Designing and Executing Tests for a Banking System

Submitted to:

Dr. Islam Ahmed Mahmoud El maddah

Eng: Eman Khaled Ahmed Ibrahuim

Submitted by:

Nagy Ahmed Nagy	23p0365
Abdelrhman Mohammed Mahmoud Salah	23p0370
Ahmed Mosta Gomaa	23p0375

Contents

1. Introduction 2
2. System Overview 2
2.1 System Architecture 2
2.2 Core Functionalities 2
3. Account States and Lifecycle 3
Section A: Black-Box Testing 4
A.1 Objective 4
A.2 Techniques Used 4
A.3 Boundary and Partition Analysis 4
Section B: White-Box Testing 5
B.1 Objective 5
B.2 Code Under Test 5
B.3 Control Flow Graph (CFG) 5
B.4 Coverage Analysis 5
Section C: UI Testing 6
C.1 Objective 6
C.2 GUI Components Tested 6
Section D: State-Based Testing 8
D.1 Objective 8
D.2 State Transition Matrix 8
Section E: Test-Driven Development (TDD) 11
E.1 New Feature 11
E.2 TDD Process 11
4. Conclusion 11

Repo:https://github.com/abdelrhmanaj/Testing_pro

1. Introduction

This project aims to design, implement, and execute a comprehensive testing strategy for a simplified banking system. The system simulates real-world banking operations such as client account management, deposits, withdrawals, transfers, and administrative control through account state transitions.

The main objective of the project is to apply theoretical software testing concepts in a practical context. Throughout the project, different testing techniques are applied to validate system correctness, detect defects, and ensure that the system behaves according to its functional and non-functional requirements.

The project follows a structured testing lifecycle. First, the system requirements are analyzed. Then, black-box and white-box test cases are designed. After that, UI testing, state-based testing, and integration testing are performed. Finally, a new feature is implemented using the Test-Driven Development (TDD) approach to demonstrate modern testing practices.

2. System Overview

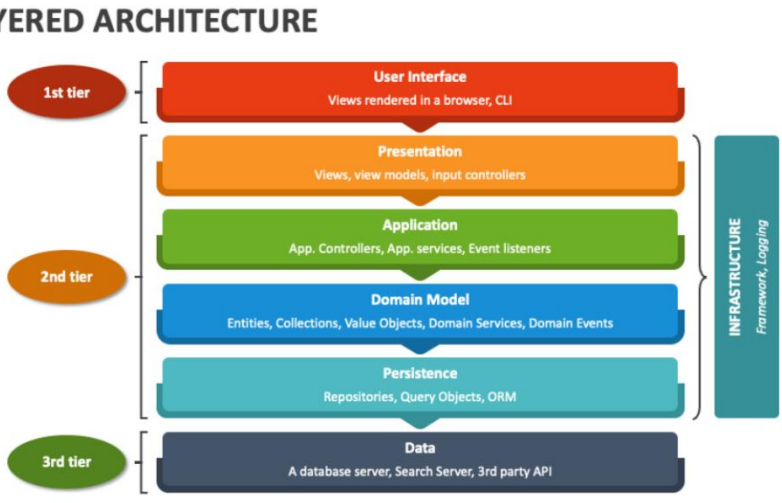
The banking system is organized into multiple logical layers to separate responsibilities and improve maintainability.

2.1 System Architecture

- **User Interface (UI):** HTML-based pages for login, dashboard, and transaction history.
- **Controller Layer:** Handles user requests and orchestrates operations.
- **Service Layer:** Contains business logic such as deposit, withdrawal, and validation.
- **Model Layer:** Represents the account entity and its internal state.

2.2 Core Functionalities

- Client profile viewing
- Deposit, withdrawal, and transfer operations
- Account state enforcement
- Administrative state transitions



