PROJECT REPORT

ON

AUTOMATE BANKING WEBSITE MANUAL & AUTOMATION TESTING

Submitted By:

Nikhil Harwani

1. ABSTRACT

In the era of digital finance, the reliability and security of online banking platforms are critical. This project report details the development and testing of the "Automate Banking Website," a web-based application designed to facilitate secure fund transfers, account management, and real-time transaction history.

The project adopts the Agile methodology to ensure iterative development and rigorous testing cycles. A major focus of this report is the Manual Test Case design and execution, ensuring that critical financial logic—such as balance validation and beneficiary addition—functions correctly under various constraints.

The resulting system offers a secure interface for customers to perform day-to-day banking activities without visiting physical branches. This report documents the Software Development Life Cycle (SDLC), highlighting the System Requirement Specifications (SRS), architectural design, and a comprehensive suite of manual test cases used to validate the system.

2. INTRODUCTION

2.1 Introduction

The Automate Banking Website is a comprehensive financial application that allows users to manage their bank accounts via the internet. It includes features for account creation, fund transfers (NEFT/RTGS), viewing mini-statements, and managing profile security. The project emphasizes robust validation mechanisms to prevent transactional errors and ensure data integrity.

2.2 Problem Identification

Traditional banking and legacy systems often face the following challenges:

- **Physical dependency:** Customers must visit branches for basic transfers.
- **Manual Errors:** Paper-based processing leads to data entry mistakes.
- **Time Consumption:** Queue management and manual verification delay transactions.
- **Limited Accessibility:** Services are restricted to banking hours.

2.3 Need of the Project

To modernize the banking experience, the system is required to:

1. Provide 24/7 access to banking services.
2. Automate balance checks and fund deduction logic.
3. Securely store sensitive user data (passwords, PINs).
4. Generate instant digital statements for users.

2.4 Project Scheduling (Agile Timeline)

The project was executed in 4 Sprints, focusing on development followed by immediate testing:

Phase	**Duration**	**Deliverable**
Sprint 1	2 Weeks	Requirement Gathering, DB Schema, Login/Auth Module
Sprint 2	2 Weeks	Account Dashboard, Balance Check, Beneficiary Mgmt
Sprint 3	2 Weeks	Fund Transfer Module, Transaction History, Pin Validation
Sprint 4	2 Weeks	Statement Generation, Integration Testing, Final Deployment

2.5 Objectives

- To develop a secure online banking portal using Java/J2EE or Node.js.
- To implement ACID properties in database transactions.
- To design comprehensive test cases covering positive and negative scenarios.
- To ensure the application is responsive and user-friendly.

3. SOFTWARE REQUIREMENT SPECIFICATION

3.1 Purpose

The purpose of this SRS is to define the functional requirements for the banking system, serving as the blueprint for both the development team and the Quality Assurance (QA) team for writing test scripts.

3.2 Scope

The scope includes the **Customer Module** (Login, Transfer, Statement) and the **Admin Module** (Approve Account, View Logs). It specifically excludes Credit Card management and Loan processing in this version.

3.3 System Requirements

Hardware Requirements (Minimum)

- **Processor:** Intel Core i5 or higher
- **RAM:** 8 GB (for running server and DB locally)
- **Hard Disk:** 10 GB free space
- **Network:** Broadband connection for API calls

Software Requirements

- **Operating System:** Windows 10/11 or Linux
- **Frontend:** HTML5, CSS3, React.js/Angular
- **Backend:** Java Spring Boot / Node.js
- **Database:** Oracle 11g or MySQL
- **Testing Tool:** Selenium WebDriver (for automation), Manual Testing

3.4 Tools Used

- **Eclipse/IntelliJ:** IDE for coding.
- **MySQL Workbench:** Database Design.

- **Jira:** Defect Tracking and Sprint Planning.
- **Selenium:** For automated regression testing.

3.5 Software Process Model

Agile Scrum Model: This model was selected to allow for continuous testing. As banking regulations and security requirements can change, Agile allows the team to adapt quickly without dismantling the entire project structure.

