Name: Majed Hussein

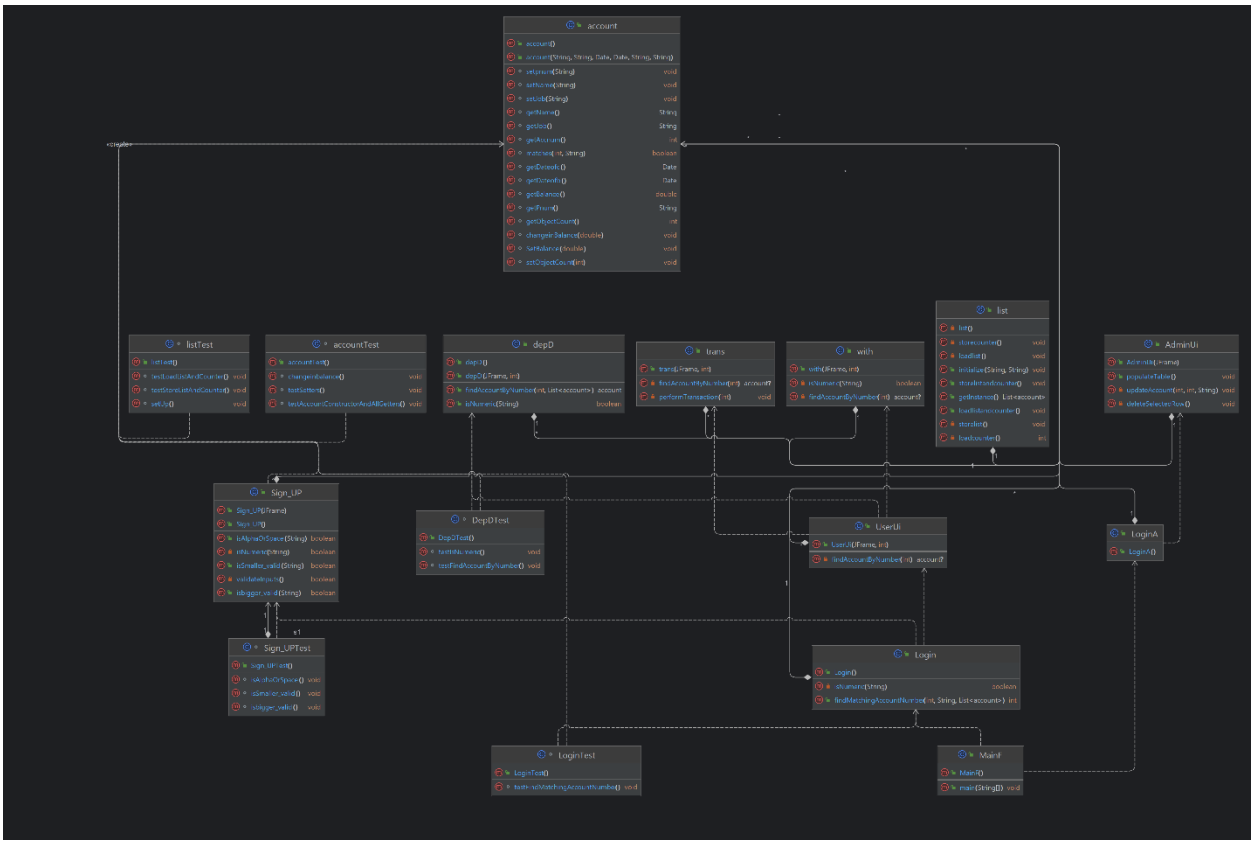
BANKING SYSTEM

1. Intro:

The world is full of banks, and these banks need a system that manages the accounts within them. The primary objective of this project is to create a basic banking management system using java. There will be two main user types in our program: Clients, Admins, both with different tasks and privileges.

2. Class Overview:

2.1) Class diagram:



I used IntelliJ IDEA and its UML tools to create the class diagrams for this project. This helped in clearly outlining the system’s structure and ensured that the design was consistent with coding standards.

