

Automation Testing on Swag Labs

ABSTRACTION

Swag labs Automation Testing using Selenium

Automation testing is an essential practice in modern software development, providing efficiency, reliability, and consistency in verifying the functionality of applications. This abstract presents the automation testing framework for Swag Labs, a web-based shopping application, using Selenium, an open-source automation tool. The primary objectives of this project are to ensure the stability and reliability of Swag Labs, enhance the speed of testing, reduce manual effort, and minimize overall testing costs.

Selenium, renowned for its versatility in browser automation, was chosen due to its compatibility with various browsers and operating systems, robust support for multiple programming languages, and active community support. The testing environment consists of Ubuntu 22.04.2 LTS, Google Chrome, and Visual Studio Code, with test scripts developed in JavaScript.

The scope of testing encompasses functional and regression testing across all modules of the application. Key functionalities tested include user login, product sorting, cart operations, product detail views, quantity modifications, and the checkout process. Test data comprising usernames, passwords, and user information was meticulously prepared to simulate real-world scenarios.

A systematic approach to test execution was adopted, beginning with the identification of test cases, script development, script execution, result analysis, and defect tracking. Potential risks such as environment-specific issues, data exposure, and changes in web elements were identified, with appropriate mitigation strategies implemented.

In conclusion, the automation testing framework for Swag Labs using Selenium significantly improves the reliability and efficiency of the application testing process. It ensures a high level of quality and performance, ultimately contributing to a better user experience and operational excellence.

CONTENTS

Abstract	i
Contents	ii

	Chapters	Page No
Chapter 1	Introduction	1
	1.1 Background	1
	1.2 Working of this model	2
	1.3 Applications	2
	1.4 Problem Statement	3
Chapter 2	Literature Survey	5
Chapter 3	System Analysis	6
	3.1 Hardware Requirements	6
	3.2 Software Requirements	6
Chapter 4	Implementation	7
	4.1 Importing libraries	7
	4.2 Initialization and setup	7
	4.3 Implementation of test cases	8
	4.4 Object creation and test case function call	8
Chapter 6	Application Interface	9
Chapter 7	Results and Discussions	12
	References	13

List of Figures

Figure No	Figure Name	Page No
5.1	Login page	09
5.2	Log In page Automation	09
5.3	Detecting Object	10
5.4	Main page Automation	10
5.5	Program Execution	11
5.6	Test cases	11

CHAPTER 1

INTRODUCTION

1.1 Background

Automation testing has become a cornerstone in software development and quality assurance, driven by the need for efficient, reliable, and rapid testing processes. As web applications continue to evolve in complexity and functionality, the demand for robust testing frameworks has grown significantly. Swag Labs, a fictional web-based shopping application used for testing and educational purposes, provides an ideal platform to explore and implement automation testing techniques.

Selenium, an open-source automation testing tool, has gained widespread acceptance in the industry due to its versatility and powerful capabilities. It supports multiple programming languages, including JavaScript, Java, C#, Python, and Ruby, making it accessible to a broad range of developers and testers. Selenium's ability to automate browser interactions allows for comprehensive testing of web applications across different browsers and operating systems, ensuring consistent user experiences.

The increasing adoption of Agile and DevOps methodologies in software development has further emphasized the importance of automation testing. These methodologies advocate for continuous integration and continuous delivery (CI/CD), where automated tests play a crucial role in maintaining code quality and accelerating release cycles. By integrating automation testing into the CI/CD pipeline, teams can identify and address issues early in the development process, reducing the risk of defects in production.

In the context of Swag Labs, automation testing aims to verify the application's core functionalities, such as user authentication, product sorting, cart operations, product detail views, quantity adjustments, and the checkout process. These functionalities are critical to the user experience and overall performance of the application. Automating these tests not only increases testing efficiency but also ensures that the application remains stable and reliable as new features and updates are introduced.

Given the dynamic nature of web applications, automation testing also addresses challenges related to changes in web elements and user data management. By organizing web locators dynamically and creating test-specific user data, the testing framework can adapt to changes and minimize the risk of errors.