4. SYSTEM DESIGN

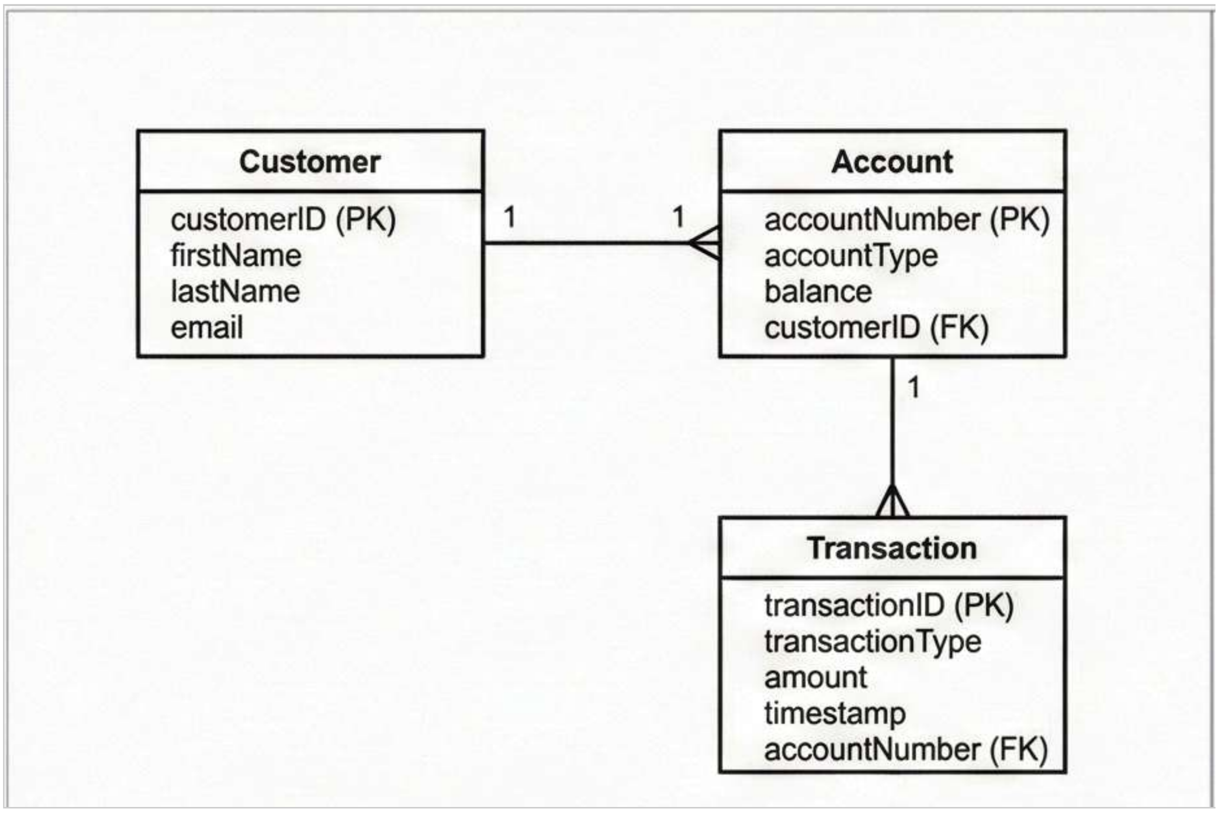
4.1 Data Dictionary

Table: Transactions

Field	**Type**	**Description**
trans_id	LONG (PK)	Unique Transaction Reference
from_acc	VARCHAR	Sender Account Number
to_acc	VARCHAR	Receiver Account Number
amount	DECIMAL	Transfer Amount
status	ENUM	Success/Failed/Pending

