**SELENIUM AUTOMATION TESTING**

**Website Name: DemoBlaze**

**Website URL:** [**https://www.demoblaze.com/**](https://www.demoblaze.com/)

**Introduction**

This project is an automated test framework created to test the DemoBlaze e-commerce site. The framework is implemented on Selenium WebDriver, TestNG, and Cucumber, adhering to Page Object Model (POM), Page Factory for better maintainability. The major objective of the project is to provide an uninterrupted shopping experience for end users by testing login, signup, product pick, cart management, and checkout functionalities.

**Problem Statement**

Manual testing of e-commerce applications is a time-consuming and labour-intensive process with room for human error. With ongoing UI changes and feature additions, redundant testing proves to be wasteful. By automating the testing process, efficiency is much improved, execution time is significantly reduced, and application stability over various test cases and browsers is ensured.

**Tools & Techniques Used**

* **Programming Language:** Java
* **Build Tool:** Maven
* **Testing Frameworks:** TestNG, Cucumber
* **Automation Tool:** Selenium WebDriver
* **Framework Design Pattern:** Page Object Model (POM), Page Factory
* **Reporting Tools:** Extent Reports, Cucumber Reports
* **Data Handling:** Properties file for configuration
* **Test Execution:** TestNG XML,POM.XML
* **Exception Handling:** Try-catch blocks

**Project Objective**

The objective of this project is to automate the user workflows of the Demoblaze online shopping platform using Selenium, TestNG, Page Object Model and Page Factory and BDD Cucumber. This includes automating tasks such as viewing product details, adding products to the cart, placing orders. The aim is to enhance the robustness of the testing process, reduce manual effort, and ensure a seamless user experience.

**Project Structure**

**📂 Capstone\_Project  
│── 📂 src/main/java  
│ ├── 📂 Base  
│ │ ├── 📄 BaseClass.java  
│ ├── 📂 Data  
│ │ ├── 📄 config.properties  
│ ├── 📂 feature  
│ │ ├── 📄 Login.feature  
│ ├── 📂 pages  
│ │ ├── 📄 CartPage.java  
│ │ ├── 📄 CheckoutPage.java  
│ │ ├── 📄 HomePage.java  
│ │ ├── 📄 LoginPage.java  
│ │ ├── 📄 ProductPage.java  
│ │ ├── 📄 SignupPage.java  
│ ├── 📂 Utility  
│ │ ├── 📄 ExtentReportManager.java  
│ ├── 📂 Hook  
│ │ ├── 📄 Hook.java  
│── 📂 src/test/java  
│ ├── 📂 MyRunner  
│ │ ├── 📄 TestRunner.java  
│ ├── 📂 StepDef  
│ │ ├── 📄 LoginSteps.java  
│ ├── 📂 Testng  
│ │ ├── 📄 AddToCartTest.java  
│ │ ├── 📄 CheckoutTest.java  
│ │ ├── 📄 LoginTest.java  
│ │ ├── 📄 SignupTest.java  
│ │ ├── 📄 TestCases.java  
│ ├── 📂 screenshot**

**│── 📂 target  
│ ├── 📂 surefire-reports  
│ │ ├── 📄 cucumber-reports.html  
│ │ ├── 📄 ExtentReports-DemoBlaze.html  
│ ├── 📂 test-output  
│ │ ├── 📄 index.html  
│ │ ├── 📄 emailable-report.html  
│ │ ├── 📄 testng-results.xml  
│── 📂 xml  
│ ├── 📄 TestRunner.xml  
│ ├── 📄 Testcase.xml  
│── 📄 pom.xml**

**Key Components**

**1. Base Class**

* Initializes the browser using data from config.properties.
* Implements WebDriver methods for launching, navigating, and closing the browser.
* Includes methods for handling waits (implicit and explicit).
* Provide method for taking screenshots and logging.

**2. Page Classes (POM Implementation)**

Each page class corresponds to a section of the website and encapsulates Web-Elements and methods for interaction.

* LoginPage.java - Manages user authentication.
* SignupPage.java - Handles user registration workflows.
* HomePage.java - Represents homepage interactions.
* ProductPage.java - Selects and validates product details.
* CartPage.java - Implements **Page Factory Model** for Cart page functionality.
* Adds, removes, and updates items in the cart.
* CheckoutPage.java - Implements **Page Factory Model** for Checkout page functionality.
* Automates the checkout and payment process.

**3. Utility Classes**

* ExtentReportManager.java - Generates Extent Reports for comprehensive test execution logs.

**4. Test Cases**

* LoginTest.java - Validates the login functionality with valid and invalid credentials.
* SignupTest.java - Automates the signup process and verifies successful registration.
* AddToCartTest.java - Adds products to the cart and checks for consistency.
* CheckoutTest.java - Ensures a smooth checkout and payment workflow.
* TestCases.java - Runs end-to-end test scenarios covering the entire purchase cycle.

**5. Test Execution**

* **Test Execution via TestNG XML:** test.xml, testcases.xml define test execution order.

