User Registration:

Verify that the registration form accepts valid input and creates a new user.

Ensure that invalid input (e.g., weak passwords, invalid email formats) is handled correctly.

User Authentication:

Validate that users can log in with correct credentials.

Check that invalid login attempts are handled securely.

Profile Management:

Ensure users can update their profile information.

Password Reset:

Test the password reset process, including email verification and new password submission.

Account Management:

Verify that users can view account details.

Test deposit, withdrawal, and transfer functionalities for various scenarios, including edge cases like insufficient funds.

4.4 Tools and Technologies

JUnit: For unit testing Java components.

Apache Ant / Maven: For build automation.

4.5 Layouts

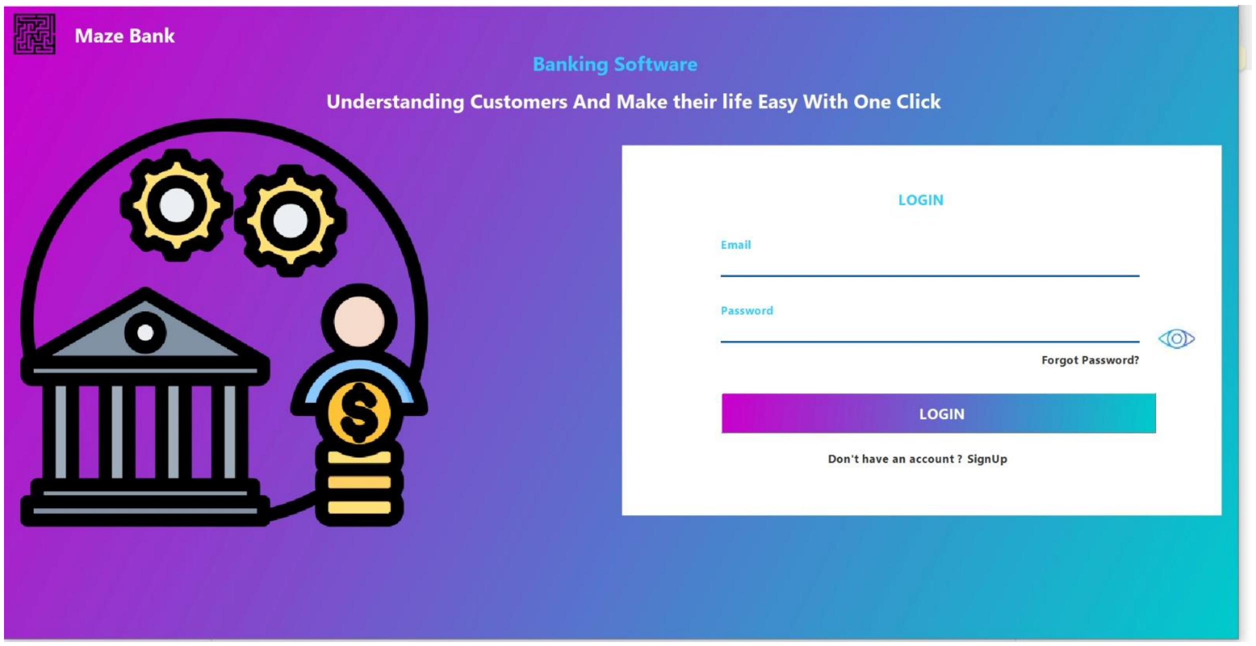
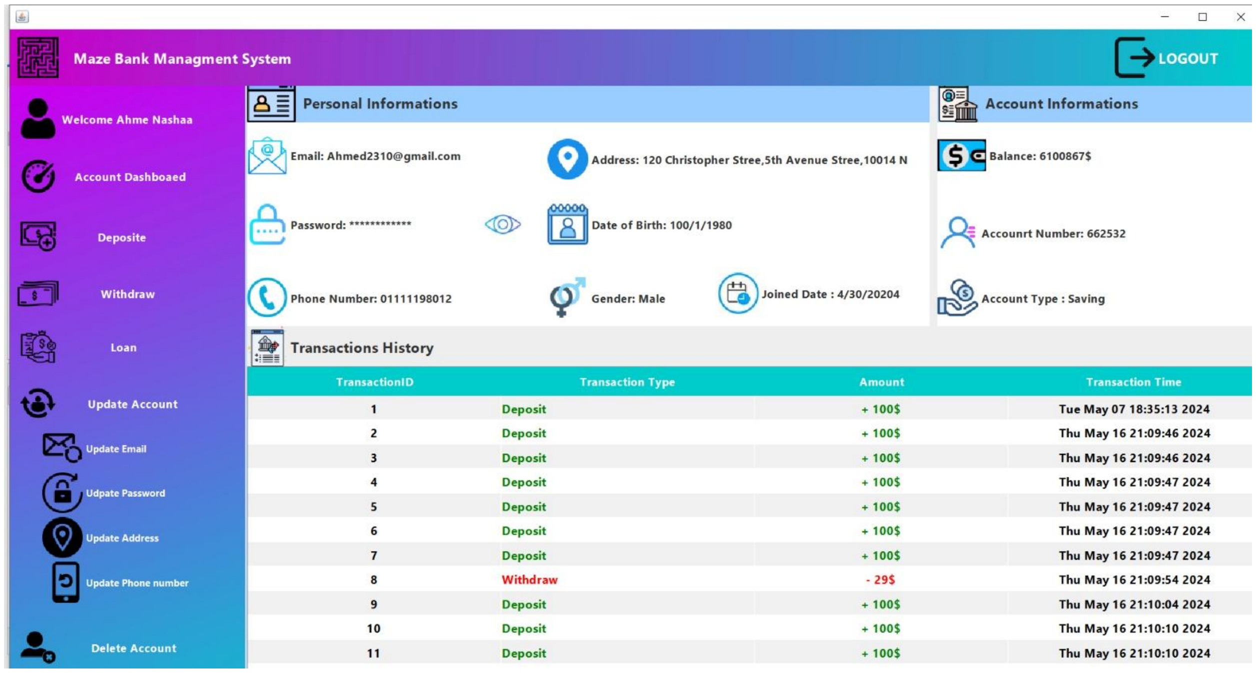