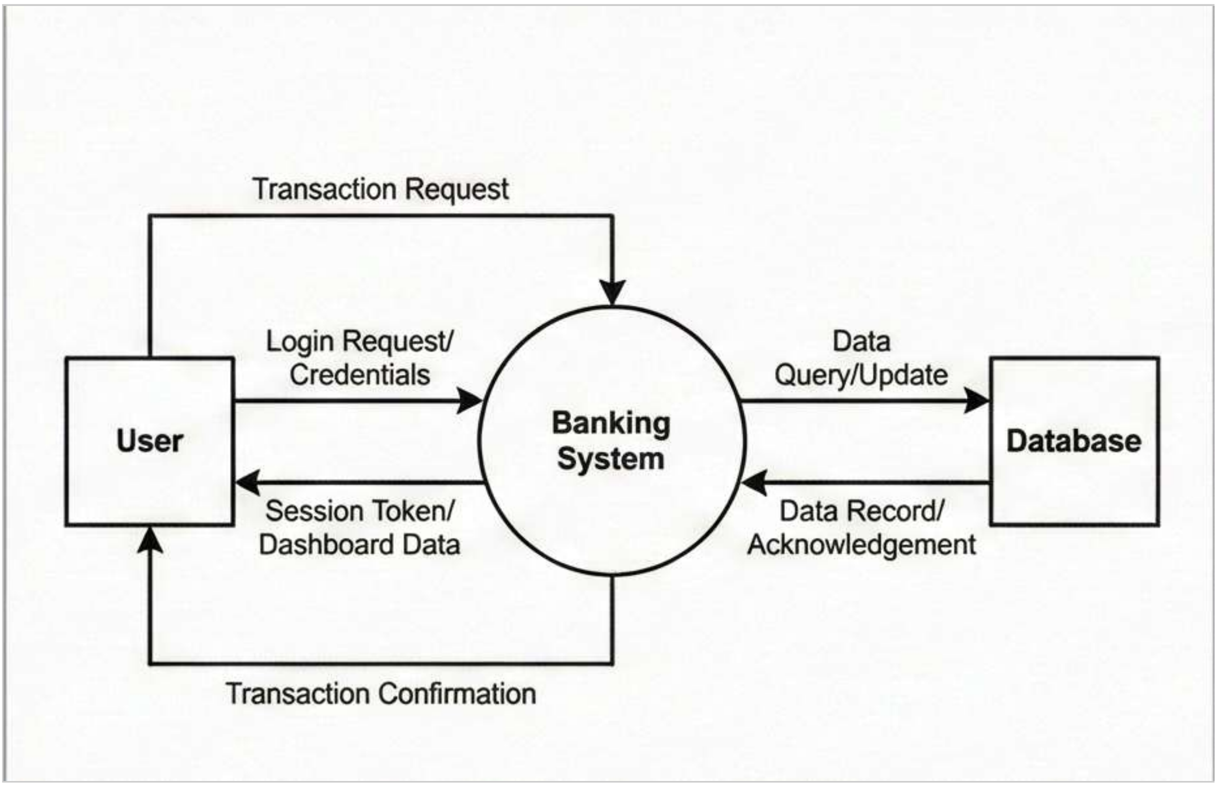
4.2 ER Diagram

Entity Relationship Diagram (Banking System)



