Case Study: Automation of E-Commerce Workflows Using Selenium WebDriver, TestNG, and POM

1 Introduction

This case study outlines a structured and practical approach to automating the end-to-end work-flow of a web-based e-commerce application, covering user registration and login, shopping by category, and the cart functionality.

Selenium WebDriver as the core automation tool. The framework is designed using the **Page Object Model (POM)**, which promotes modularity and re-usability by separating the test scripts from the page element definitions.

Java is used as the primary programming language for writing the automation scripts due to its strong object-oriented features, cross-platform compatibility, and seamless integration with Selenium.

For test execution management, **TestNG** is incorporated, providing advanced functionality such as annotations, parallel test execution, parameterization, and detailed reporting, which make the test suite scalable and maintainable.

The **Eclipse IDE** serves as the development environment, offering features like code suggestions, debugging tools, and Maven integration that simplify the overall development and debugging process.

2 Objective

The objective of this case study is to develop a robust and maintainable automation framework for a web-based e-commerce application Hafsaad. The focus is on automating critical end-to-end workflows such as user registration, login, shopping by category, cart summarization. By adopting the **Page Object Model (POM)** design pattern, the framework ensures modularity, scalability, and reusability of code.

The use of **TestNG** enhances test execution management by enabling annotations, parallel execution, and detailed reporting. Additionally, **Extent Reports** are integrated to provide comprehensive execution insights with screenshots and logs, making debugging easier. Overall, the objective is to deliver an efficient and reusable test automation solution that reduces manual effort, improves accuracy, and ensures consistent validation of application functionalities, while being flexible enough to support future enhancements.

3 Tools and Technologies Used

- **Selenium WebDriver** – Core automation tool to interact with web elements.
- **Java** – Primary programming language for scripting test cases.
- **TestNG** – Testing framework for managing test execution, annotations, reporting, and parallel execution.
- **Eclipse IDE** – Development and debugging environment offering Maven integration, code suggestions, and debugging support.

- **Maven** – Build tool to manage dependencies and run test suites efficiently.
- **Extent Reports** – For generating detailed, visually appealing HTML test execution reports.

4 Test Scenarios

✧ **TC001** - Register a new user with valid details
Expected Result - User account created successfully
✧ **TC002** - LogIn with the credentials used in registration
Expected Result - User login successful
✧ **TC003** - Navigate through a product category - Best Sellers
Expected Result - Relevant products are displayed
✧ **TC004** – Select a product from Best Sellers, select size and Add to Cart
Expected Result - Selected product with mentioned size added to cart
✧ **TC005** - Navigate through a product category - Sarees and move to Ready To Wear
Expected Result - Relevant products are displayed
✧ **TC006** - Select a product from Ready To Wear, select hip size and Add to Cart
Expected Result - Selected product with mentioned size added to cart
✧ **TC007** - Navigate through a product category - Blouses and move to Premium Blouses
Expected Result - Relevant products are displayed
✧ **TC008** - Select a product from Premium Blouses, select bust size, style and Add to Cart
Expected Result - Selected product with mentioned size and style added to cart
✧ **TC008** - Navigate through a product category - Celebrity Diaries
Expected Result - Relevant products are displayed
✧ **TC0010** - Select a product from Celebrity Diaries, select size, quantities, add wrap on option and Add to Cart
Expected Result - Selected product with mentioned size and quatity with wrap option added to cart
✧ **TC0011** - Navigate to Cart and Cart summary
Expected Result - All the added products and price details are displayed