1.2 Working of this Model

Selenium WebDriver: The core component for interacting with web elements. Supports multiple programming languages (including Java). Allows you to automate browser actions such as clicks, form submissions, and navigation.

Selenium IDE (Integrated Development Environment): A record-and-playback tool suitable for beginners. Less powerful than WebDriver but useful for quick test creation. Enables parallel test execution across different browsers and machines. Set up an Integrated Development Environment (IDE) like Eclipse or IntelliJ IDEA.

Adding Dependencies: Add Selenium WebDriver JAR files to your project. Configure build paths to include these dependencies. **Writing Test Scripts:** Use Java to write test scripts using WebDriver methods. Create scenarios (e.g., login, form submission, navigation). Employ assertions to verify expected outcomes.

Locating Web Elements: Locators: Use various locators (IDs, class names, XPath, CSS selectors) to identify web elements.

Examples:

```
driver.findElement(By.id("elementId")),  
driver.findElement(By.xpath("//input[@name='username']"))
```

Running Tests: Browser Execution: Execute test scripts on different browsers (Chrome, Firefox, etc.). Handle exceptions (e.g., element not found, timeout). **Test Management:** Use TestNG or JUnit for organizing test cases into suites and generating reports.

1.3 Applications

1. Web Application Testing:

- Selenium is widely used for automating web application testing across different browsers (such as Chrome, Firefox, Edge, and Safari) and platforms. It allows testers to create automated test cases and execute them efficiently.

2. Cross-Browser Testing:

- Selenium's cross-browser compatibility ensures that web applications work consistently across different browsers. It helps identify browser-specific

issues and ensures a seamless user experience.

3. Regression Testing:

- Selenium is valuable for regression testing, where existing functionality is tested after code changes or updates. Automated regression tests help catch unintended side effects.

4. Continuous Integration and Delivery (CI/CD):

- Selenium integrates well with CI/CD pipelines, allowing automated tests to run as part of the development process. This ensures early detection of issues and faster feedback.

5. Data-Driven Testing:

- Selenium supports data-driven testing, where test cases are executed with different input data. This is useful for testing scenarios with varying parameters.

6. Parallel Test Execution:

- Selenium enables parallel execution of test cases, improving efficiency and reducing test execution time.

7. Automated UI Testing:

- Selenium automates interactions with web elements (buttons, forms, links, etc.) to verify that the user interface behaves correctly.

8. Integration with Frameworks and Tools:

- Selenium can be integrated with testing frameworks (e.g., TestNG, JUnit) and tools (e.g., Maven, Jenkins) to enhance test management and reporting.

9. Electrical and Optical Applications:

- Selenium is used in electrical and optical applications, including photoelectric and photovoltaic cells. Its properties make it suitable for these technologies.

1.4 Problem Statement

To automate and test the Swag Labs web application using the Cypress inbuilt tools with the following actions:

- **Create a Cypress Test Suite:**
 - Set up the Cypress testing environment and create a test suite for Swag Labs.
- **Navigate to the Swag Labs Web Page:**
 - Open the Swag Labs website and ensure it loads correctly.
- **Locate Web Elements on the Navigated Page:**
 - Identify and interact with various web elements such as input fields, buttons, product listings, and the shopping cart using Cypress commands.
- **Perform Actions on Located Elements:**
 - Execute actions like logging in, sorting products, adding/removing items from the cart, viewing product details, editing product quantities, and proceeding through the checkout process.
- **Assert the Performed Actions:**
 - Validate that the actions performed on the web elements produce the expected outcomes, such as successful login, correct product sorting, accurate item addition/removal in the cart, and correct order completion during checkout.
- **End the Test Session:**
 - Conclude the test session by logging out or closing the browser, ensuring all resources are properly released.
- **Report the Results of the Assertions:**
 - Generate detailed test reports that capture the results of the assertions, highlighting any discrepancies or issues encountered during the testing process.

CHAPTER 2

LITERATURE SURVEY

1. Scope and Objectives:

- **Scope:** The scope of this survey is to automate and test the Swag Labs web application using Selenium. The focus is on improving the efficiency, reliability, and coverage of testing processes.
- **Objectives:** The objectives are to:
 - Summarize existing research on automation testing using Selenium.
 - Identify emerging trends in automation testing.
 - Provide practical insights for testers and developers.