4.3 Data Flow Diagram (DFD)

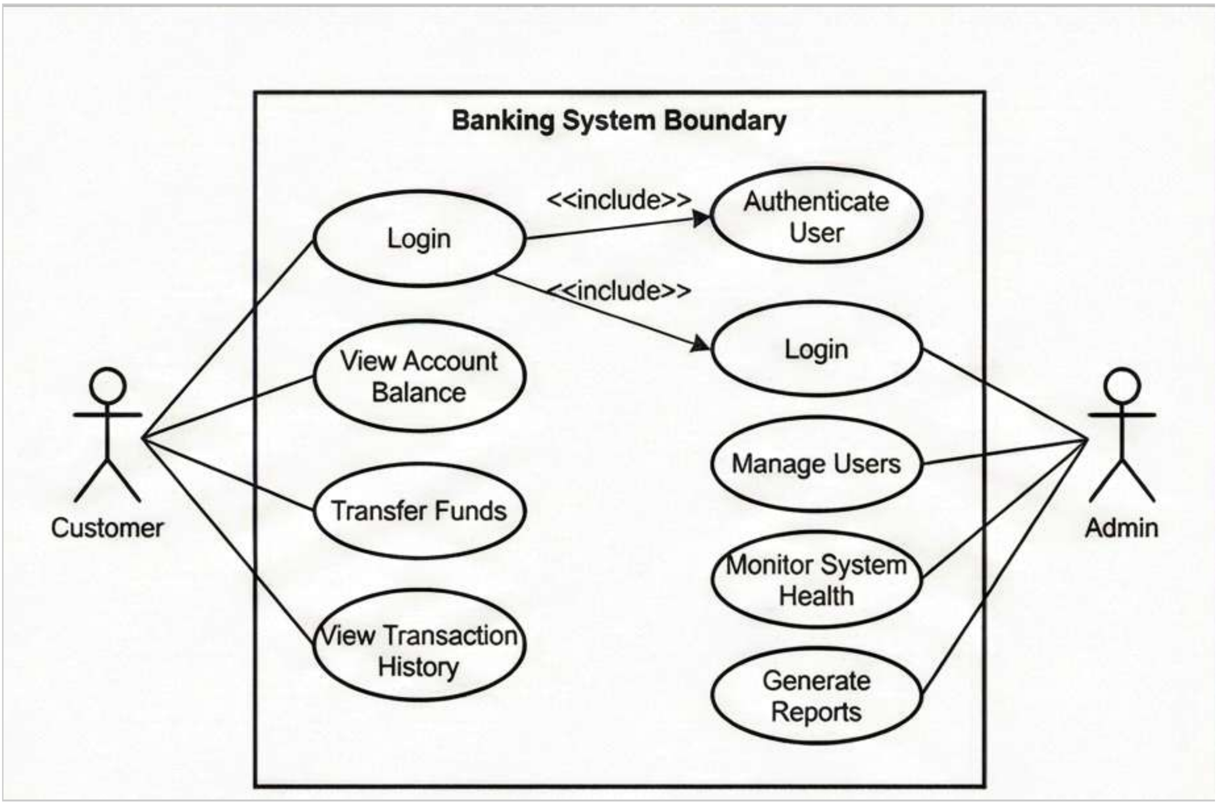
Data Flow Diagram (Level 1)