2.2) Classes purpose and brief explanation:

1) MainF: Serves as the project's main class. It contains a JMenuBar for users to choose between the user and admin panels and handles the serialization process.

2) account: Responsible for creating account objects, a key element in the project.

3) List: Manages a collection of all account objects and includes necessary methods for serialization.

4) Login: this class manages the login system for the user panel.

5) Sign_Up: Oversees the creation of new accounts. It prompts users for their details and password, creates their account using the Account class, and provides them with an account number.

6) LoginA : Manages the login process for the admin panel.

7) AdminUi : this class manages the admin user interface, by providing a Jtable that stores all the accounts where specific fields can be edited like name, phone number, and it allows the deletion of accounts by selecting them and clicking a delete button.

8) UserUI: Manages the user interface post-login, offering options such as depositing, conducting transactions, checking balances, and withdrawing funds.

9) DepD: Handles the deposit operations within the system.

10) trans: Oversees all transaction-related activities.

11) with: Responsible for managing the withdrawal processes.

12) accountTest: Conducts tests on the Account class, focusing on its getters, setters, and key methods.

13) listTest: Tests the serialization processes, specifically the storing and loading methods in the project.

14) LoginTest: Evaluates essential functions of the login system to ensure reliability.

15) Sign_UPTest: Assesses the functionality of the sign-up methods to verify they work as intended.

16)DepDTest: Tests critical methods used throughout the project, ensuring their correct operation.

3) Requirements compliance overview:

1) Swing-Based GUI: Most classes in this project utilize Swing components for the graphical user interface. Notably, the AdminUI class features a JTable displaying all account details, and the MainF class includes a JMenuBar that lets users choose between the user and admin panels.

2) Java Collections Framework: The List class employs an ArrayList to efficiently store account objects, leveraging the robust functionality of the Java Collections Framework.

3) Java Serialization for File I/O: The List class is equipped with methods for serialization, which are utilized by the MainF class to ensure proper data storage and retrieval.

4) Unit Testing: Various dedicated test classes have been created to rigorously test key methods and functionalities within the project, ensuring reliability and robustness.

4) Description of the unit tests:

I did 11 tests that test the most important functions of the code and they are the following:

1) testAccountConstructorAndAllGetters: This test validates the Account class constructor by creating an account and verifying each getter's functionality. It uses assertEquals to ensure that each getter returns the expected result, confirming that the constructor initializes the object correctly.

2)testSetters: This test involves creating an account and then setting values using the class's setter methods. It employs assertEquals to verify that each setter correctly updates the account's attributes, ensuring they function as intended.

3) changeinbalance : this tests a very important method that is used for deposit, withdraw and transaction by creating an account setting the balance and using the assert equals to check if it work currently at changing the balance with a negative number.

4) testIsNumeric: This test evaluates a crucial method used in multiple classes, which checks whether a given string consists solely of numbers. It uses assertTrue and assertFalse to confirm that the method correctly identifies numeric and non-numeric strings, ensuring it provides the desired output.

5) testFindAccountByNumber: This test assesses a vital method used in various classes. It involves creating two accounts, storing them in a list, and then verifying whether the method can accurately return the correct account when provided with an account number. This is done using assertEquals to ensure that the method reliably identifies and retrieves the specified account.

6) testStoreListAndCounter: This test is linked to test 7. It involves creating an account and then saving it to a file.

7) testLoadListAndCounter: This test utilizes the file created by test 6. It checks that the file is not empty and contains valid data by using assertNotNull and assertFalse.

8) testFindMatchingAccountNumber: Creates two accounts with unique details and adds them to a list.

Verifies the method successfully returns the correct account number when given valid credentials.

Confirms the method returns -1 (indicating no match) when either the account number is incorrect or the password doesn't match.