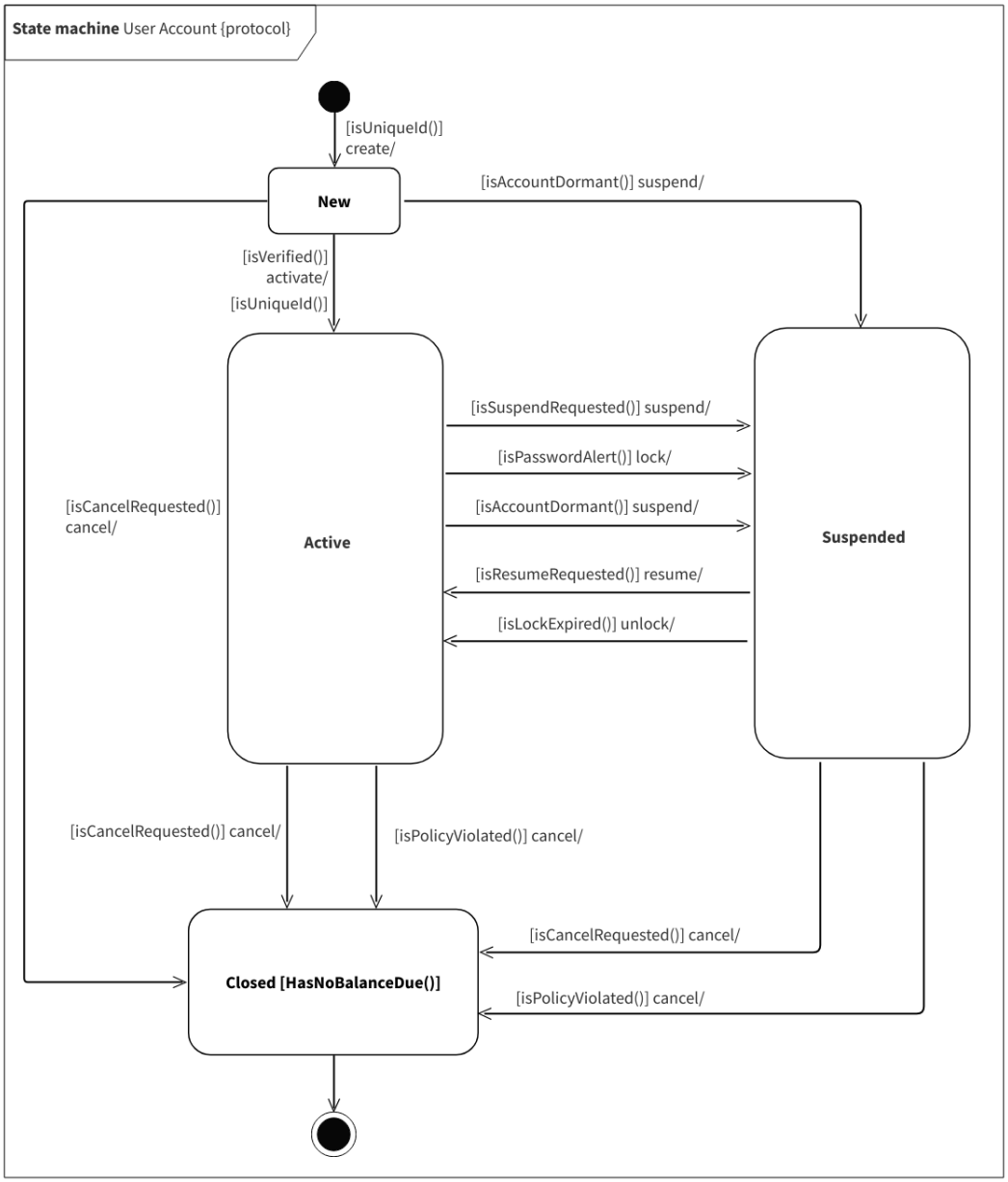
3. Account States and Lifecycle

The banking system uses a state-based model to control account behavior.

Supported States

- **Unverified**
- **Verified**
- **Suspended**
- **Closed**

Each state defines what actions are allowed or blocked. This prevents illegal operations and enforces business rules.



Section A: Black-Box Testing

A.1 Objective

The goal of black-box testing is to validate system functionality based on requirements without considering the internal implementation.

A.2 Techniques Used

- Equivalence Partitioning
- Boundary Value Analysis

A.3 Boundary and Partition Analysis

Equivalence Classes

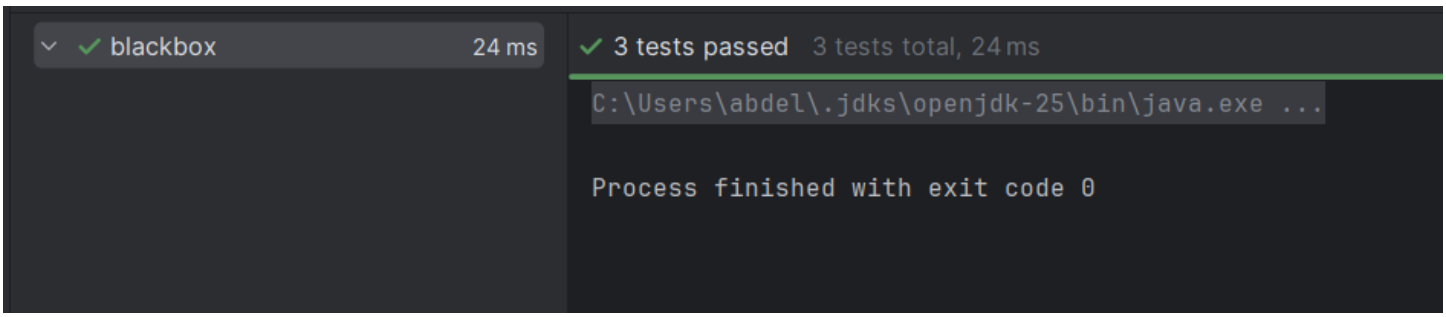
- Negative deposit values → Invalid
- Valid deposit values → Valid
- Deposit in Closed state → Invalid

Boundary Values

- Deposit amount = 0
- Withdraw amount = balance
- Withdraw amount > balance

Sample Black-Box Test Cases

Test Case ID	Input	Expected Output
BB01	deposit(-100)	false
BB02	withdraw(50)	true if balance ≥ 50
BB03	withdraw(500)	false



Section B: White-Box Testing

B.1 Objective

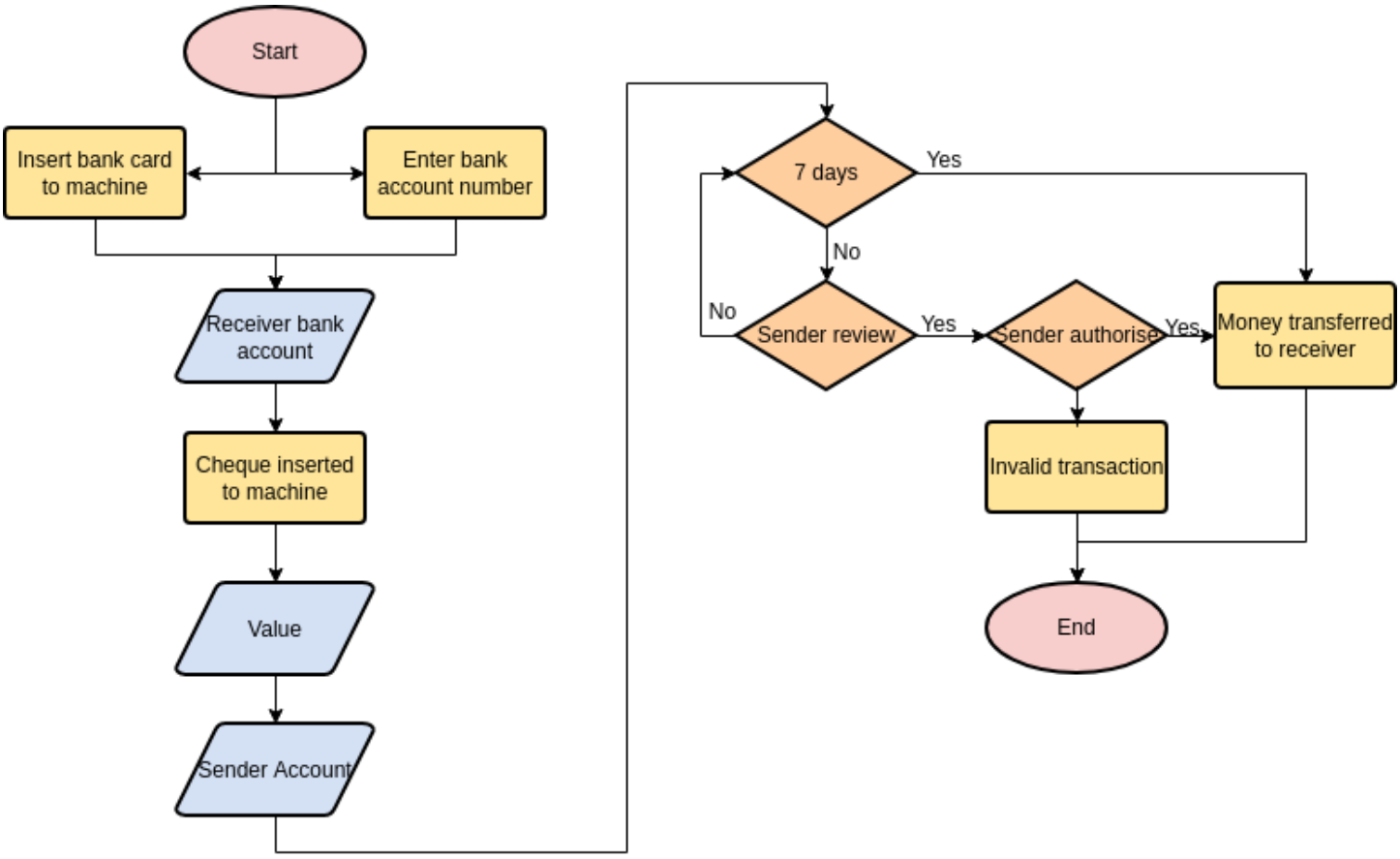
White-box testing focuses on verifying internal logic, decision paths, and branch execution.