4.4 UML Diagrams

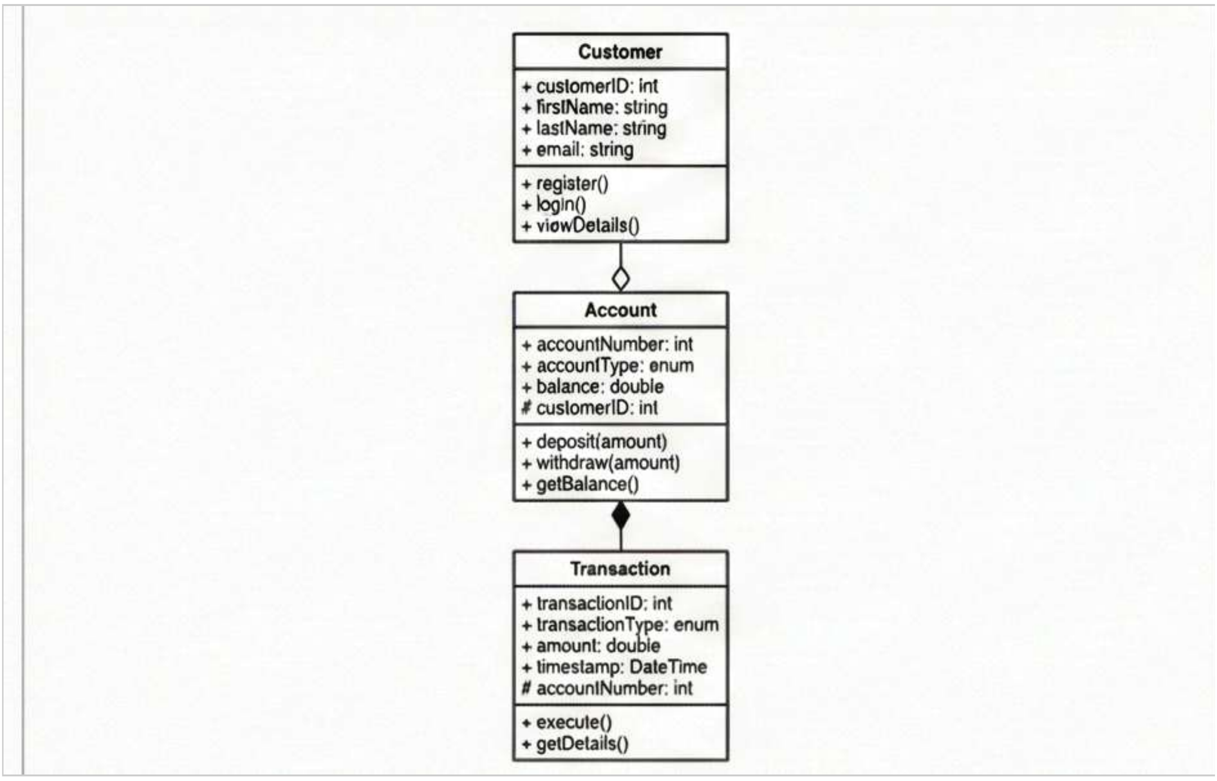
Use Case Diagram

Use Case Diagram



Class Diagram

Class Diagram



5. IMPLEMENTATION

5.1 Program Code

Below are core snippets of the backend logic used for Fund Transfers.

Backend Controller (Fund Transfer Logic)

```java
// TransferController.java
@PostMapping("/api/transfer")
public ResponseEntity transferFunds(@RequestBody TransferRequest req) {
    Account sender = accountRepo.findByAccNo(req.getFromAcc());
    Account receiver = accountRepo.findByAccNo(req.getToAcc());

    if (sender.getBalance() < req.getAmount()) {
        return ResponseEntity.badRequest().body("Insufficient Funds");
    }

    sender.setBalance(sender.getBalance() - req.getAmount());
    receiver.setBalance(receiver.getBalance() + req.getAmount());

    Transaction trans = new Transaction(req.getAmount(), "SUCCESS");
    transactionRepo.save(trans);

    return ResponseEntity.ok("Transfer Successful");
}
```