5 Framework Design

```
HAFSAAD_AUTOMATION/
|
 ├── pom.xml
|   └── Maven dependencies
|      (Selenium, TestNG, WebDriverManager, Extent Reports)
|
 ├── src
|    ├── main
|    |   └── java
|    |       └── Base_class
|    |           └── Base.java
|    |              - WebDriver setup
|    |              - @BeforeClass, @BeforeMethod
|    |              - @AfterMethod, @AfterClass
|    |              - Extent Report configuration
|    |              - Popup handling
```

```
|
|   ├── test
|   |   └── java
|   |       ├── HafsaadPages
|   |       |   ├── HomePage.java
|   |       |   ├── CelebrityPage.java
|   |       |   ├── BlousesPage.java
|   |       |   ├── SareePage.java
|   |       |   └── CartPage.java
|   |       |     - Page Object Model classes
|   |       |
|   |       └── HafsaadTests
|   |           └── HafTest.java
|   |             - TestNG test cases
|   |             - Test execution flow
|
|   └── resources
|       └── testng.xml
|         - Test suite configuration
|
├── Report
|   └── AutomationReport.html
|     - Extent test execution report
|
├── screenshot
|   └── CartScreen.png
|     - Failure / validation screenshots
```

6 Maven Configuration

The pom.xml file declares dependencies for Selenium and TestNG, and integrates the Maven.

7 Test Script Overview

Base Class : The **Base Class** acts as the backbone of the automation framework and uses TestNG annotations to control the test flow. The **@BeforeClass** annotation is used to initialize the WebDriver, launch the browser, open the Hafsaad website, and configure Extent Reports—this runs once before any test methods in the class. The **@BeforeMethod** annotation runs before every test case to create a test entry in the report and handle any pop-up, ensuring a clean start for each test. The **@AfterMethod** annotation executes after every test to log the test result (Pass, Fail, or Skip) into the Extent Report. Finally, the **@AfterClass** annotation closes the browser and flushes the report after all tests are completed. By using these annotations, the Base Class ensures proper setup, execution, cleanup, and reporting in a structured and efficient way.

Page Classes : The **Page Classes** in this framework are designed following the **Page Object Model (POM)**, where each page of the e-commerce application is represented as a separate class containing web elements and methods that interact with those elements. This design ensures a clear separation between the test logic and the user interface, making the scripts reusable and

easier to maintain. By encapsulating locators and actions within these Page Classes, the framework achieves modularity, improves readability, and supports efficient automation of end-to-end scenarios.

Test Class : The **Test Classes** act as the execution layer of the framework, where the actual test cases are defined and orchestrated using **TestNG annotations**. Each Test Class makes use of the corresponding Page Classes to perform user actions, ensuring that the test logic remains clean and readable.

The @Test annotation in TestNG ensures proper sequencing and grouping of these scenarios, while priority are used to manage execution order where necessary. This separation of concerns allows the Test Classes to function as a driver layer that integrates seamlessly with the Base Class for setup and teardown, and with the Page Classes for actions, thereby achieving a structured, maintainable, and reusable automation framework.

8 Test Execution

The test was executed directly in Eclipse IDE by right clicking on the Mainproject_test.java class and selecting Run As > TestNG Test, or simply by clicking the green play button on the editor. This method is efficient for development-time feedback and integrates with Eclipse's TestNG view for result monitoring

9 Results Test Execution Summary

Class	Method	Start	Time (ms)
Default suite			
Default test — passed			
HafsaadTests.HafTest	Cart	1765819800768	2419
	OrderBlouse	1765819460197	150649
	OrderCelebritySaris	1765819630989	149637
	OrderSaree	1765819308184	131800
	orderBestSellers	1765819158799	129132

10 Conclusion

The automation framework built using **Selenium WebDriver, TestNG, Java, and POM** successfully demonstrates how modular and reusable test scripts can be developed for an e-commerce application. By covering key workflows such as registration, login, product search, checkout, blog validation, and address book management, the framework ensures both **functional correctness** and **regression stability**. The integration of **Extent Reports** provides detailed insights into execution results, while the use of **TestNG** makes test management and execution more efficient.
Overall, this case study highlights a **scalable, maintainable, and reliable automation approach** that can be easily extended as the application evolves.

11 Screenshot