B.2 Code Under Test

- TransactionProcessor
- deposit()
- withdraw()

B.3 Control Flow Graph (CFG)

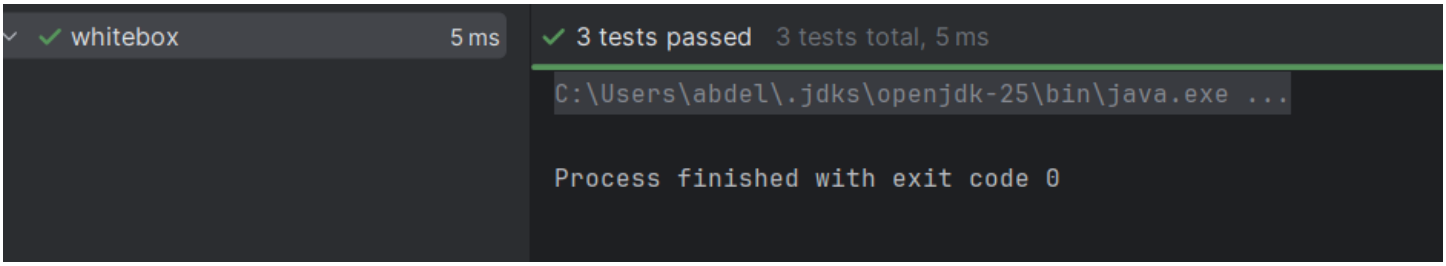
CFGs were created to identify all execution paths, decision points, and return statements.



B.4 Coverage Analysis

All branches were executed at least once, achieving:

- 100% statement coverage
- 100% branch coverage



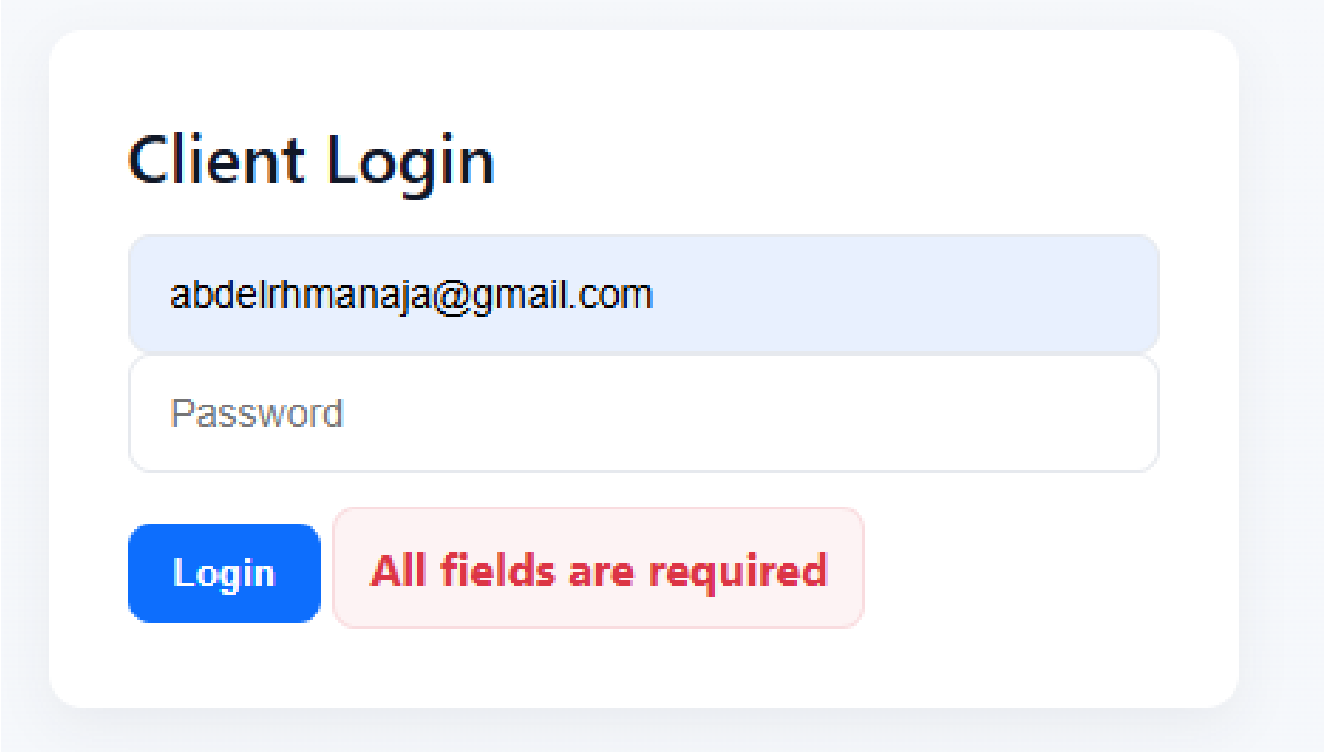
Section C: UI Testing

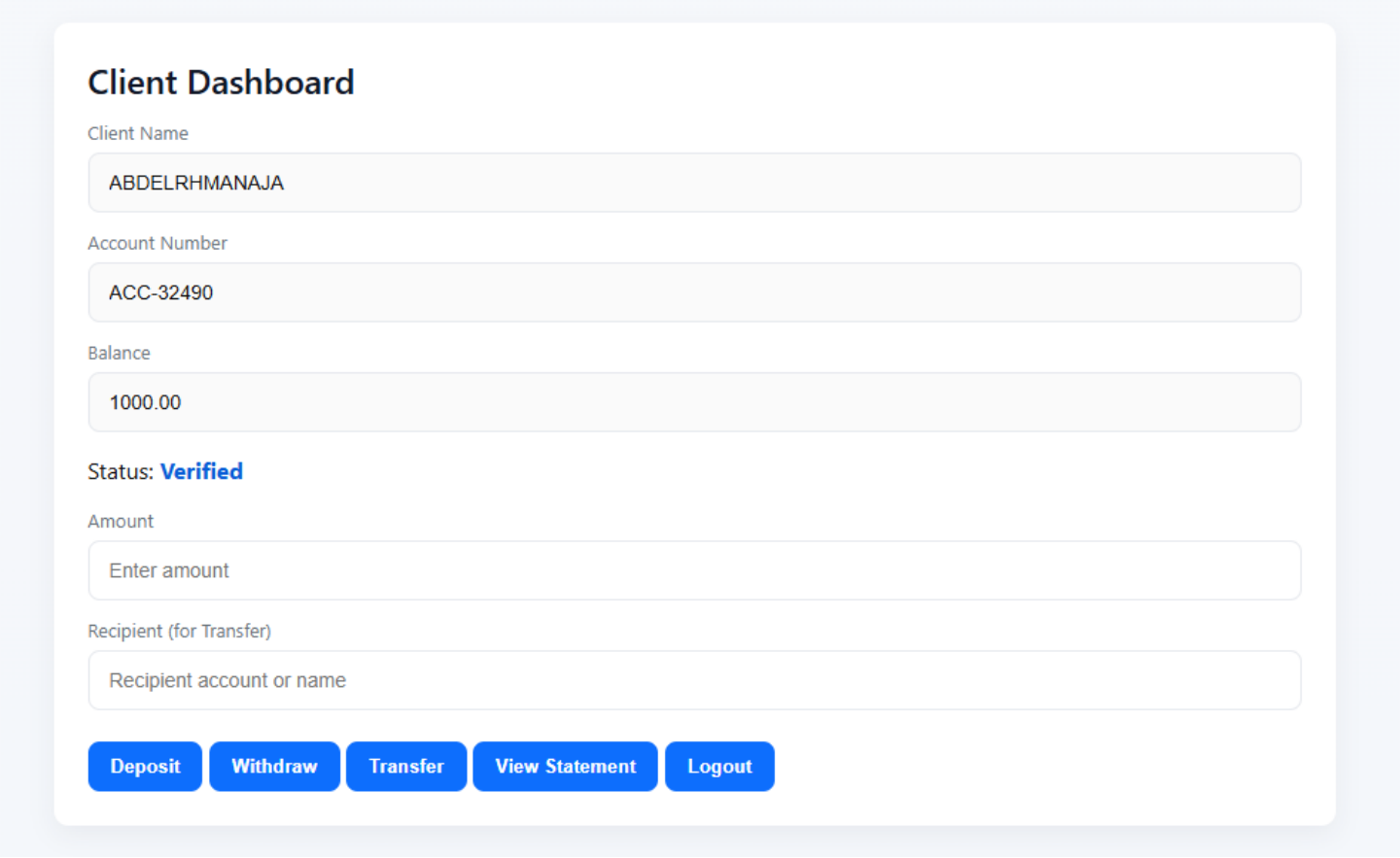
C.1 Objective

UI testing validates that the graphical interface behaves correctly and provides appropriate feedback to users.

C.2 GUI Components Tested

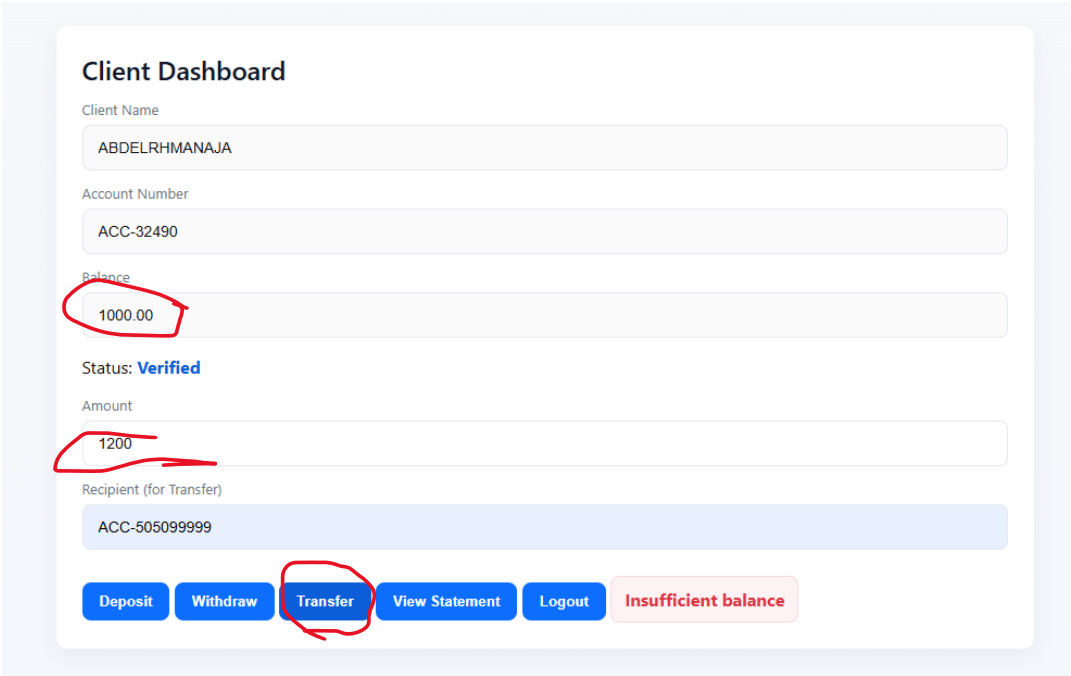
- Client name, account number, balance fields
- Deposit, Withdraw, Transfer, View Statement buttons
- Status label
- Notification message box

