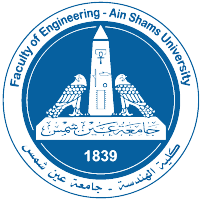


****

`

**Faculty of Engineering - Ain Shams University**

**CSE388 – Major Task (**Banking System**)**

**Submitted to:**

Dr. Islam El Maddah

Eng. Mahmoud Wagih

**Submitted by:**

Mohamed El Sayed 1901757

Jumana Yasser Mahrous 20P8421

Ahmed Hesham Mohamed 19P8052

**(3) Septeber2023**

Table of Contents

[ABSTRACT III](#_Toc144600422)

[1.0 INTRODUCTION: 4](#_Toc144600423)

[2.0 UNIT-TESTING: 5](#_Toc144600424)

[3.0 MAIN FLOW STATE DIAGRAM: 17](#_Toc144600425)

[4.0 GUI-TESTING: 18](#_Toc144600426)

[5.0 INTEGERATION TESTING: 23](#_Toc144600427)

[6.0 DEBUGGING USING TEST CASES: 25](#_Toc144600428)

[7.0 CONCLUTION: 26](#_Toc144600429)

## ABSTRACT

#### This report details the software testing project for an online banking system. The report covers the characteristics of a standard banking application, the challenges in testing the banking domain and their mitigation, and the importance of test reports.

#### The report also explains the components of a test report, including the purpose, overview of the product in test, test objective, and test summary. Additionally, the report provides a sample test report template and tips on how to write an effective test summary report. Overall, this report serves as a guide for testing an online banking system and creating a comprehensive test summary report.

## INTRODUCTION:

Software testing is an essential part of the software development process that ensures the quality, reliability, and security of the software product. Testing an online banking system is crucial as it involves sensitive financial information and transactions. A test summary report is a critical deliverable that provides insights into the overall health of the testing cycle and the application being tested.

This report captures various details and activities about the testing performed for the project and is shared with stakeholders like senior management, clients, etc. This report serves as a guide for creating an effective test summary report for an online banking system.

## UNIT-TESTING:

1. DBManegment Class:

**🡪Attributes:**

* **url:** a private final string attribute that stores the URL of the database.
* **username:** a private final string attribute that stores the username of the database.
* **password:** a private final string attribute that stores the password of the database.
* **connection:** a private Connection attribute that stores the connection to the database.

**🡪Methods:**

* **DBMangment ():** a constructor that establishes a connection to the database and prints the contents of the Login table.
* **signUp (String username, String password, String fullname, String nationalID, String phoneNo):** a method that takes in five parameters and inserts a new row into the Login table if the parameters are valid and the username, national ID, and phone number do not already exist in the table. Returns 1 if successful, -1 otherwise.
* **exists (String username, String nationalID, String phoneNo):** a method that takes in three parameters and checks if the username, national ID, or phone number already exist in the Login table. Returns 1 if any of them exist, -1 otherwise.
* **logIn (String username, String password)**: a method that takes in two parameters and checks if the username and password match a row in the Login table. Returns 1 if successful, -1 otherwise.
* **updateBalanceWithAmount (String username, double amount):** a method that takes in two parameters and updates the balance of the user with the given username by adding the amount to the current balance. Returns the final balance if successful, -1 otherwise.
* **retrieveBalance (String username):** a method that takes in one parameter and retrieves the balance of the user with the given username from the Login table. Returns the balance if successful, -1 otherwise.
* **updateBalance(String username, double updatedBalance):** a method that takes in two parameters and updates the balance of the user with the given username to the updated balance in the Login table. Returns the balance if successful, -1 otherwise.
* **recordTransaction (String username, String transactionType, double amount):** a method that takes in three parameters and inserts a new row into the transactions table with the given username, transaction type, and amount. Returns 1 if successful, -1 otherwise.
* **withdraw (String username, Double amount):** a method that takes in two parameters and withdraws the given amount from the balance of the user with the given username in the Login table. Returns the final balance if successful, -1 otherwise.
* **transferMoney (String fromUsername, String toUsername, Double amount):** a method that takes in three parameters and transfers the given amount from the balance of the user with the fromUsername to the balance of the user with the toUsername in the Login table. Returns 1 if successful, -1 otherwise.
* **usernameExists (String username):** a method that takes in one parameter and checks if the username exists in the Login table. Returns 1 if it exists, -1 otherwise.
* **deposit (String username, double amount):** a method that takes in two parameters and deposits the given amount to the balance of the user with the given username in the Login table. Returns the final balance if successful, -1 otherwise.
* **retrieveAndDisplayTransactions(String username):** a method that takes in one parameter and retrieves all the transactions of the user with the given username from the transactions table. Returns 1 if successful, -1 otherwise.
* **delete\_user (String username):** a method that takes in one parameter and deletes the user with the given username from the Login table if it exists. Does not return anything.