Maze Bank
Banking Software
Understanding Customers And Make their life Easy With One Click
LOGIN
Email
Password
Forgot Password?
LOGIN
Don't have an account ? SignUp

Figure 1-Login

Figure 2-*Account Page*



Personal Informations

Email: Ahmed2310@gmail.com

Address: 120 Christopher Stree,5th Avenue Stree,10014 N

Password: ************

Date of Birth: 100/1/1980

Phone Number: 01111198012

Gender: Male

Joined Date : 4/30/20204

Account Informations

Balance: 6100867$

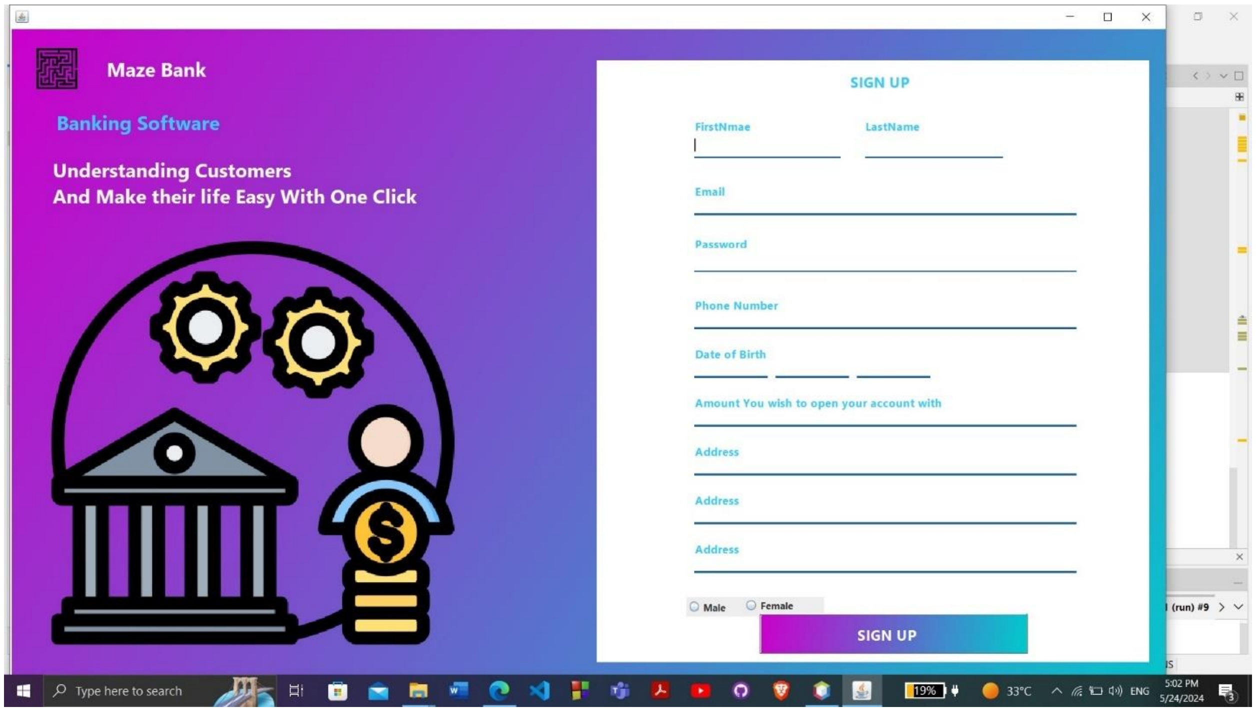
Accounrt Number: 662532

Account Type : Saving

Transactions History

TransactionID	Transaction Type	Amount	Transaction Time
1	Deposit	+ 100$	Tue May 07 18:35:13 2024
2	Deposit	+ 100$	Thu May 16 21:09:46 2024
3	Deposit	+ 100$	Thu May 16 21:09:46 2024
4	Deposit	+ 100$	Thu May 16 21:09:47 2024
5	Deposit	+ 100$	Thu May 16 21:09:47 2024
6	Deposit	+ 100$	Thu May 16 21:09:47 2024
7	Deposit	+ 100$	Thu May 16 21:09:47 2024
8	Withdraw	- 29$	Thu May 16 21:09:54 2024
9	Deposit	+ 100$	Thu May 16 21:10:04 2024
10	Deposit	+ 100$	Thu May 16 21:10:10 2024
11	Deposit	+ 100$	Thu May 16 21:10:10 2024

Figure 3-Sign Up Page



References

How to program this GUI in Java Swing - Stack Overflow

Swing (Java) - Wikipedia

مخطط التتابع - ويكيبيديا (wikipedia.org)

Class diagram - Wikipedia