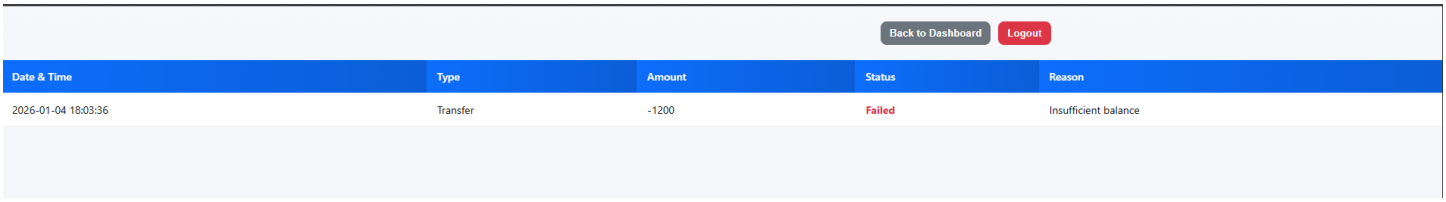




C.3 Functional UI Checks

- Proper rendering of account status
- Buttons enabled/disabled based on state
- Input validation for negative and empty values
- Error and success messages displayed correctly





UI testing was conducted manually using a structured checklist. Selenium automation was considered optional.