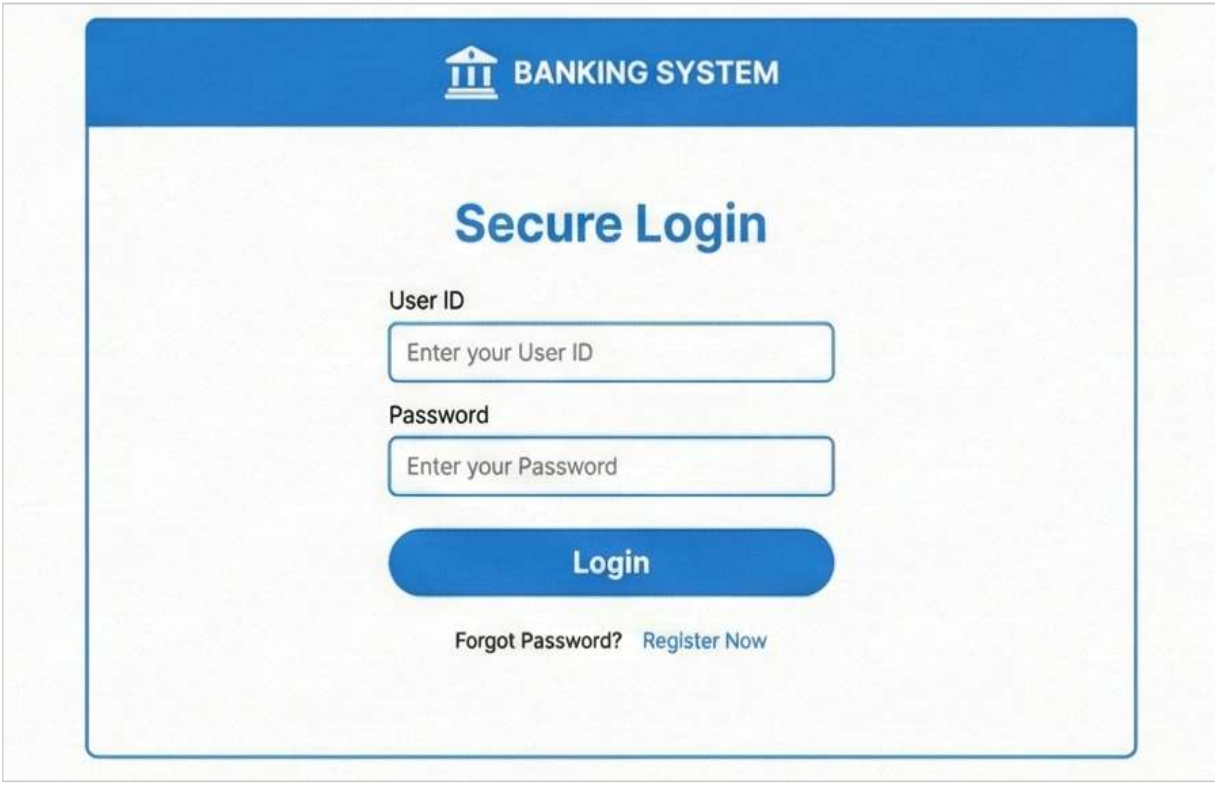
Database Connection

```
// DBConnection.java
public class DatabaseConfig {
    public Connection connect() {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            return DriverManager.getConnection(
                           "jdbc:mysql://localhost:3306/bank_db",  "root",
"password");
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

5.2 Output Screens

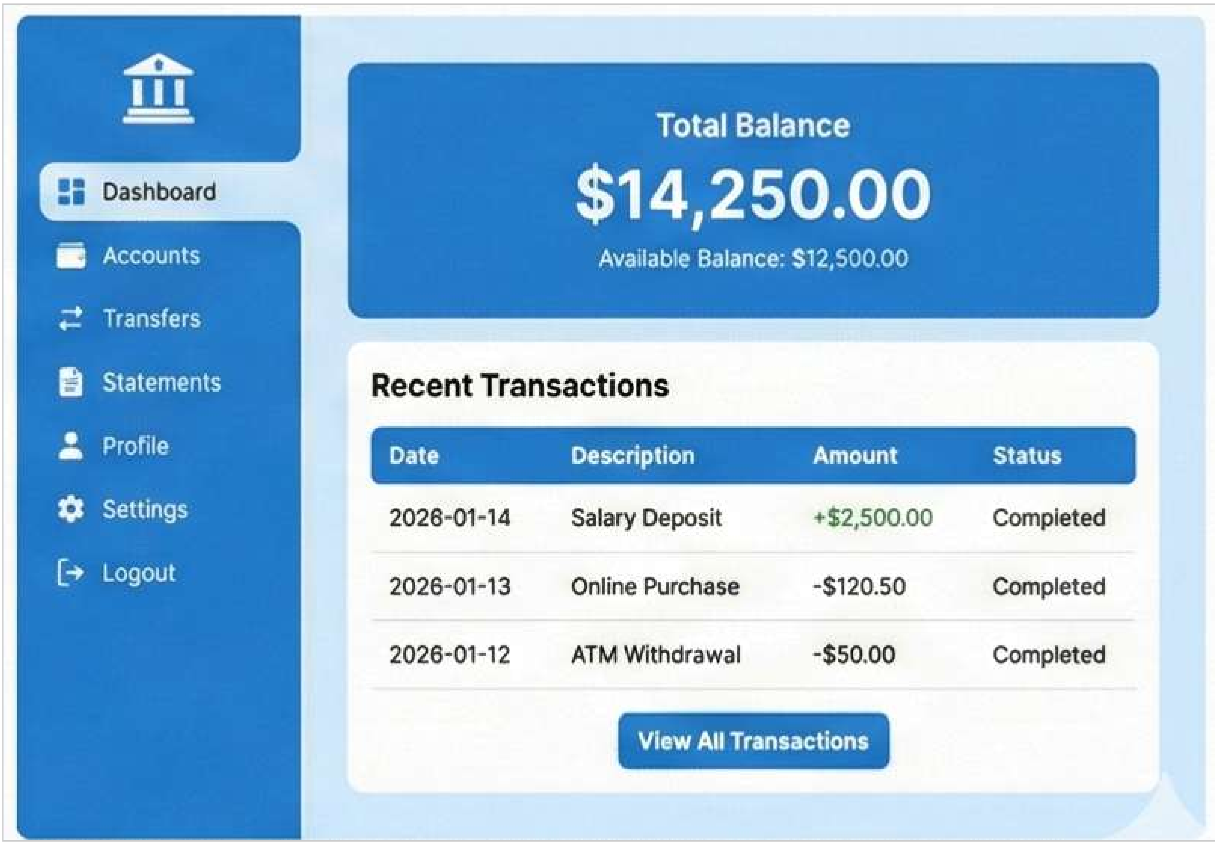
Login Screen:

Login Interface Screenshot



Account Dashboard:

Account Summary Screenshot



Total Balance

$14,250.00

Available Balance: $12,500.00

Recent Transactions

Date	Description	Amount	Status
2026-01-14	Salary Deposit	+$2,500.00	Completed
2026-01-13	Online Purchase	-$120.50	Completed
2026-01-12	ATM Withdrawal	-$50.00	Completed

6. TESTING

6.1 Test Data & Strategy

The testing strategy focused on Black Box Testing. Valid and invalid data sets were used to verify boundary value analysis (e.g., transferring 0 amount, negative amount, or exact balance amount).

6.2 Test Results (Manual Test Cases)

Test Case ID	Description	Input Data	Expected Output	Status
TC_01	Verify Login with Valid Credentials	User: customer1 / Pass: Pass@123	Redirect to Dashboard	PASS
TC_02	Verify Login with Invalid Password	User: customer1 / Pass: wrongpass	Error: "Invalid Credentials"	PASS
TC_03	Fund Transfer (Sufficient Balance)	From: Acc A, To: Acc B, Amt: 5000	"Transfer Successful", Balance updates	PASS
TC_04	Fund Transfer (Insufficient Balance)	Balance: 1000, Transfer: 5000	Error: "Insufficient Funds"	PASS
TC_05	Beneficiary Account Format Check	Acc No: "ABCDE" (Alphabets)	Error: "Account No must be numeric"	PASS

7. USER MANUAL

7.1 How to use project guidelines

1. **Registration:** Visit the homepage and click "New User". Fill in KYC details.
2. **Login:** Use the Customer ID sent to your email to log in.
