**Test Cases for Parabank**

**1. User Registration:**

Automate the registration process by filling out the form with different sets of data (positive and negative scenarios).

Validate success or failure messages based on the input data.

1. **Go to parabank link**
2. **Click on the Register link**
3. **Fill the details**
4. **Click on Register button**
5. **Once after clicking on Register link verify Welcome “Username” is displayed**
6. **Click on logout link.**

For a banking project to practice Selenium TestNG automation with data-driven testing and Page Object Model (POM), Parabank is one of the best choices due to its comprehensive banking functionalities. Here’s a detailed guide to the Parabank project, including key features to automate and how you can use it for testing.

Website: Parabank

URL: Parabank

Features to Automate in Parabank:

User Registration:

Automate the registration process by filling out the form with different sets of data (positive and negative scenarios).

Validate success or failure messages based on the input data.

Login and Logout:

Automate login with valid and invalid credentials.

Verify that the user can log out successfully after logging in.

Account Overview:

After logging in, validate the Account Overview page, which shows all the user’s accounts and balances.

Verify the correct account details are displayed based on the user data.

Fund Transfer:

Automate the Transfer Funds functionality.

Validate successful fund transfer between two accounts (e.g., savings to checking).

Test error scenarios like transferring more funds than available balance.

Request Loan:

Automate the Loan Request process, where a user can request a loan by filling in the required fields.

Validate that the loan request submission works correctly and displays the success or failure messages.

Transaction History:

Verify the Transaction History feature where you can view recent transactions made on an account.

Ensure all transaction details (amount, type, date) are displayed accurately.

Bill Pay:

Automate the bill payment functionality where users can enter recipient details and pay bills.

Validate success messages and ensure the correct amount is deducted from the user’s account.

Account Creation:

Automate account creation (e.g., opening a new savings or checking account).

Verify that the account appears in the Account Overview after creation.

or a banking project using Selenium TestNG with Data-Driven Testing and Page Object Model (POM), your folder structure should be clean, organized, and scalable. Here’s a sample folder structure to help you set up your banking project efficiently.

Folder Structure for Selenium TestNG Project:

Parabank\_E2E\_TestNG

│

├── src

│ ├── main

│ │ ├── java

│ │ │ ├── com.bank.pages

│ │ │ │ ├── HomePage.java

│ │ │ │ ├── RegistrationPage.java

│ │ │ │ ├── LoginPage.java

│ │ │ │ ├── AccountOverviewPage.java

│ │ │ │ ├── TransferFundsPage.java

│ │ │ │ ├── BillPayPage.java

│ │ │ │ ├── LoanRequestPage.java

│ │ │ │ ├── TransactionHistoryPage.java

│ │ │ ├── com.bank.utils

│ │ │ │ ├── ExcelUtil.java

│ │ │ │ ├── ConfigReader.java

│ │ │ │ ├── TestUtils.java

│ │ │ ├── com.bank.base

│ │ │ │ ├── BasePage.java

│ │ │ │ ├── BaseTest.java

│ ├── test

│ │ ├── java

│ │ │ ├── com.bank.tests

│ │ │ │ ├── RegistrationTest.java

│ │ │ │ ├── LoginTest.java

│ │ │ │ ├── AccountOverviewTest.java

│ │ │ │ ├── TransferFundsTest.java

│ │ │ │ ├── LoanRequestTest.java

│ │ │ │ ├── TransactionHistoryTest.java

│ │ │ │ ├── BillPayTest.java

│ ├── resources

│ │ ├── testdata

│ │ │ ├── TestData.xlsx

│ │ ├── config

│ │ │ ├── config.properties

│ │ ├── logs

│ │ │ ├── project.log

│ │ ├── reports

│ │ │ ├── TestReport.html

│

├── pom.xml

├── README.md

└── .gitignore

Explanation of Folder Structure:

1. src/main/java:

This directory contains all the Page Object Model (POM) classes, utility files, and base classes for the project.

com.bank.pages: This package contains individual page classes that represent the web pages of the Parabank application. Each class contains locators and actions related to that page.

HomePage.java: Contains elements and methods for login and navigation.

RegistrationPage.java: Handles the registration form automation.

LoginPage.java: Automates login functionality.

AccountOverviewPage.java: Automates viewing account details.

TransferFundsPage.java: Automates the fund transfer feature.

BillPayPage.java: Handles bill payment automation.

LoanRequestPage.java: Automates loan requests.

TransactionHistoryPage.java: Automates viewing transaction history.

com.bank.utils: This package contains utility classes that help with repetitive tasks, such as reading data from Excel or properties files.

ExcelUtil.java: Handles reading test data from Excel files (for data-driven testing).

ConfigReader.java: Reads configurations from a properties file.

TestUtils.java: Contains common utilities such as taking screenshots, waiting for elements, etc.

com.bank.base: This package contains base classes for pages and test classes.

BasePage.java: Contains common methods like click, send text, wait for elements, and other reusable WebDriver actions.

BaseTest.java: Sets up and tears down the WebDriver. All test classes extend this.

2. src/test/java:

This directory contains the test classes, which hold the actual test cases for each feature of the banking application.

com.bank.tests: This package contains test classes for different functionalities.

RegistrationTest.java: Contains test cases for user registration.

LoginTest.java: Contains login tests with valid and invalid credentials.

AccountOverviewTest.java: Verifies account overview details.

TransferFundsTest.java: Contains test cases for fund transfers.

LoanRequestTest.java: Tests loan requests.

TransactionHistoryTest.java: Verifies transaction history.

BillPayTest.java: Tests bill payment functionality.

3. src/test/resources:

This directory contains all the external resources required for the tests.

testdata: Contains test data files such as TestData.xlsx, which is used for data-driven testing.

config: Contains configuration files like config.properties that store environment settings (URL, browser type, etc.).

logs: Stores log files generated during the test execution. You can configure loggers (like Log4j) to track the actions in your tests.

reports: Stores the generated test reports. After test execution, TestNG reports or custom reports (e.g., HTML reports) can be saved here.

4. pom.xml:

Maven configuration file, which manages dependencies (e.g., Selenium, TestNG, Apache POI for Excel, Log4j, etc.).

5. README.md:

A readme file to explain the project, how to set it up, and how to run the tests.

6. .gitignore:

Specifies files and folders to ignore when pushing to version control (e.g., logs, target folder, etc.).

How to Use This Structure:

Page Objects: Store all locators and methods for a page in one file inside com.bank.pages. Each page should handle its own elements and interactions.

Data-Driven Testing: Use ExcelUtil.java to read test data from the TestData.xlsx file located in the testdata folder.

ConfigReader: Use config.properties to manage configurations such as base URL, browser settings, or any environment-related information.

BasePage and BaseTest: These classes will handle common functionalities such as WebDriver setup, teardown, and utility methods to avoid repetition in your test classes.

Logs and Reports: Use logging frameworks like Log4j to generate logs in the logs folder and TestNG reports (or custom reports) in the reports folder.

Project Workflow:

Add Test Data: Store all your test data in the Excel file (TestData.xlsx).

Create Page Classes: Implement your Page Object Model in the com.bank.pages package.

Write Test Classes: Use TestNG to write test cases in the com.bank.tests package.

Execute Tests: Run your tests using Maven (mvn test) or TestNG XML suites.

View Reports: After test execution, review logs and reports generated in the logs and reports folders.

This structure ensures modularity, reusability, and scalability, which are essential for real-world projects. Let me know if you need more details on specific classes or test cases!