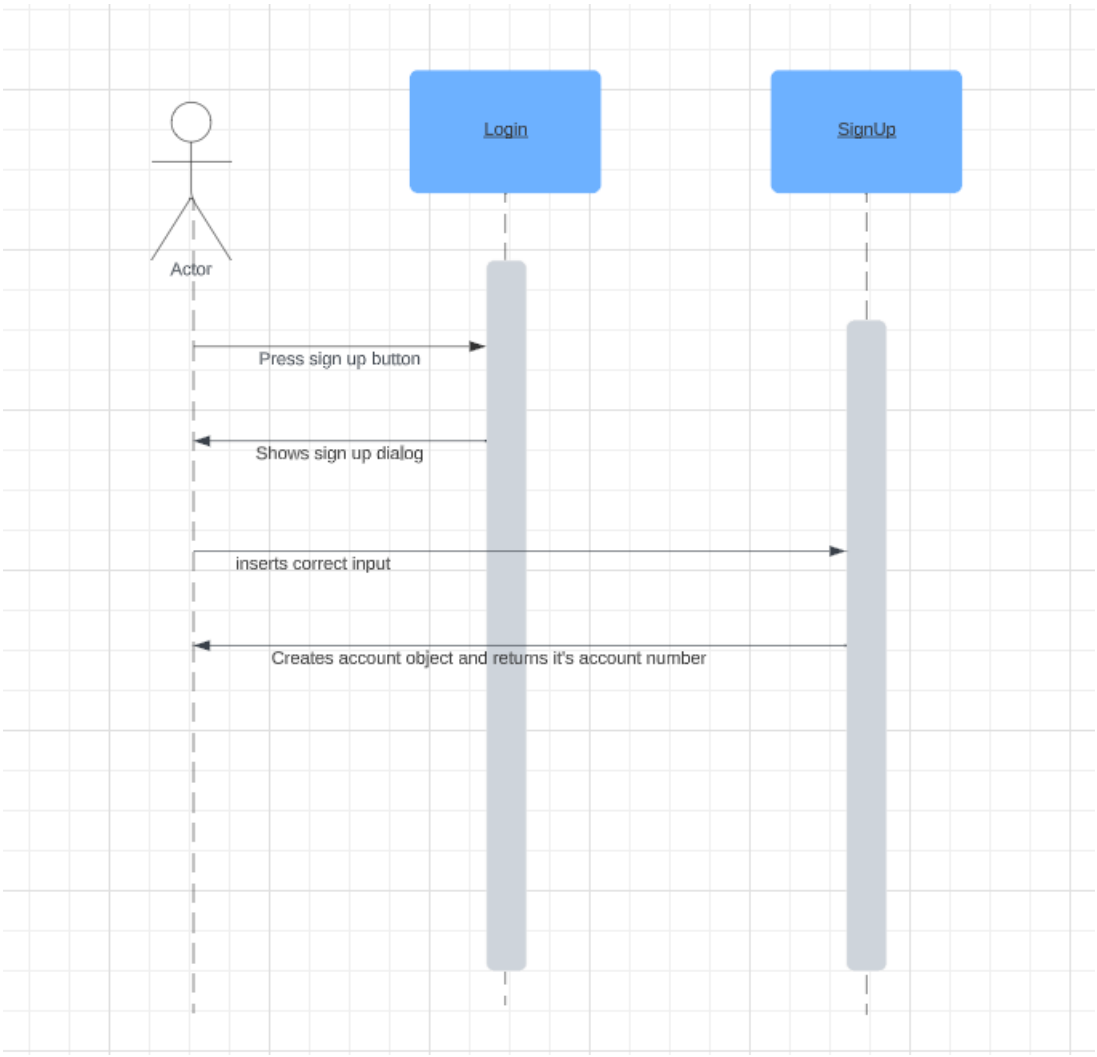
9) isAlphaOrSpace: test checks whether the input string is a valid name or professional title, allowing only alphabetic characters and spaces. It uses assertTrue to confirm that valid strings pass the test and assertFalse to ensure that strings with invalid characters are correctly identified as such. This test ensures the method accurately distinguishes between acceptable and unacceptable strings for names and professional titles.