3. **Add Beneficiary:** Go to Transfers > Manage Beneficiaries > Add New.
4. **Transfer:** Select Beneficiary, Enter Amount, Enter Transaction PIN.

7.2 Screen Layouts and Description

Header: Contains Bank Logo, Date, and Logout button.
Left Menu: Quick links to Account Summary, Fund Transfer, Statement, and Profile.
Main Content Area: Displays the active form or transaction receipt.

8. PROJECT APPLICATIONS AND LIMITATIONS

Applications

- **Retail Banking:** Personal account management for individuals.
- **Corporate Banking:** Salary disbursements for small companies.
- **E-Statement:** Reducing paper waste by generating PDF statements.

Limitations

- OTP (One Time Password) integration is simulated, not real-time SMS.
- Cross-border (International) transactions are not supported.
- The system does not currently support biometric authentication.

9. CONCLUSION AND FUTURE ENHANCEMENT

The "Automate Banking Website" project was successfully designed and tested. The manual test cases confirmed that the system handles financial transactions accurately and securely. By adopting Agile, the team identified critical bugs in the "Fund Transfer" module early in Sprint 3, which were resolved before final deployment.

Future Enhancements:

- **Chatbot Integration:** AI-powered bot to answer FAQs regarding balance and status.
- **Mobile App:** Developing a React Native version for Android/iOS.
- **Blockchain:** Implementing blockchain for immutable transaction ledgers.

10. BIBLIOGRAPHY & REFERENCES

Books:

- Pressman, R. S. (2014). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education.
- Gupta, A. (2019). *Software Testing: Principles and Practice*. Alpha Science.

Websites:

- Selenium Documentation (https://www.selenium.dev/)
- Spring Boot Guide (https://spring.io/projects/spring-boot)
- W3Schools (HTML/CSS Standards)