2. Components of Selenium:

- The different components of Selenium, such as Selenium IDE, Selenium WebDriver (Selenium 2), and Selenium Grid.

3. Comparison with Other Tools:

- Other popular testing tools are Watir, QTP, TestComplete, etc.
- Highlight strengths, weaknesses, and use cases for each tool.

4. Challenges and Trends:

- Handling dynamic web elements, synchronization issues, Integration with CI/CD Pipelines, Cross-Browser Testing.

CHAPTER 3

SYSTEM ANALYSIS

3.1 Hardware Requirements

1. Sufficient RAM and CPU:

- While Selenium itself doesn't have high hardware requirements, ensure your system has enough RAM and CPU resources to handle the browsers and tests efficiently.
- For distributed setups (like Selenium Grid), consider the combined resources of all nodes.
- CPU - Intel dual core processor or higher
- RAM – 4gb or higher

3.2 Software Requirements

- Java 11 or Higher:
 - Ensure that Java 11 or a more recent version is installed on your system. Selenium relies on Java for its execution.
- Installed Browsers:
 - Make sure you have the browsers (such as Chrome, Firefox, Edge, etc.) installed on your machine. Selenium interacts with these browsers during testing.
- Browser Drivers:

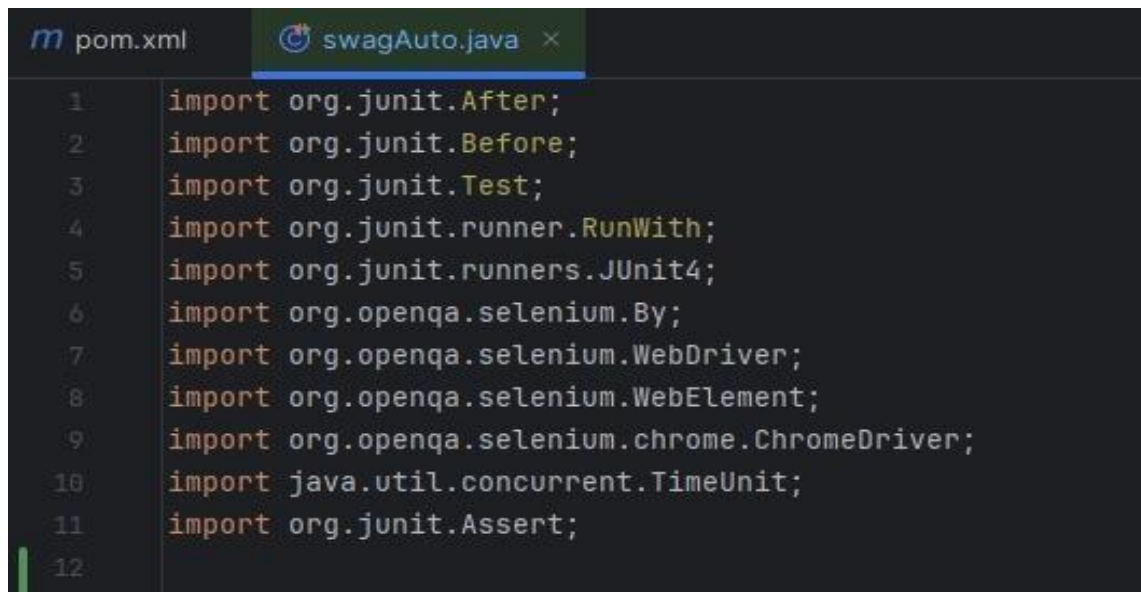
Install the appropriate browser drivers (e.g., ChromeDriver, GeckoDriver) for the browsers you intend to automate. These drivers allow Selenium to control the browsers programmatically.

