**Capstone Project 2: Banking Loan & Financial Transaction Automation Testing**

**Introduction**

Banks and financial institutions manage **customer accounts, loans, and transactions** through online platforms.  
Any errors in these systems can lead to:

* Financial frauds
* Incorrect balance/interest calculations
* Compliance violations

This project focuses on building a **robust automation framework** for testing critical banking workflows such as **loan management, fund transfers, and transaction validation** using **ParaBank** (<http://parabank.parasoft.com/parabank>).

**Problem Statement**

Manual verification of banking transactions, loan approvals, EMI schedules, and account balances is:

* **Repetitive**
* **Time-consuming**
* **Error-prone**

A well-structured **automation framework** is required to:

* Validate **loan applications and approval workflows**
* Verify **fund transfers and balance updates**
* Ensure **transaction history**

**Objectives**

* Automate **loan application processing** and approval/rejection workflows.
* Validate **fund transfers** and **balance updates**.
* Automate **transaction history search and validation**.
* Implement **cross-browser** compatibility testing.
* Enable **parallel execution and CI/CD reporting** using TestNG and Jenkins.

**Scope**

The automation suite covers the following modules:

1. **Customer Authentication (Login)** – validate login/logout, invalid cases.
2. **Account Dashboard / Balances** – verify balances after operations.
3. **Loan Application** – apply valid/invalid loans, verify approval/rejection.
4. **Fund Transfers** – money transfer between accounts and validations.
5. **Transaction History** – search by date/amount and validate entries.

**Tech Stack**

* **Language:** Java
* **Automation Tool:** Selenium WebDriver
* **Testing Framework:** TestNG
* **Build Tool:** Maven
* **Driver Management:** WebDriverManager
* **Reporting:** TestNG HTML reports + screenshots on failure
* **CI/CD:** Jenkins Pipeline (Jenkinsfile)
* **Design Pattern:** Page Object Model (POM)

**Functionalities Automated**

1. **Login & Account Dashboard** – validate login, logout, and account overview.
2. **Loan Application Form** – automate loan submission and approval workflow.
3. **Loan Approval/Rejection Rules** – handle valid, invalid, and edge cases.
4. **Fund Transfers** – verify debit/credit updates for valid/invalid transfers.
5. **Transaction History** – validate recent transfers and searches.

**Test Scenarios Implemented**

**🔐 Login Module (10 tests)**

* Valid login → verify dashboard, then logout.
* Invalid login (wrong password / wrong username).
* Empty credentials → show error.
* Multiple invalid attempts → show error/lockout.
* Valid login with uppercase username (currently fails – reflects app behavior).
* Login without password → error.
* Login without username → error.
* Logout after login → verify Customer Login page.
* Valid login with uppercase password (fails – reflects app behavior).

**🏦 Loan Module (9 tests)**

* Apply valid loan → verify approval/rejection result.
* Apply loan exceeding balance.
* Apply loan with zero down payment.
* Apply loan with invalid input (non-numeric).
* Apply minimum loan amount.
* Apply maximum loan amount.
* Apply loan with empty fields.
* Apply loan with down payment > loan amount.
* Apply multiple loans → validate system’s handling of concurrent requests.

**🔄 Transfer Module (7 tests)**

* Valid transfer.
* Small transfer (e.g., $1).
* Invalid transfer (non-existing accounts).
* Negative amount transfer.
* Exceeding balance transfer.
* Explicit test: transferNegativeAmount.
* Explicit test: transferInvalidAccounts.

**💰 Balance Module (1 test)**

* balanceAfterDeposit → verify Accounts Overview shows updated balances.

**📜 Transaction Module (3 tests)**

* Verify recent transfer appears in history.
* Search by amount → expect matching results.
* Search by date range → expect results or “no transactions” message.

**Framework Design**

* **Page Object Model (POM):** each web page modeled into reusable page classes.
* **BaseTest:** handles setup/teardown, WebDriver, screenshots.
* **WebDriverFactory:** creates browser instances (Chrome/Firefox/Edge).
* **TestNG Suite (testng.xml):** groups all modules for single-run execution.
* **Jenkinsfile:** pipeline automation (checkout → build → test → reports).

**Expected Outcomes**

* Automated suite covering all **core banking flows**.
* Reliable regression testing for login, loans, transfers, and transactions.
* Reduced manual effort & faster execution cycles.
* Clear reporting (console logs, TestNG reports, Jenkins artifacts).
* Improved confidence in banking system stability.

**Conclusion**

* Built a **scalable, maintainable automation framework** for banking applications.
* Automated **30+ functional and negative test cases** across Login, Loan, Transfer, Balance, and Transaction modules.
* Delivered an **end-to-end solution** that improves software quality, reduces manual testing effort, and provides quick feedback to stakeholders.

**Future Enhancements**

* Add **ExtentReports / Allure** for detailed HTML reports.
* Add **parallel execution** for faster test cycles.
* Extend coverage to **Bill Payments, Account Creation, Profile Updates**.
* Integrate with **cloud-based Selenium Grids (e.g., BrowserStack/SauceLabs)** for cross-browser/device testing.