**Data**

* config.properties - Stores configurable values like browser type, application URL, and user credentials.

**Feature(Cucumber Feature File)**

* A Cucumber feature file that defines Login scenarios in Gherkin syntax.
* Includes test cases likeLogin with valid credentials and with invalid input credentials

**MyRunner(Cucumber Test Runner)**

* The runner class executes the feature files. I utilized TestRunner.java as the runner class. It informs Cucumber where the feature files and step definition files are. It also assists in executing the tests using TestNG and creating reports.

**StepDef(Step Definitions for Cucumber)**

* Step definition file ties every line from the feature file to real code. In my project, I have written LoginSteps.java inside the StepDef package. It contains the code that deals with the webpage objects such as providing Login credentials, clicking login, and verifying outcomes

**Challenges and Solutions**

**1. Synchronization Issues**

* **Problem:** Web elements were not loading within the expected time, causing NoSuchElementException.
* **Solution:** Implemented explicit waits (WebDriver Wait), Implicit wait and Thread.Sleep() to ensure elements load before interaction.

**2. Handling Dynamic Elements**

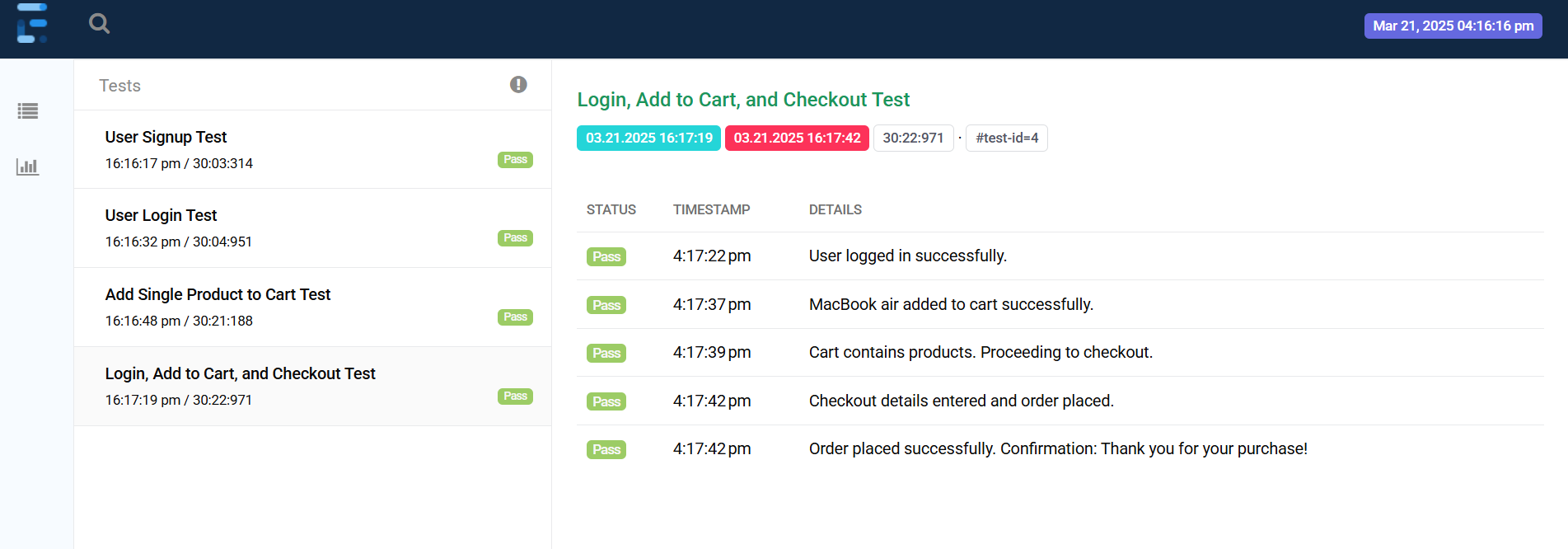
* **Problem:** Product listings and cart elements had dynamically generated attributes, making XPath unreliable.
* **Solution:** Used dynamic XPath and CSS selectors, incorporating attribute-based matching techniques.

**3. Exception Handling**

* Managing unexpected pop-ups, timeouts, and element not found exceptions.
* Implementing robust try-catch blocks for smooth test execution.

**Reports:**

**1.Extent Report:** Provides visually rich HTML test execution logs (ExtentReports-DemoBlaze.html).

****

**2.Cucumber Report:** Captures BDD-style test execution details (cucumber-reports.html).

**A screenshot of a phone

AI-generated content may be incorrect.**

**Conclusion**

The Selenium TestNG automation framework effectively automates the core functionalities of the DemoBlaze website. The integration of Page Object Model (POM), Page Factory, Cucumber BDD, and robust reporting mechanisms ensures maintainability, scalability, and efficient test execution. The project lays a solid foundation for further enhancements, including API automation, performance testing, and integration with CI/CD pipelines.