CHAPTER 4

IMPLEMENTATION

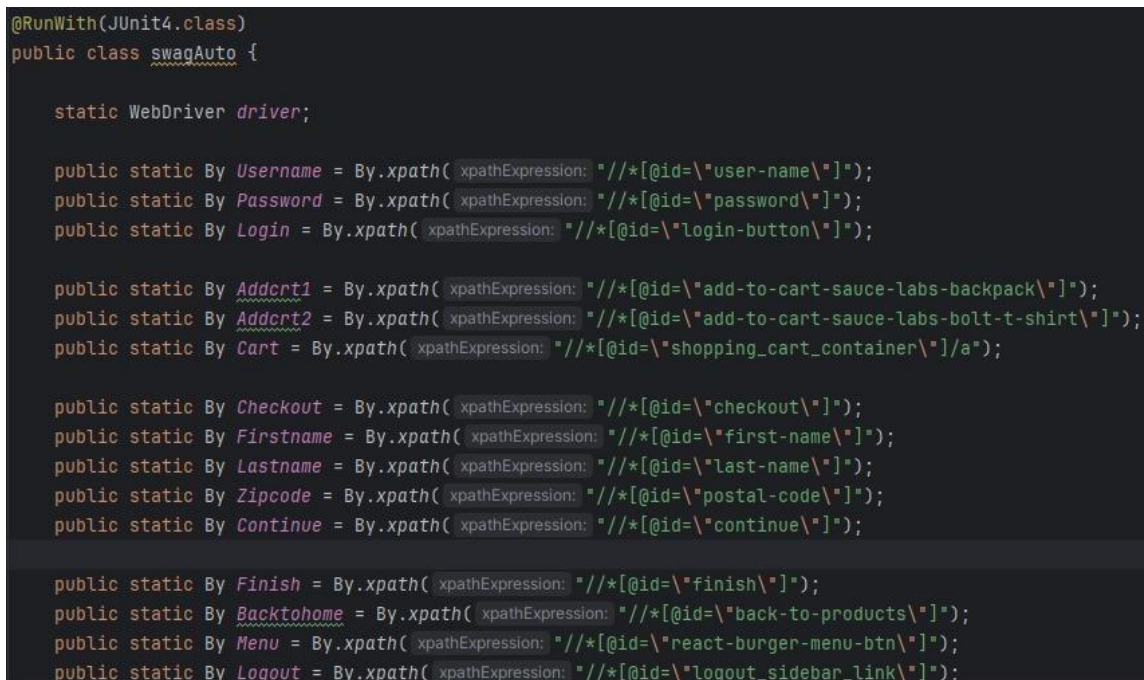
The initial setup is to update or download the latest web drivers and required libraries. The dependencies must be included in the pom.xml file of the project since the project uses maven build system.

4.1 Importing libraries :-



```
m pom.xml swagAuto.java x
1 import org.junit.After;
2 import org.junit.Before;
3 import org.junit.Test;
4 import org.junit.runner.RunWith;
5 import org.junit.runners.JUnit4;
6 import org.openqa.selenium.By;
7 import org.openqa.selenium.WebDriver;
8 import org.openqa.selenium.WebElement;
9 import org.openqa.selenium.chrome.ChromeDriver;
10 import java.util.concurrent.TimeUnit;
11 import org.junit.Assert;
12
```