10) isSmaller_valid: this test checks whether the date entered by the user is earlier than the current date. This test is primarily for input validation, ensuring that the date provided is in the past relative to the present moment. It's a crucial check for scenarios where future dates would be inappropriate or invalid.

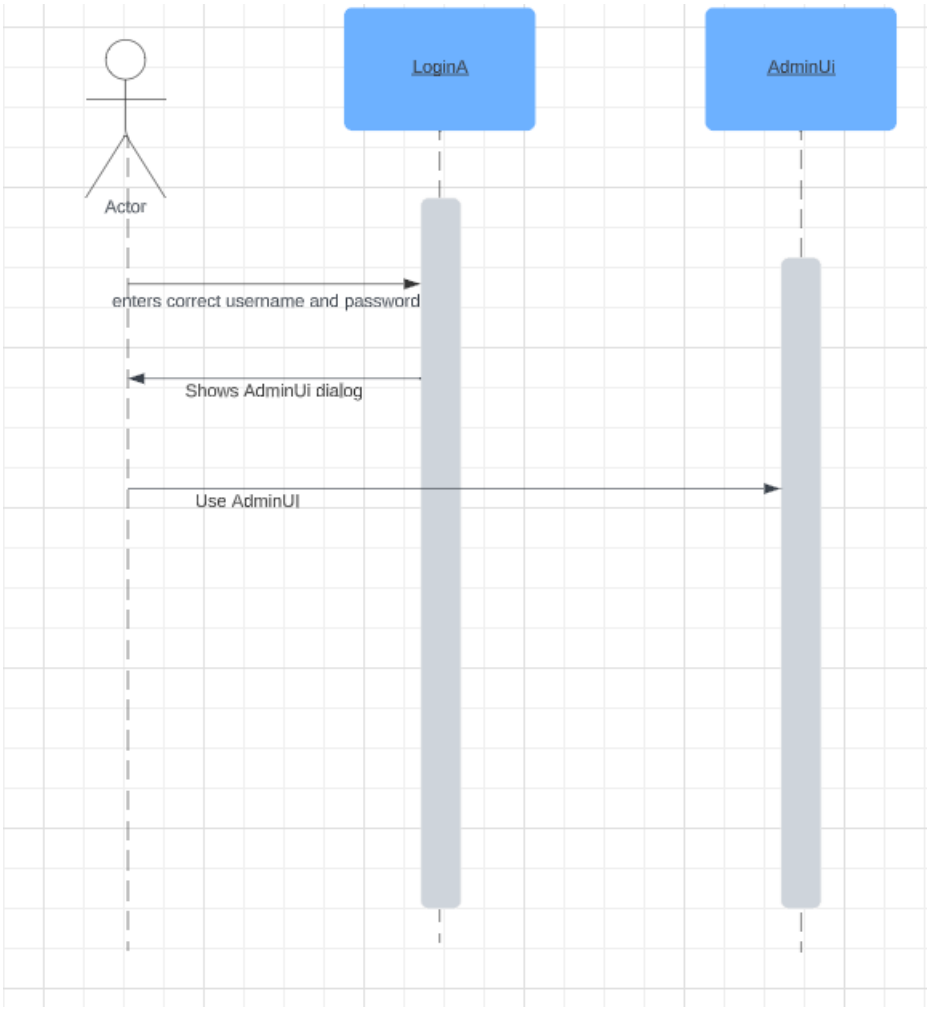
11) isbigger_valid: this test verifies that the date entered by the user is later than or equal to the current date. This test is essential for situations where only future or current dates are valid, such as setting appointment dates or deadlines. It ensures that the user input for dates is appropriately constrained to future or present dates.