1. DBMangment\_Test:

**🡪Attributes:**

* **db:** a static DBMangment attribute that is used to create an instance of the DBMangment class.
* **username:** a static string attribute that is used to store the username for testing.
* output: a double attribute that is used to store the output of the methods being tested.
* **amount:** a double attribute that is used to store the amount for testing.
* **expected:** a double attribute that is used to store the expected output of the methods being tested.

**🡪 Methods:**

* **retrieveBalance\_Test ():** a method that tests the retrieveBalance method in the DBMangment class by updating the balance of the user with the username "1" to 100 and retrieving the balance. The method then uses the Assert.assertEquals method to compare the output with the expected value of 100.
* **updateBalanceWithAmount\_Test ():** a method that tests the updateBalanceWithAmount method in the DBMangment class by updating the balance of the user with the username "1" by adding 100 to the current balance. The method then uses the Assert.assertEquals method to compare the output with the expected value of the updated balance.
* **updateBalance\_Test():** a method that tests the updateBalance method in the DBMangment class by updating the balance of the user with the username "1" to 300. The method then uses the Assert.assertEquals method to compare the output with the expected value of the updated balance.
* **recordTransaction\_Test():** a method that tests the recordTransaction method in the DBMangment class by recording a deposit transaction of 100 for the user with the username "1". The method then uses the Assert.assertEquals method to compare the output with the expected value of 1.
* **retrieveAndDisplayTransactions\_Test ():** a method that tests the retrieveAndDisplayTransactions method in the DBMangment class by retrieving all the transactions of the user with the username "1" and adding them to a Transactions object. The method then uses the Assert.assertEquals method to compare the output with the expected value of 1.

1. Deposit\_Tests:

**🡪Attributes:**

* **db:** a static DBMangment attribute that is used to create an instance of the DBMangment class.
* **username:** a static string attribute that is used to store the username for testing.
* **output:** a double attribute that is used to store the output of the methods being tested.
* **before:** a double attribute that is used to store the balance of the user before the deposit.
* **amount:** a double attribute that is used to store the amount for testing.

**🡪Methods:**

* **accepted\_case\_test():** the deposit method in the DBMangment class by depositing 100.50 for the user with the username "1". The method then uses the Assert.assertEquals method to compare the output with the expected value of the balance before the deposit plus the amount deposited.
* **negative\_entry():** the deposit method in the DBMangment class by depositing -100 for the user with the username "1". The method then uses the Assert.assertEquals method to compare the output with the expected value of -1.
* **zero\_entry():** deposit method in the DBMangment class by depositing 0 for the user with the username "1". The method then uses the Assert.assertEquals method to compare the output with the expected value of -1.

1. Login\_Tests:

**🡪 Attributes:**

* **db:** a static DBMangment attribute that is used to create an instance of the DBMangment class.
* **username:** a static string attribute that is used to store the username for testing.
* **password:** a static string attribute that is used to store the password for testing.
* **fullname:** a static string attribute that is used to store the fullname for testing.
* **nationalID:** a static string attribute that is used to store the national ID for testing.
* **phoneNo:** a static string attribute that is used to store the phone number for testing.
* **output:** an integer attribute that is used to store the output of the methods being tested.

**🡪Methods:**

* **accepted\_case\_test():** tests the logIn method with a correct username and password. It asserts that the output of the method is 1, indicating a successful login.
* **incorrect\_username():** tests the logIn method with an incorrect username. It asserts that the output of the method is -1, indicating a failed login.
* **incorrect\_password():** the logIn method with an incorrect password. It asserts that the output of the method is -1, indicating a failed login.
* **incorrect\_username\_and\_password():** tests the logIn method with both an incorrect username and password. It asserts that the output of the method is -1, indicating a failed login.