4.2 Initialization and setup:-



```
@RunWith(JUnit4.class)
public class swagAuto {

    static WebDriver driver;

    public static By Username = By.xpath(xpathExpression: "//*[@id=\"user-name\"]");
    public static By Password = By.xpath(xpathExpression: "//*[@id=\"password\"]");
    public static By Login = By.xpath(xpathExpression: "//*[@id=\"login-button\"]");

    public static By Addcart1 = By.xpath(xpathExpression: "//*[@id=\"add-to-cart-sauce-labs-backpack\"]");
    public static By Addcart2 = By.xpath(xpathExpression: "//*[@id=\"add-to-cart-sauce-labs-bolt-t-shirt\"]");
    public static By Cart = By.xpath(xpathExpression: "//*[@id=\"shopping_cart_container\"]/a");

    public static By Checkout = By.xpath(xpathExpression: "//*[@id=\"checkout\"]");
    public static By Firstname = By.xpath(xpathExpression: "//*[@id=\"first-name\"]");
    public static By Lastname = By.xpath(xpathExpression: "//*[@id=\"last-name\"]");
    public static By Zipcode = By.xpath(xpathExpression: "//*[@id=\"postal-code\"]");
    public static By Continue = By.xpath(xpathExpression: "//*[@id=\"continue\"]");

    public static By Finish = By.xpath(xpathExpression: "//*[@id=\"finish\"]");
    public static By Backtohome = By.xpath(xpathExpression: "//*[@id=\"back-to-products\"]");
    public static By Menu = By.xpath(xpathExpression: "//*[@id=\"react-burger-menu-btn\"]");
    public static By Logout = By.xpath(xpathExpression: "//*[@id=\"logout_sidebar_link\"]");
}
```

```

@Before  ⚡ karanyadachi
public void setUp() {
    System.setProperty("webdriver.chrome.driver", "F:\\Swag\\chromedriver-win64\\chromedriver.exe");
    driver = new ChromeDriver();
    driver.get("https://www.saucedemo.com/");
    driver.manage().window().maximize();
    driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
}

@After  ⚡ karanyadachi
public void tearDown() {
    if (driver != null) {
        driver.quit();
    }
}

```

4.3 Implementation of test cases

```

@Test  ⚡ karanyadachi
public void logoTest() {
    WebElement Logo = driver.findElement(By.xpath(xpathExpression: "//*[@id=\"root\"]/div/div[1]"));
    Assert.assertTrue(Logo.isDisplayed());
}

@Test  ⚡ karanyadachi
public void productPageTest() throws InterruptedException {...}

@Test  ⚡ karanyadachi
public void cartTest() throws InterruptedException {...}

@Test  ⚡ karanyadachi
public void logoutTest() {
    String expectedLogTitle = "Swag Labs";
    String actualLogTitle = driver.getTitle();
    Assert.assertEquals(message: "Logout page title matched", expectedLogTitle, actualLogTitle);
}

```

4.4 Object creation and test case function call

```

public static void main(String[] args) { ⚡ karanyadachi
    swagAuto s=new swagAuto();
    try {
        s.setUp();

        login();
        addToCart();
        checkout();
        logout();

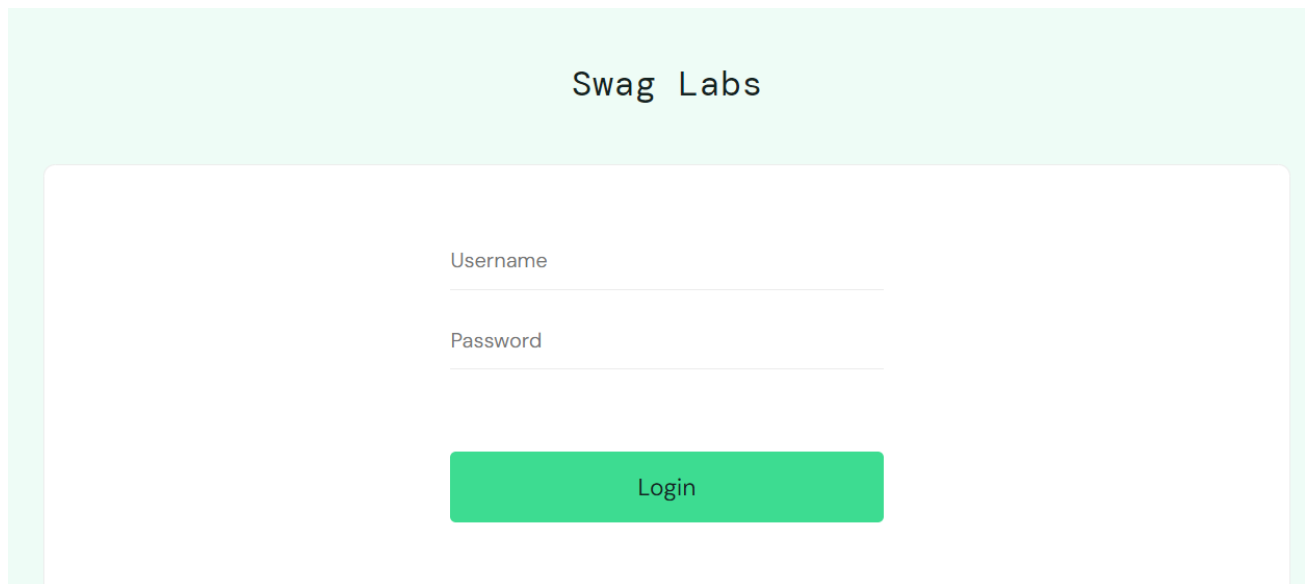
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    } finally {
        s.tearDown();
    }
}
}