6) Sequence Diagrams for Use-Cases:

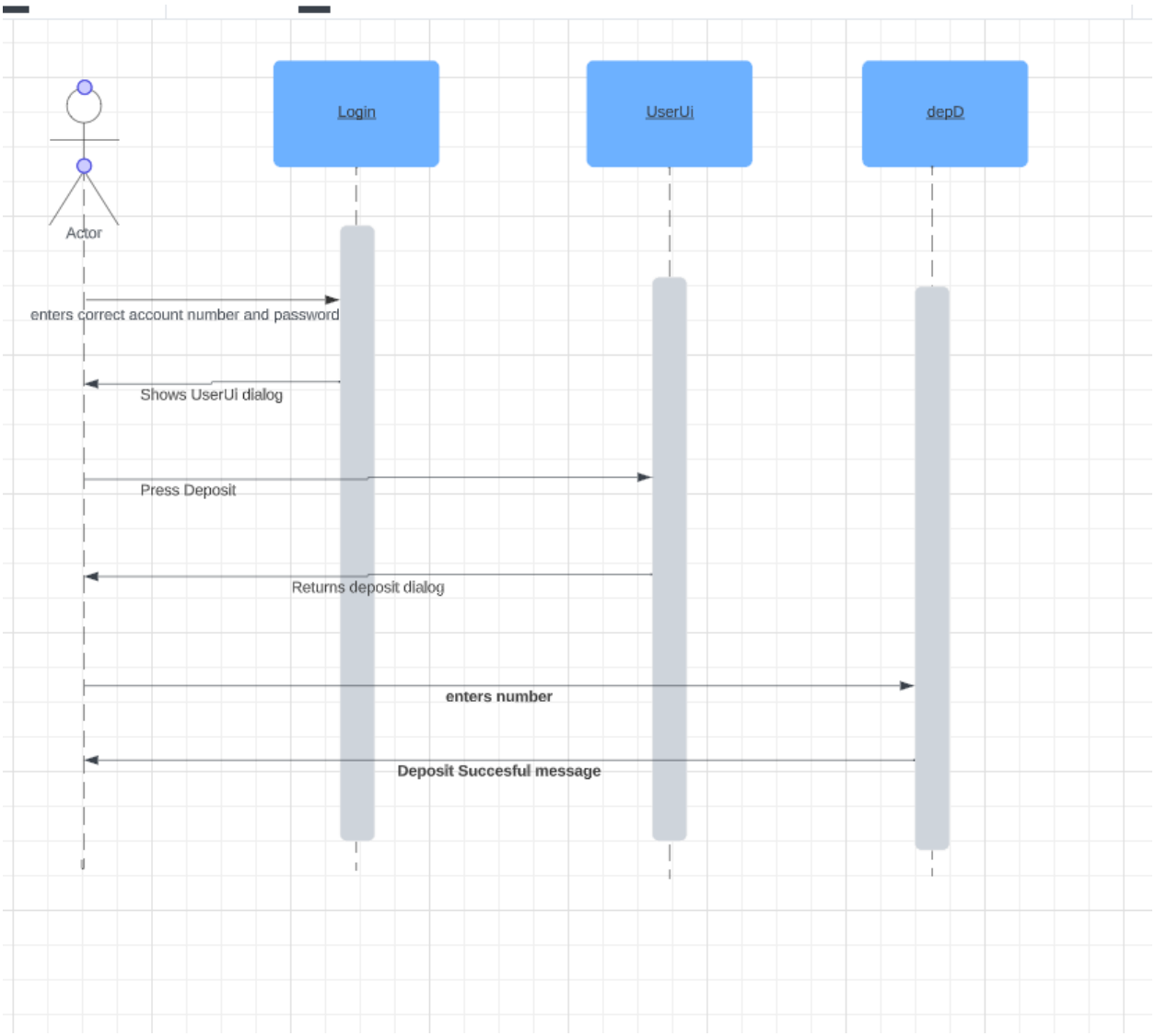
1) Creating a new account



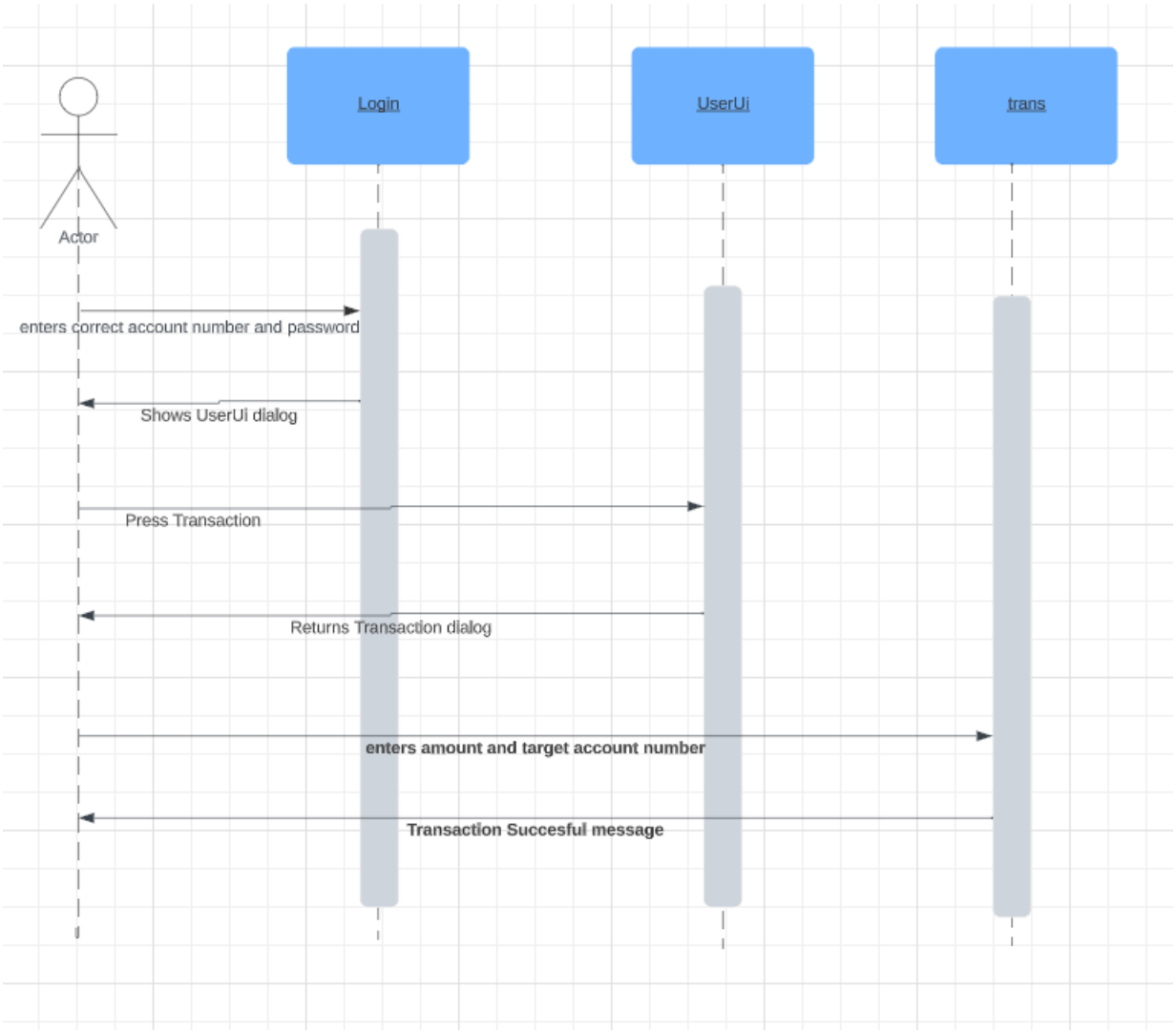