1. SignUp\_Tests:

**🡪 Class Members:**

* **db:** An instance of the DBMangment class used for interacting with the database.
* **username:** Holds the username value for the test cases.
* **password:** Holds the password value for the test cases.
* **fullname:** Holds the fullname value for the test cases.
* **nationalID:** Holds the nationalID value for the test cases.
* **phoneNo:** Holds the phoneNo value for the test cases.
* **output:** Holds the output value returned by the signup method for the test cases.

**🡪Test Cases:**

* **accepted\_case\_test():** This test case verifies the behaviour of the signup method when provided with valid input values. It sets the username, password, fullname, nationalID, and phone No variables to valid values and calls the signup method. The expected output is 1, indicating a successful sign-up operation. The `Assert.assertEquals` method is used to compare the expected and actual output values.
* **empty\_username():** This test case checks the handling of an empty username value. It sets the username variable to an empty string, while keeping other input values valid. The signUp method is called, and the expected output is -1, indicating a failure due to an empty username.
* **empty\_password():** This test case verifies the behaviour when the password value is empty. Similar to the previous test case, it sets the password variable to an empty string and expects a return value of -1, indicating a failure due to an empty password.
* **empty\_fullname():** This test case tests the handling of an empty fullname value. It sets the fullname variable to an empty string and expects a return value of -1, indicating a failure due to an empty fullname.
* **empty\_nationalID ():** This test case checks the behavior when the nationalID value is empty. It sets the nationalID variable to an empty string and expects a return value of -1, indicating a failure due to an empty nationalID.
* **empty\_phoneNo():** This test case verifies the handling of an empty phoneNo value. It sets the phoneNo variable to an empty string and expects a return value of -1, indicating a failure due to an empty phone No.
* **used\_username():** This test case tests the behaviour when an already used username is provided. It first signs up a user with a specific username and then attempts to sign up another user with the same username. The expected output is -1, indicating a failure due to a duplicate username.
* **used\_nationalID():** This test case checks the handling of an already used nationalID value. It first signs up a user with a specific nationalID and then attempts to sign up another user with the same nationalID. The expected output is -1, indicating a failure due to a duplicate nationalID.
* **used\_phoneNo():** This test case verifies the behaviour when an already used phone No is provided. It first signs up a user with a specific phoneNo and then attempts to sign up another user with the same phone No. The expected output is -1, indicating a failure due to a duplicate phone No.

**🡪 Test Execution Flow:**

* The Setup method is executed once before running any test cases. It is responsible for setting up the testing environment, in this case, by deleting specific users from the database.
* Each test case is executed independently and in no specific order.

1. Transfer Money\_Tests:

**🡪Class Members:**

* **fromUsername:** Holds the username of the sender for the test cases.
* **toUsername:** Holds the username of the recipient for the test cases.
* **db:** An instance of the DBMangment class used for interacting with the database.
* **amount:** Holds the amount of money for the test cases.
* **output:** Holds the output value returned by the transfer Money method for the test cases.

**🡪 Test Cases:**

* **accepted\_case\_test()**: This test case verifies the behavior of the transferMoney method when a valid transfer is made. It sets the sender, recipient, and amount variables to valid values and calls the transferMoney method. It then compares the balances of the sender and recipient before and after the transfer to ensure they have been updated correctly.
* **insufficient\_balance():** This test case checks the handling of a transfer with an amount greater than the sender's balance. It sets the sender, recipient, and amount variables accordingly and expects a return value of -1, indicating a failure due to insufficient balance.
* **negative\_entry():** This test case verifies the behavior when a negative amount is provided for the transfer. It sets the sender, recipient, and amount variables accordingly and expects a return value of -1, indicating a failure due to a negative amount.
* **zero\_entry():** This test case checks the handling of a transfer with a zero amount. It sets the sender, recipient, and amount variables accordingly and expects a return value of -1, indicating a failure due to a zero amount.
* **username\_not\_found():** This test case tests the behavior when the recipient username is not found in the database. It sets the sender, recipient, and amount variables accordingly and expects a return value of -1, indicating a failure due to the recipient username not being found.