```

CHAPTER 5

APPLICATION INTERFACE

Results:



The image shows the Swag Labs login page. It has a light green header with the text "Swag Labs". Below the header is a white login form with two input fields: "Username" and "Password". Below the password field is a green "Login" button.

Fig 5.1 Log In page

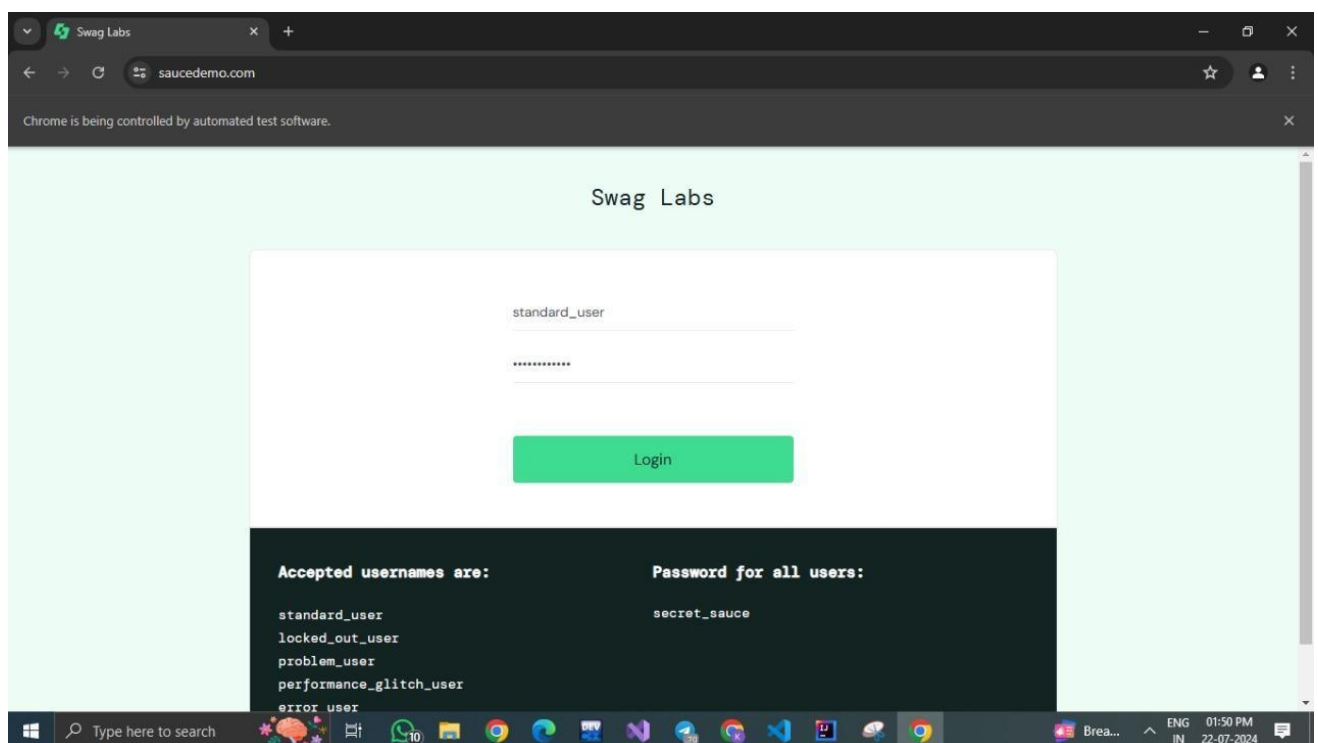


Fig 5.2 Log In page Automation