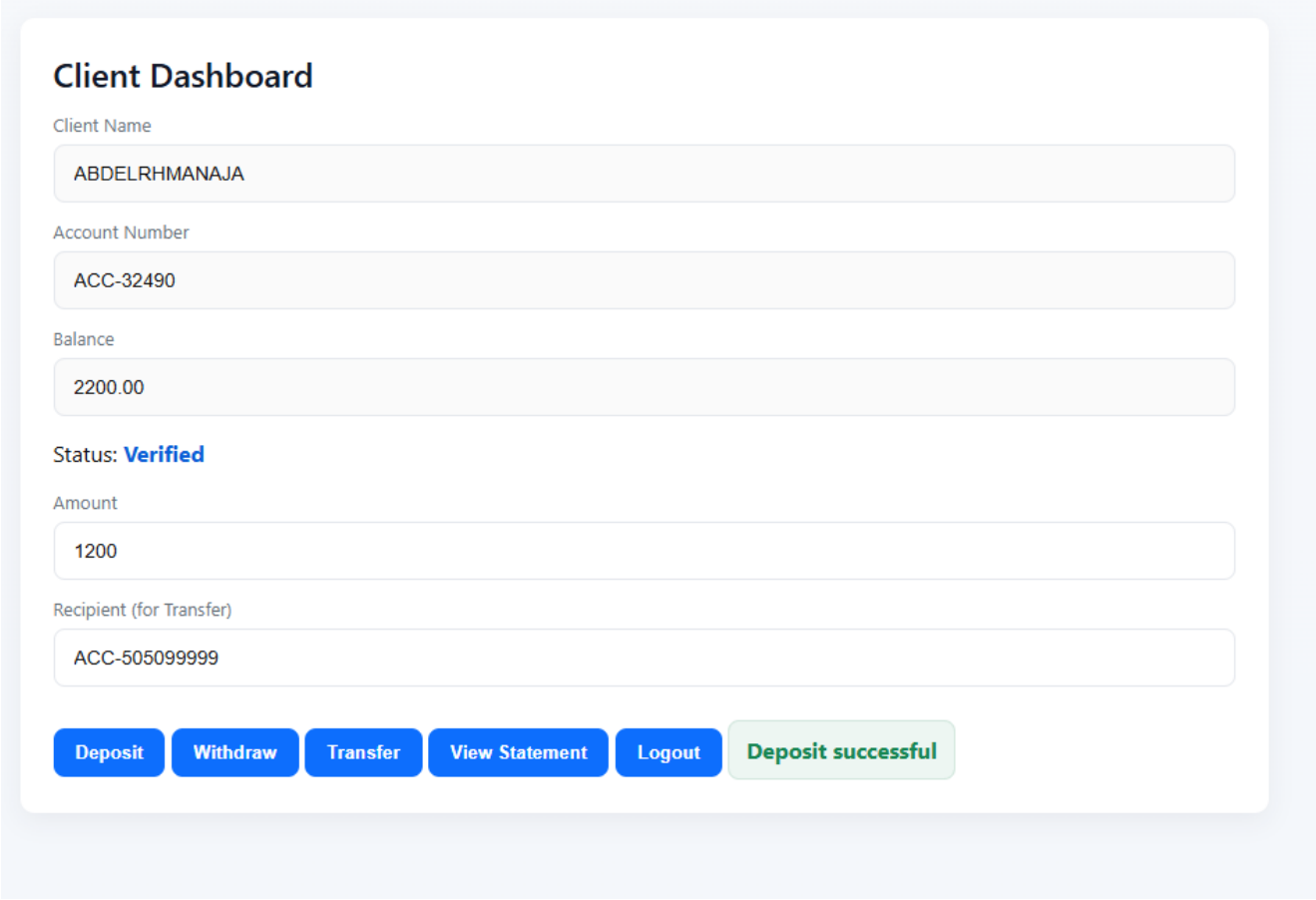
Section D: State-Based Testing

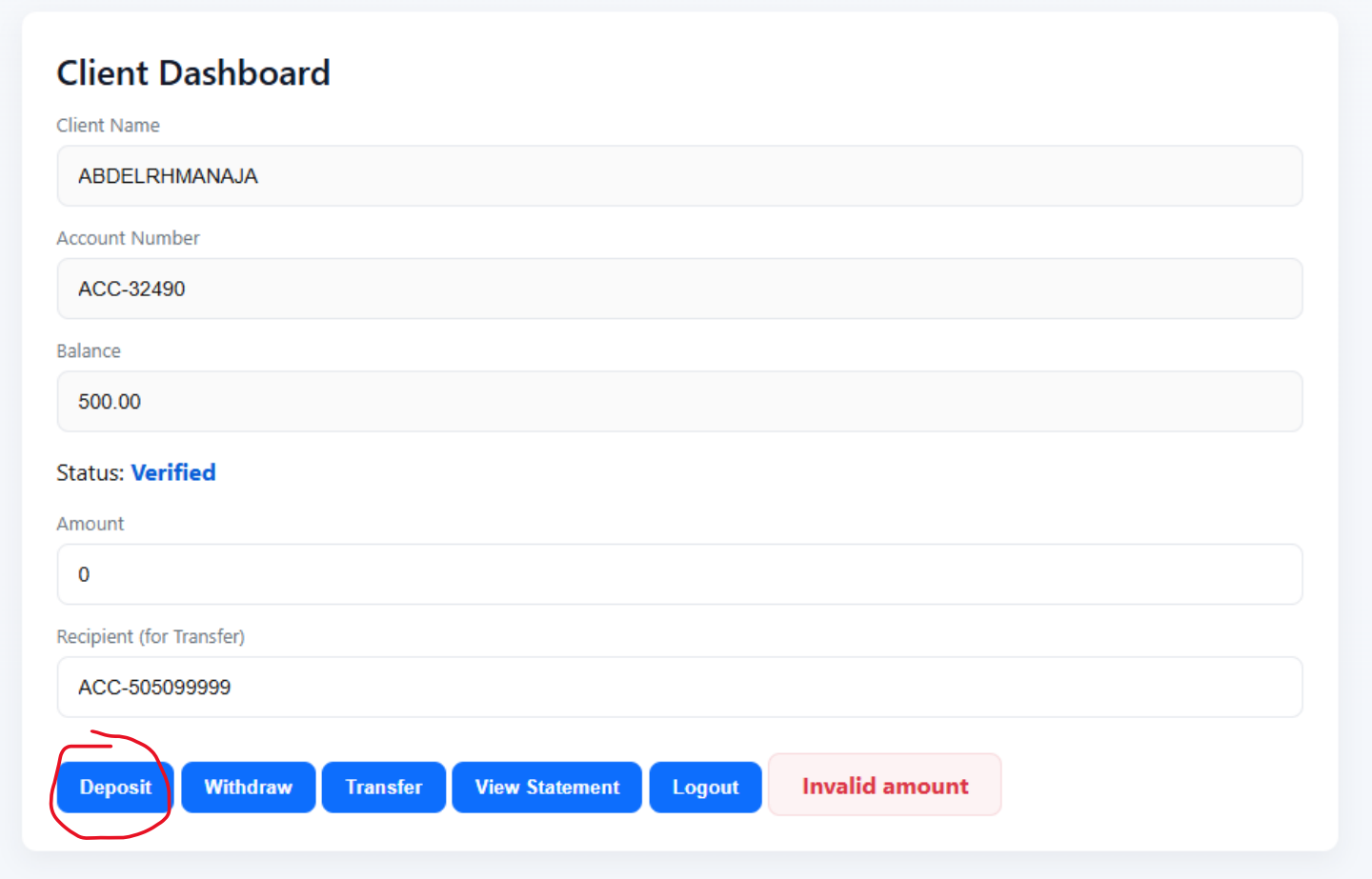
D.1 Objective