1. Withdraw\_Tests:

**🡪Class Members:**

* **db:** An instance of the DBMangment class used for interacting with the database.
* **username:** Holds the username of the user for the test cases.
* **output:** Holds the output value returned by the withdraw method for the test cases.
* **before:** Holds the balance of the user's account before the withdrawal.
* **amount:** Holds the amount of money to be withdrawn for the test cases.

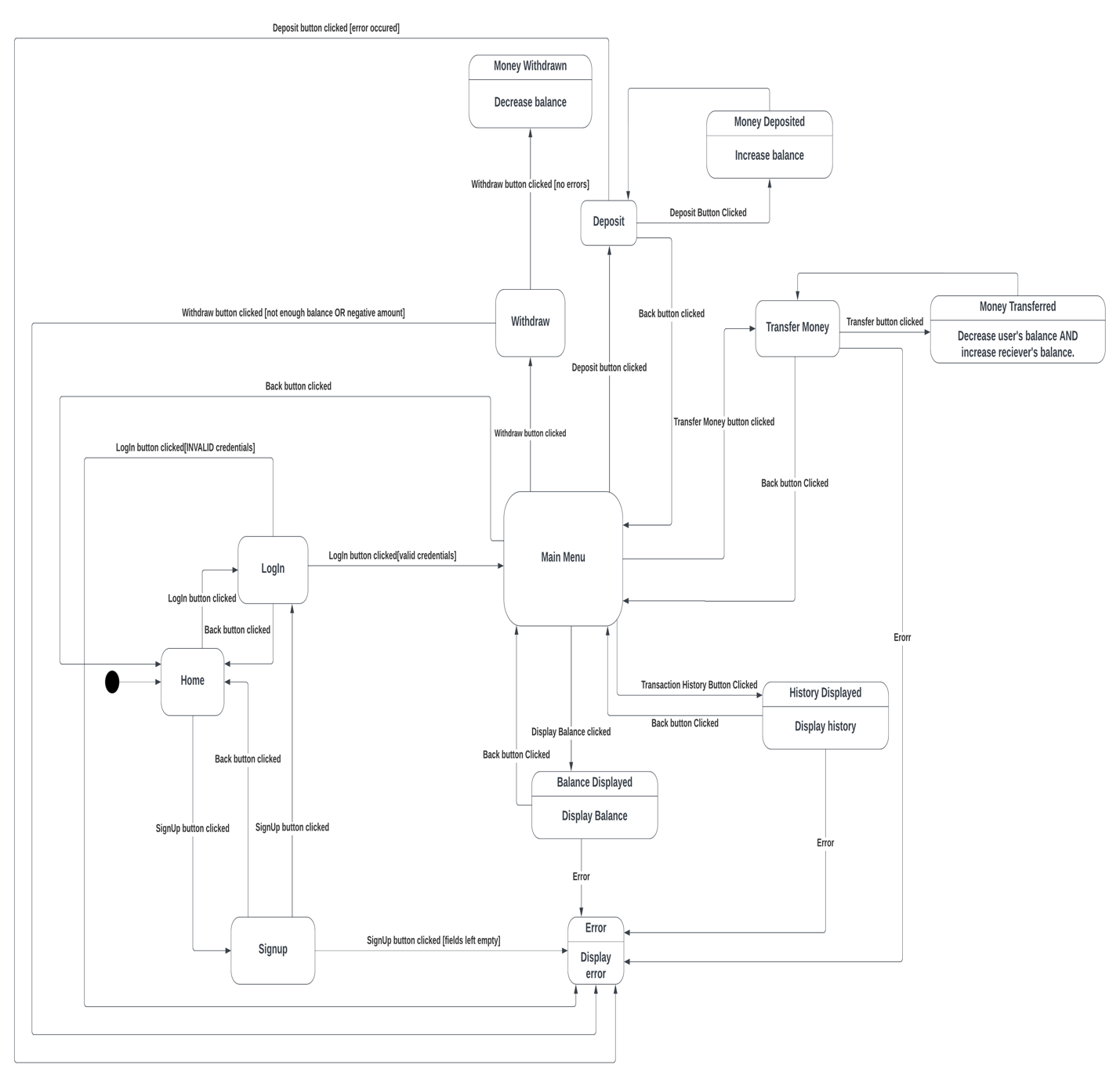
**🡪Test Cases:**

* **accepted\_case\_test ():** This test case verifies the behavior of the withdraw method when a valid withdrawal is made. It sets the amount variable to a valid value and calls the `withdraw` method. It then compares the balance of the user's account before and after the withdrawal to ensure it has been updated correctly.
* **negative\_entry():** This test case checks the handling of a withdrawal with a negative amount. It sets the amount variable accordingly and expects a return value of -1, indicating a failure due to a negative amount.
* **zero\_entry():** This test case verifies the behavior when a withdrawal with a zero amount is attempted. It sets the amount variable accordingly and expects a return value of -1, indicating a failure due to a zero amount.
* **insufficient\_amount ():** This test case tests the handling of a withdrawal with an amount greater than the user's balance. It sets the amount variable accordingly and expects a return value of -1, indicating a failure due to insufficient funds.

## MAIN FLOW STATE DIAGRAM:

**Diagrams Link:**

**(**[**https://drive.google.com/drive/folders/1mP12h51WuxadL9XGw9m0C9ktldvZ9FN-?usp=share\_link**](https://drive.google.com/drive/folders/1mP12h51WuxadL9XGw9m0C9ktldvZ9FN-?usp=share_link)**)**



## GUI-DIAGRAMS:

**Diagrams Link:**

**(**[**https://drive.google.com/drive/folders/1mP12h51WuxadL9XGw9m0C9ktldvZ9FN-?usp=share\_link**](https://drive.google.com/drive/folders/1mP12h51WuxadL9XGw9m0C9ktldvZ9FN-?usp=share_link)**)**

