

Case Study: Testing & Automation of Shopsy Online Shopping Website

1. Introduction

This case study presents a structured approach to testing the **Shopsy Online Shopping Website**, covering both **Manual Testing and Automation Testing** of critical e-commerce workflows.

The project focuses on validating key functionalities such as:

- ❑ Logo Verification
- ❑ Home Page Verification
- ❑ Navigation to Next Page
- ❑ Product Search Functionality
- ❑ Screenshot Capture for Test Evidence

Manual testing was performed to validate functional, integration, usability, and regression scenarios. Automation testing was implemented using **Selenium WebDriver, Java, TestNG, and Page Object Model (POM)** to automate repetitive regression test cases.

2. Objective

The objective of this case study is to develop a robust and maintainable automation framework for a web-based e-commerce application. The focus is on automating critical end-to-end workflows such as user registration, login, product search, shopping by category, checkout, blog validation, address book management, and final logout. By adopting the Page Object Model (POM) design pattern, the framework ensures modularity, scalability, and reusability of code.

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 report

4. Test Scenarios – Shopsy Automation Project

TC001 – Logo Verification

Description:

Verify that the Shopsy logo is displayed correctly on the homepage.

Steps:

1. Launch the Shopsy application.
2. Locate the logo element.
3. Check whether the logo is displayed.

Expected Result:

The Shopsy logo should be visible and properly aligned on the homepage.

TC002 – Home Page Verification

Description:

Verify that the homepage loads successfully with all key elements.

Steps:

1. Launch the application.
2. Verify the page title.
3. Validate presence of search bar, navigation menu, and banners.

Expected Result:

Homepage should load successfully with all important elements displayed.

TC003 – Navigation to Next Page

Description:

Verify that user can navigate to the next page by clicking a product or link.

Steps:

1. Click on any product or category.
2. Verify the URL changes.
3. Validate the page title.

Expected Result:

User should be successfully redirected to the respective page.

TC004 – Product Search**Description:**

Verify search functionality using valid keyword.

Steps:

1. Enter product name in search bar (e.g., “Shoes”).
2. Click on search button.
3. Verify search results.

Expected Result:

Relevant products related to the searched keyword should be displayed.

TC005 – Taking Screenshot of Page**Description:**

Capture screenshot of the page during test execution.

Steps:

1. Perform any action (e.g., search or navigation).
2. Use Selenium TakesScreenshot method.
3. Save screenshot in project folder.

Expected Result:

Screenshot should be captured and saved successfully.

5. Framework Design – Shopsy Automation Project

Shopsy_Automation_Project/

```
|  
|   └── pom.xml  
|  
└── src  
    |   └── main  
    |       |   └── java  
    |       |       └── base  
    |       |           └── Base.java  
    |       |  
    |       |  
    |       └── pages  
    |           |   └── HomePage.java  
    |           |   └── LogoPage.java  
    |           |   └── NavigationPage.java  
    |           |   └── SearchPage.java  
    |           |   └── ScreenshotUtil.java  
|  
|  
└── test  
    |   └── java  
    |       └── ShopsyTest.java  
|  
└── resources  
    └── testng.xml
```

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 serves as the foundation of the automation framework, managing the essential setup and teardown processes required for executing test scripts. It is responsible for WebDriver initialization using the @BeforeTest annotation, where the browser configuration and driver setup are defined. The @BeforeMethod annotation ensures that the application under test is launched before each test case, providing a clean state for execution. Similarly, the @AfterMethod annotation is used to close the browser or clean up resources after each test, ensuring that tests remain isolated and independent. By centralizing these core functions in the Base Class, the framework avoids code duplication, improves maintainability, and provides a structured way to handle browser sessions and application loading seamlessly

Perfect Akhila  I'll now update this **Page Classes section** completely based on your actual 5 test cases.

Here is the corrected version aligned with your Shopsy project:

Page Classes

The Page Classes in this framework are designed using the **Page Object Model (POM)** design pattern, where each web page of the Shopsy application is represented as a separate class containing web elements (locators) and methods to perform actions on those elements.

This approach ensures a clear separation between:

- Test logic (Test Class)
- Web elements and actions (Page Classes)

This improves code reusability, readability, and maintainability.

The implemented Page Classes cover the following test scenarios:

- **Logo Verification (TC001)** – Verifies that the Shopsy logo is displayed correctly on the homepage.
- **Home Page Verification (TC002)** – Validates homepage title, search bar, navigation menu, and key UI elements.
- **Navigation to Next Page (TC003)** – Ensures that clicking on a product or link redirects the user to the appropriate page.

- **Product Search (TC004)** – Validates search functionality by entering a keyword and verifying relevant search results.
- **Screenshot Capture (TC005)** – Captures screenshots during test execution for validation and reporting purposes.

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 rightclicking 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

8. Results – Test Execution Summary

The automation test suite was executed using **TestNG** in Eclipse IDE. All five test cases were executed successfully without any failures.

Test Suite	Total Tests Run	Passed	Failed	Skipped	Execution Time (ms)
-------------------	------------------------	---------------	---------------	----------------	----------------------------

Default Suite	5	0	0	—
---------------	---	---	---	---

Execution Details

- Total Test Cases Executed: **5**
- Passed: **5**
- Failed: **0**
- Skipped: **0**
- Status:  All test cases passed successfully.

10. Conclusion

The automation framework developed using **Selenium WebDriver**, **Java**, **TestNG**, and **Page Object Model (POM)** successfully demonstrates how modular and reusable test scripts can be implemented for an e-commerce application like Shopsy.

The framework covers important UI and functional validation scenarios including:

- Logo Verification
- Home Page Validation
- Navigation to Next Page
- Product Search Functionality
- Screenshot Capture

By automating these core functionalities, the project ensures application stability, UI correctness, and proper navigation flow.

The use of **POM design pattern** improves code reusability and maintainability, while **TestNG** enables structured test execution and reporting. Screenshot functionality provides additional test evidence for validation and debugging.

Overall, this project showcases a simple, scalable, and maintainable automation framework that can be further enhanced by adding more end-to-end scenarios such as cart, checkout, and user account management in future.

The screenshot shows a TestNG results report titled "Functional Testing". The interface is dark-themed. At the top, there are tabs for "Status" (with a green triangle icon), "Dashboard" (with a red circle icon), and "Search" (with a magnifying glass icon). The status bar indicates the date and time: "Dec 17, 2025 07:10:32 PM" and "3.1.2".

The main area is divided into two sections: "Tests" and "Steps". Both sections show a large green circular progress indicator. Below each indicator, the status is listed as "Pass".

The "Tests" section contains a summary: "5 test(s) passed", "0 test(s) failed, 0 others".

The "Steps" section contains a summary: "5 step(s) passed", "0 step(s) failed, 0 others".

Below these sections, a detailed table for "Logo verification" is shown:

Logo verification		
Status	Timestamp	Details
Dec 17, 2025 07:10:46 PM	Dec 17, 2025 07:10:46 PM	[Redacted]
Pass	7:10:46 PM	Test case Failedlogoverify

At the bottom of the table, there are several small colored icons: a blue circle, a green checkmark, a red circle with a minus sign, a yellow circle with a question mark, a green triangle, and a red X.