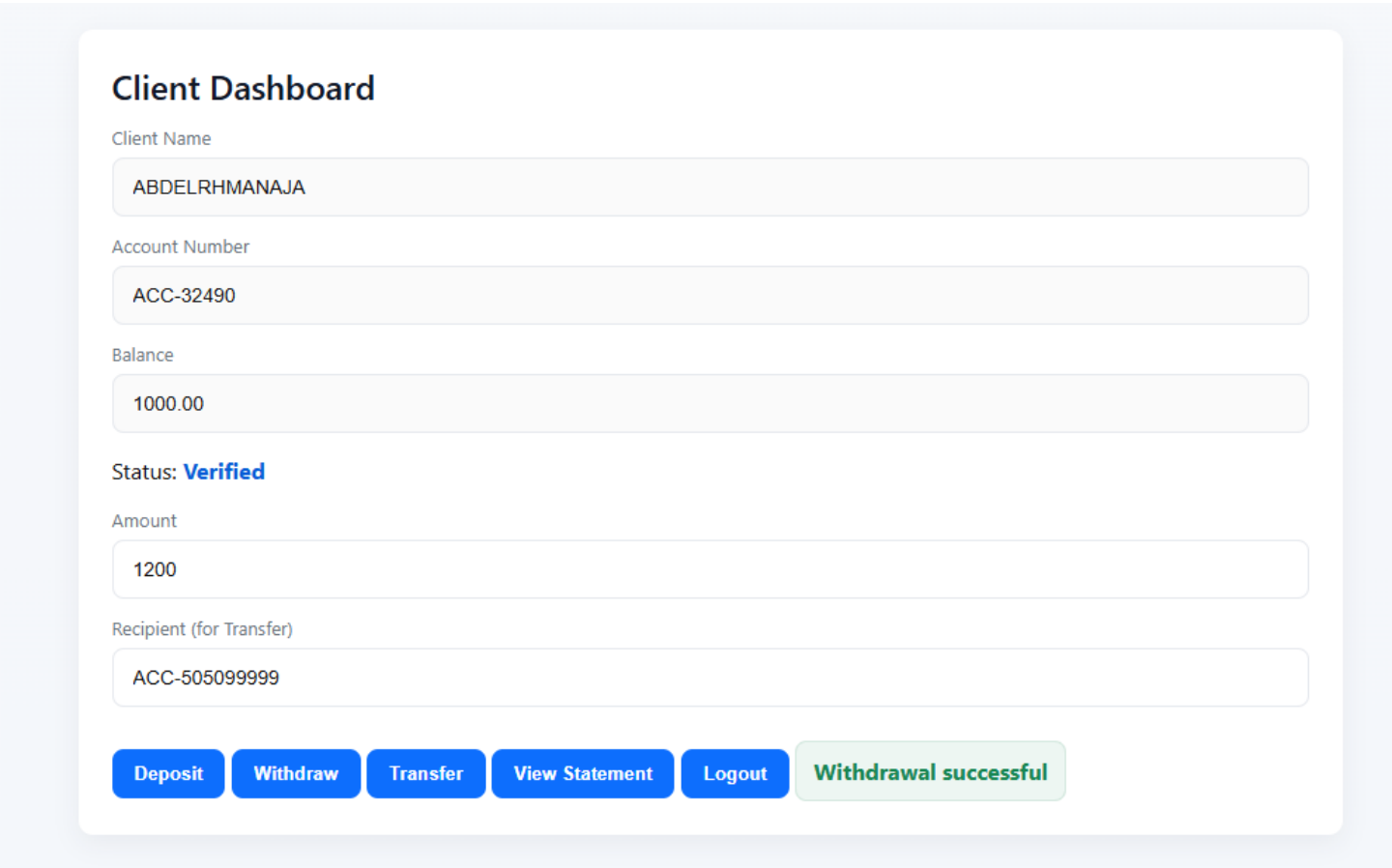
State-based testing ensures that the system behaves correctly under different account states.