1. **Start-Home Page**

A diagram of a login button

Description automatically generated

1. **Signup**

A blurry image of a diagram

Description automatically generated

1. **Login**

A diagram of a company

Description automatically generated

1. **Main Menu**

A diagram of a company

Description automatically generated

1. A diagram of a company

   Description automatically generated**Deposit**
2. **Withdraw**

A diagram of a company

Description automatically generated

1. **Transfer Money**

A diagram of a company

Description automatically generated

1. **Show Balance**

A diagram of a diagram

Description automatically generated

1. **History of Transactions**

A diagram of a computer

Description automatically generated

1. **Logout**

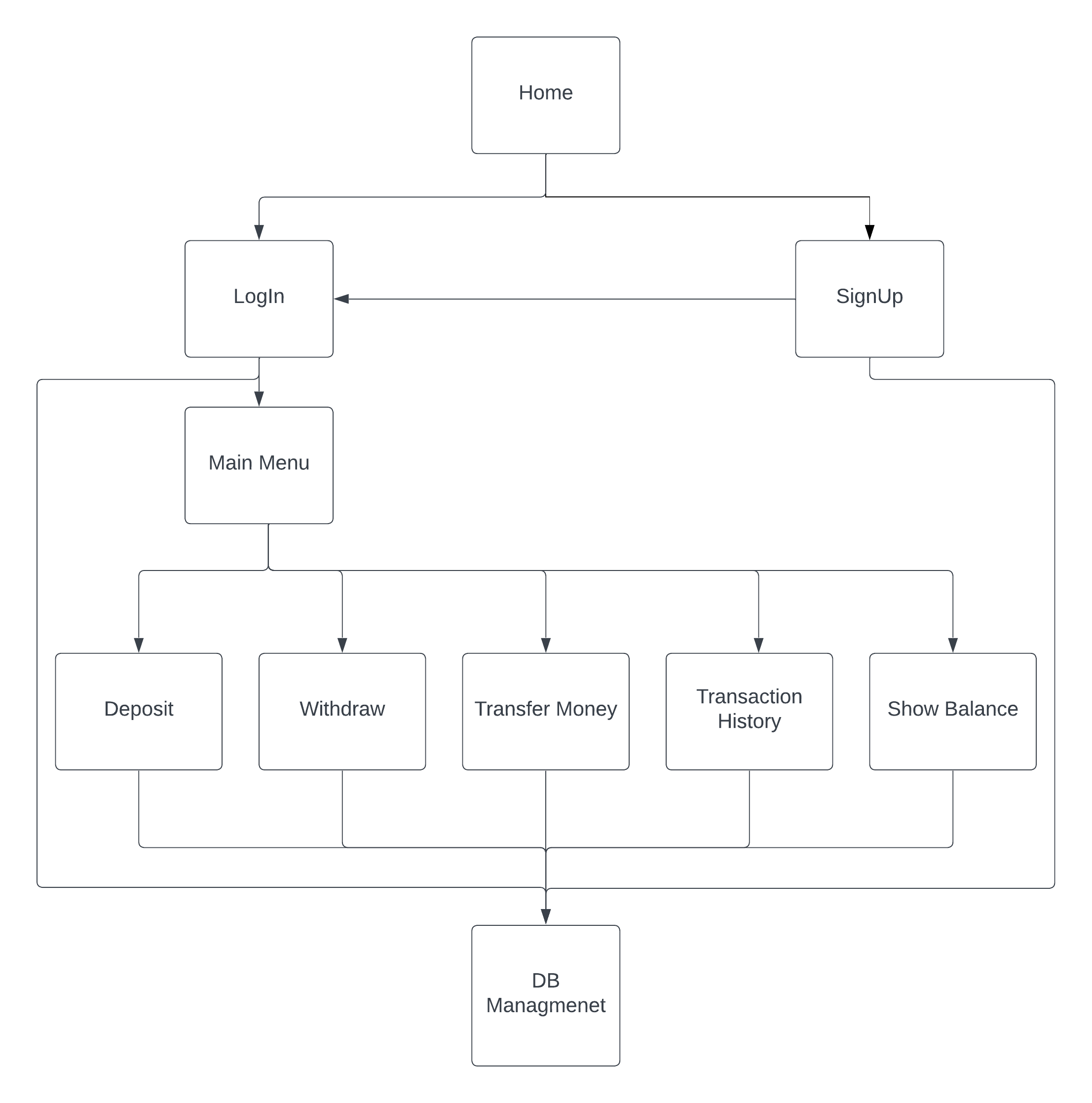
A diagram of a website

Description automatically generated

## INTEGERATION TESTING:

**Diagrams Link:**

**(**[**https://drive.google.com/drive/folders/1mP12h51WuxadL9XGw9m0C9ktldvZ9FN-?usp=share\_link**](https://drive.google.com/drive/folders/1mP12h51WuxadL9XGw9m0C9ktldvZ9FN-?usp=share_link)**)**



1. Test the **home module** with login and sign up as stubs.
2. Test the **DB Management** module with deposit, withdraw, transfer money, transaction history, show balance, login, and sign up as drivers.
3. Test the **home, login, and DB Management** modules with the sign up and main menu as stubs.
4. **Test home, login, DB Management and signup** modules with main menu as stub, Deposit, withdraw, transfer money, transaction history and show balance as drivers.
5. Test **main menu, login, dB, home** with stubs for deposit, Withdraw, Transfer Money, Transaction History and Show Balance
6. Test the **home, login, signup, main menu, DB Management, and deposit** with withdraw, transfer money, transaction history, and show balance as both subs and drivers.