2) Admin login



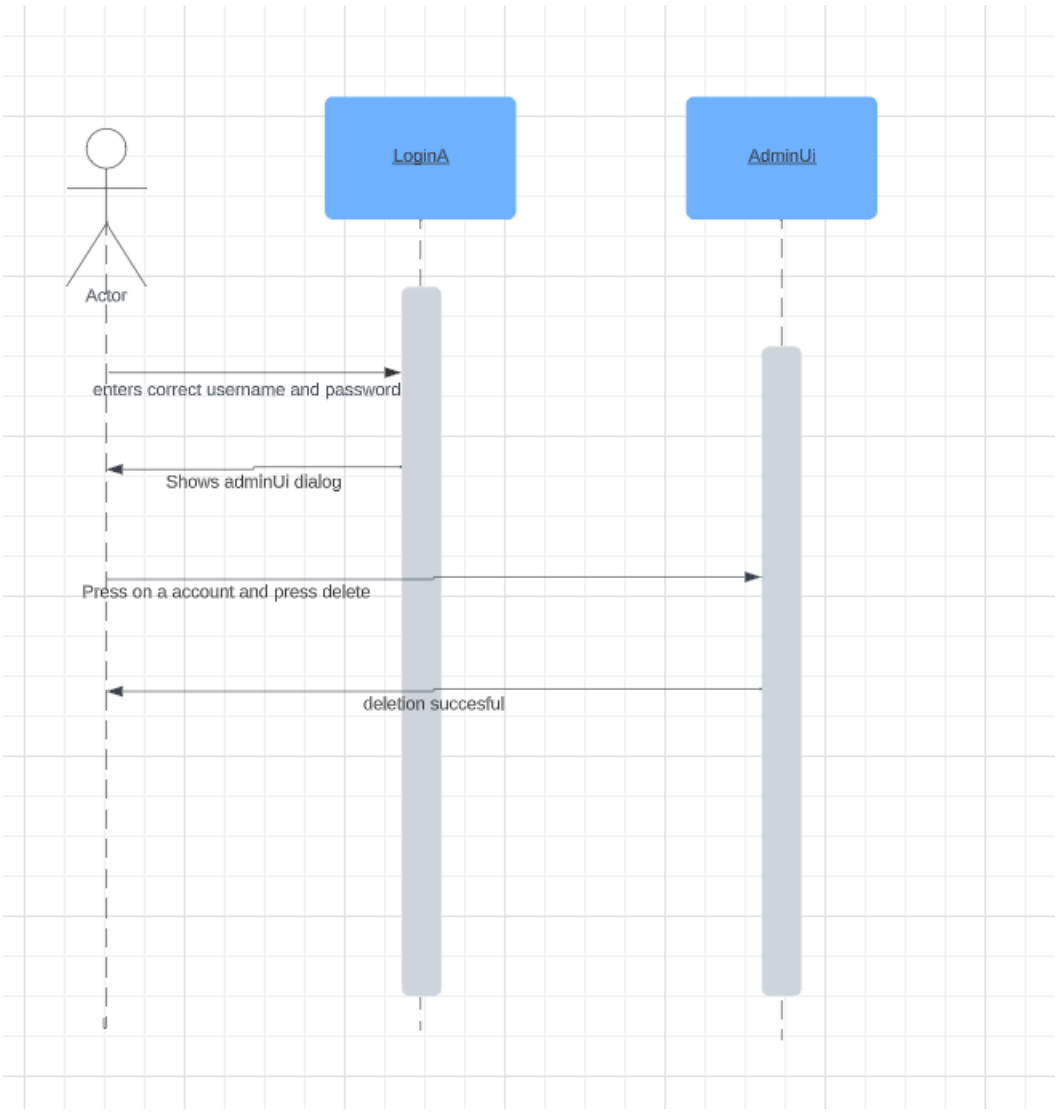
3) depositing money



4) Transaction of money



5) deleting an account


LoginA
AdminUi
Actor
enters correct username and password
Shows adminUi dialog
Press on a account and press delete
deletion succesful