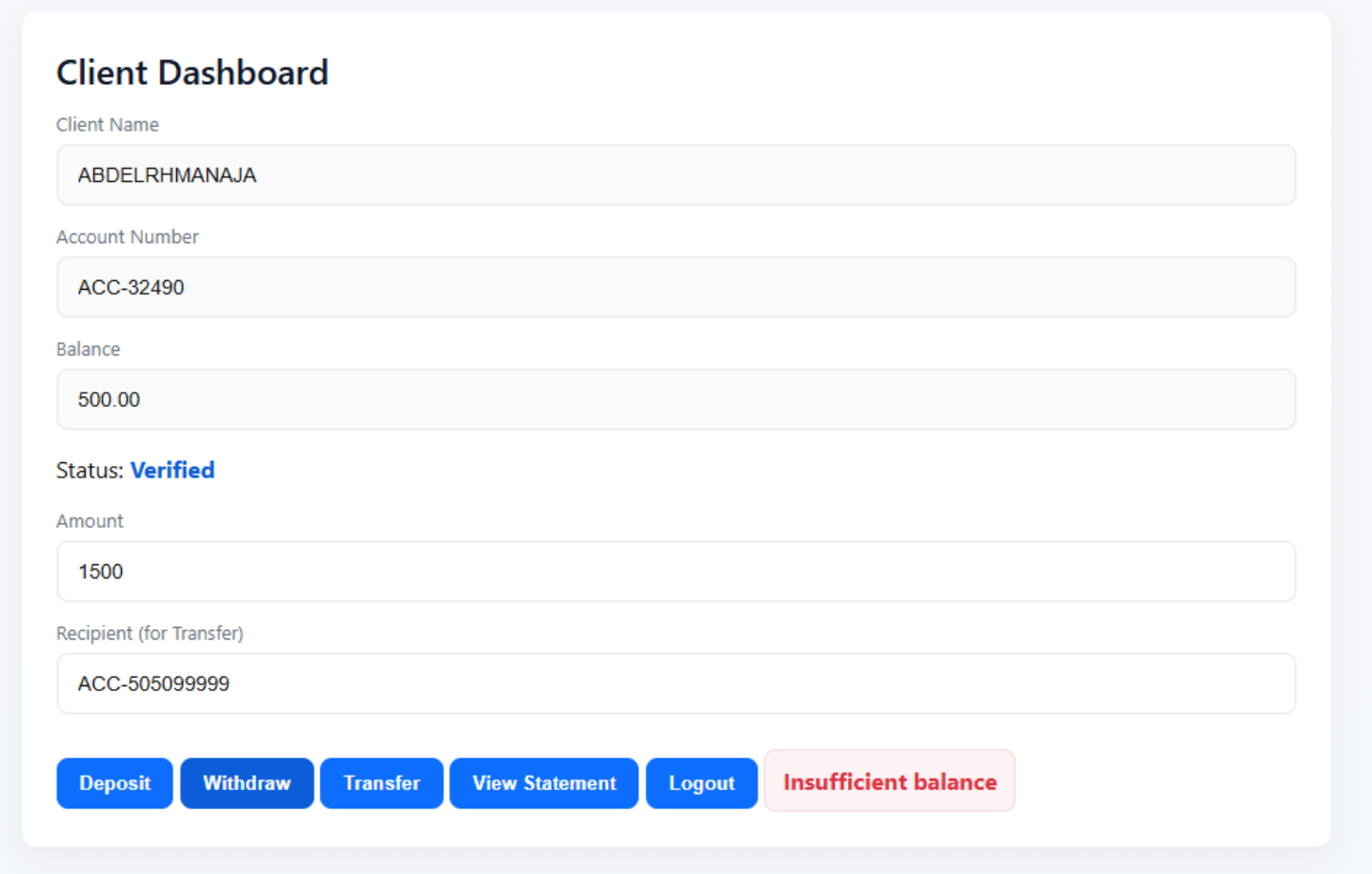
D.2 State Transition Matrix

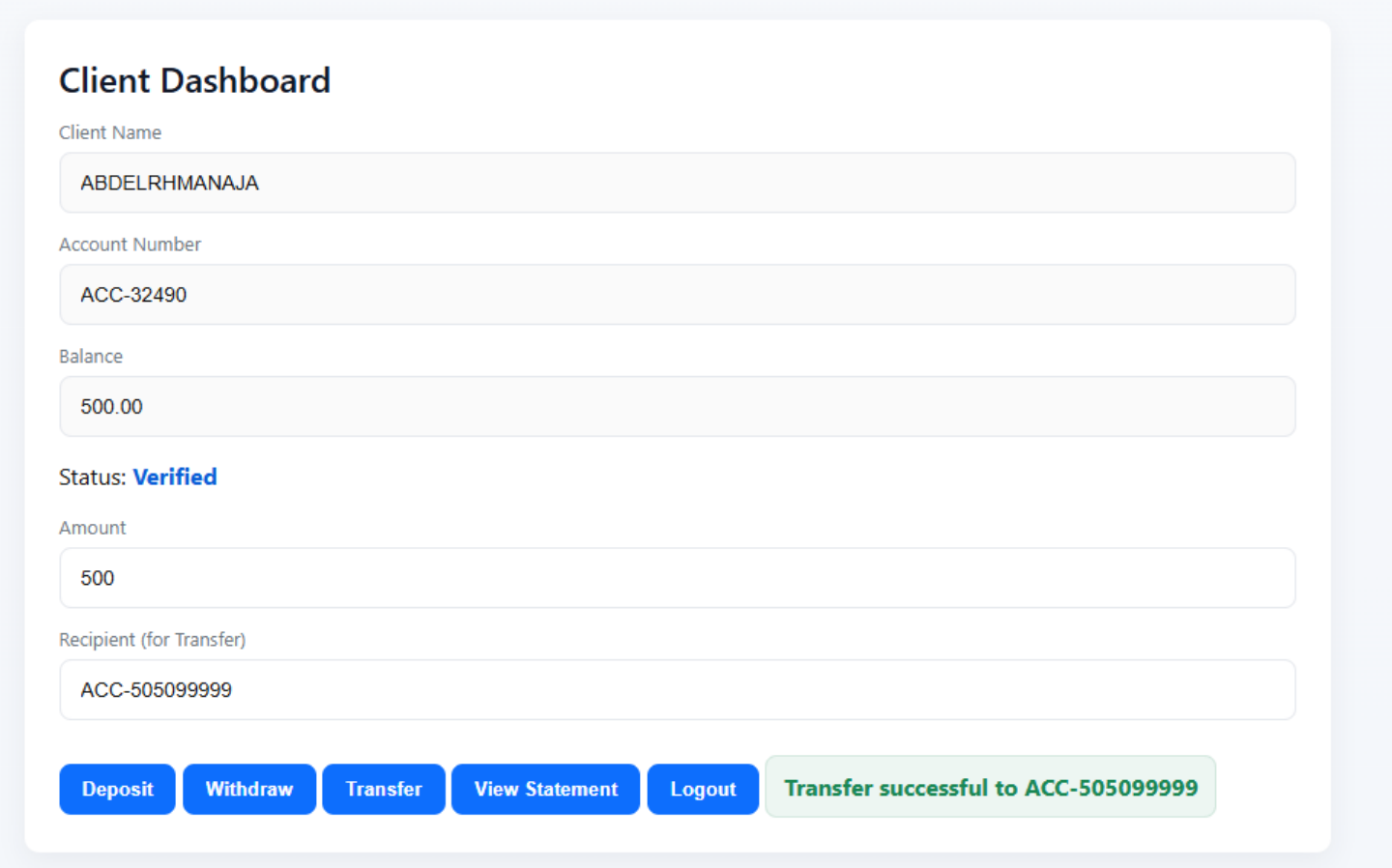
Current State	Action	Result
Verified	Deposit	Allowed
Suspended	Withdraw	Blocked
Closed	Deposit	Blocked
Suspended	Appeal	Verified




Client Dashboard
Client Name
ABDELRHMANAJA
Account Number
ACC-32490
Balance
500.00
Status: Verified
Amount
0
Recipient (for Transfer)
ACC-505099999
Deposit
Withdraw
Transfer
View Statement
Logout
Invalid amount


Client Dashboard
Client Name
ABDELRHMANAJA
Account Number
ACC-32490
Balance
1000.00
Status: Verified
Amount
1200
Recipient (for Transfer)
ACC-505099999
Deposit
Withdraw
Transfer
View Statement
Logout
Withdrawal successful


Client Dashboard
Client Name
ABDELRHMANAJA
Account Number
ACC-32490
Balance
500.00
Status: Verified
Amount
1500
Recipient (for Transfer)
ACC-505099999
Deposit
Withdraw
Transfer
View Statement
Logout
Insufficient balance


Client Dashboard
Client Name
ABDELRHMANAJA
Account Number
ACC-32490
Balance
500.00
Status: Verified
Amount
500
Recipient (for Transfer)
ACC-505099999
Deposit
Withdraw
Transfer
View Statement
Logout
Transfer successful to ACC-505099999

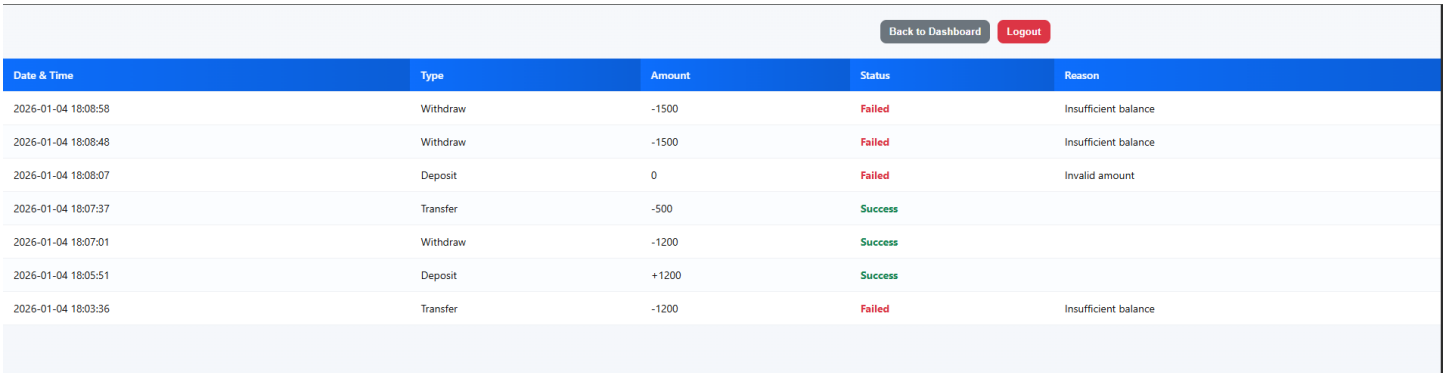
Section E: Test-Driven Development (TDD)

E.1 New Feature

Client Credit Score Check

E.2 TDD Process

1. Write failing test
2. Create stub code
3. Implement full logic
4. Refactor



4. Conclusion

This project applied several software testing techniques to a simplified banking system to ensure correct and reliable behavior. Black-box testing verified core functionalities such as deposits, withdrawals, and transfers, while white-box testing ensured that all internal logic paths were executed and covered.

UI testing confirmed that the user interface displays correct information, enforces input validation, and restricts actions based on account state. State-based testing ensured that illegal operations were blocked and valid state transitions were allowed. Integration testing verified proper interaction between system components, and test-driven development was demonstrated through the implementation of a credit score feature.

Overall, the project shows how combining different testing approaches helps improve software quality and reliability.