7. Test **home, login, DB Management, signup, main menu and deposit** with withdraw, transfer money, transaction history, show balance as stubs and drivers.
8. Test **main menu, Login, Signup, DB Management, Home, Deposit, Withdraw, Transfer Money with both stubs** and drivers for Transaction, Show Balance.
9. Test **home, login, DB Management, signup, main menu, deposit, withdraw, transfer money and transaction history** with show balance as a stub and driver.
10. Test the **whole program.**

## DEBUGGING USING TEST CASES:

* **User transferring money to himself.**
* **Depositing, withdrawing, or transferring a negative amount.**
* **Depositing, withdrawing, or transferring amount = 0.**
* **Transferring to an account that doesn't exist.**
* **Leaving the username and password empty while logging in.**
* **Leaving any fields empty in any page.**
* **Checking if a username, phone number, or national ID already exist before signing up.**
* **Changing the data types of the phone number and national ID from int to String.**

## CONCLUSION:

The banking system project has been thoroughly tested using JUnit 4, ensuring that all the functionalities of the system are working as expected. The tests covered various scenarios, including accepted cases, empty fields, and duplicate entries. The tests have been successful in identifying any issues and ensuring that the system is functioning correctly. The use of JUnit 4 has provided a reliable and efficient way to test the system, allowing for quick identification and resolution of any issues. Overall, the project has been successfully tested, and the banking system is ready for deployment.

A qr code on a green background

Description automatically generated