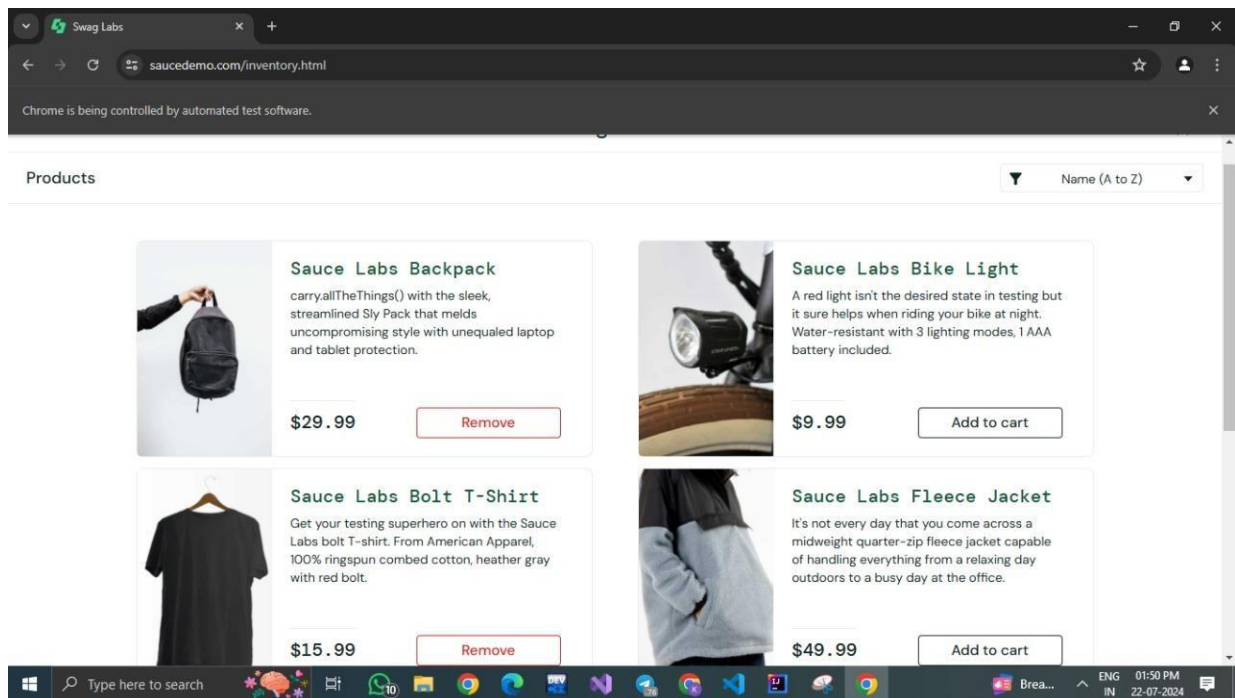


Fig 5.3 Main Page

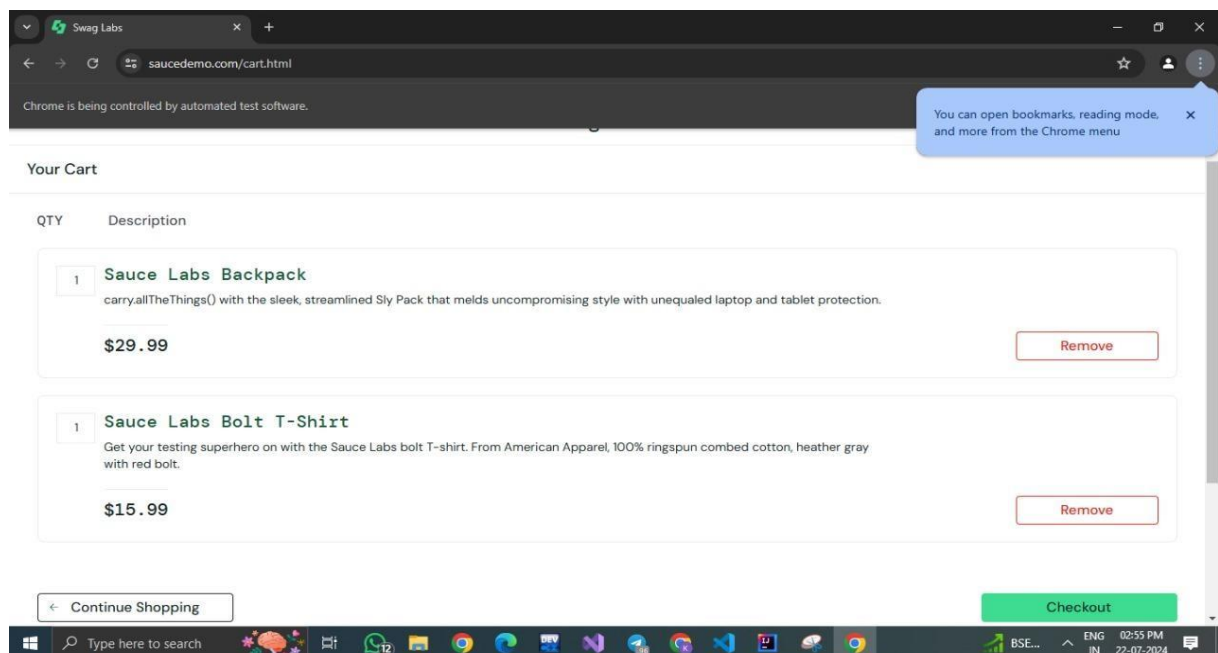


Fig 5.4 Main Page Automation

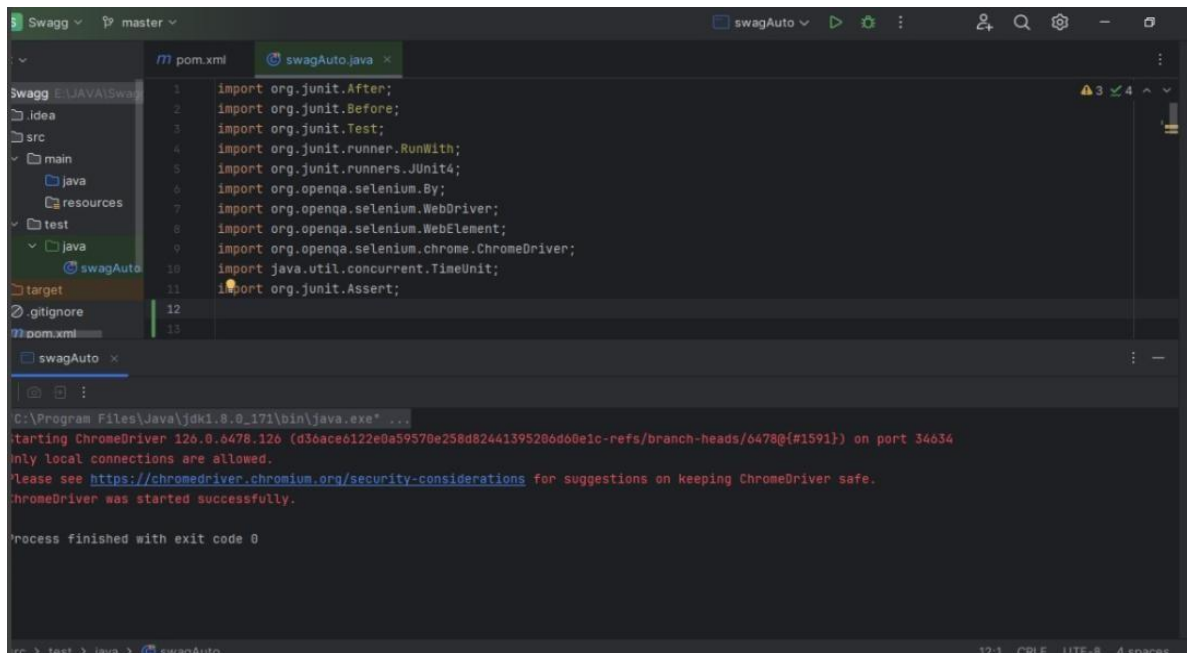


Fig 5.5 Program Execution

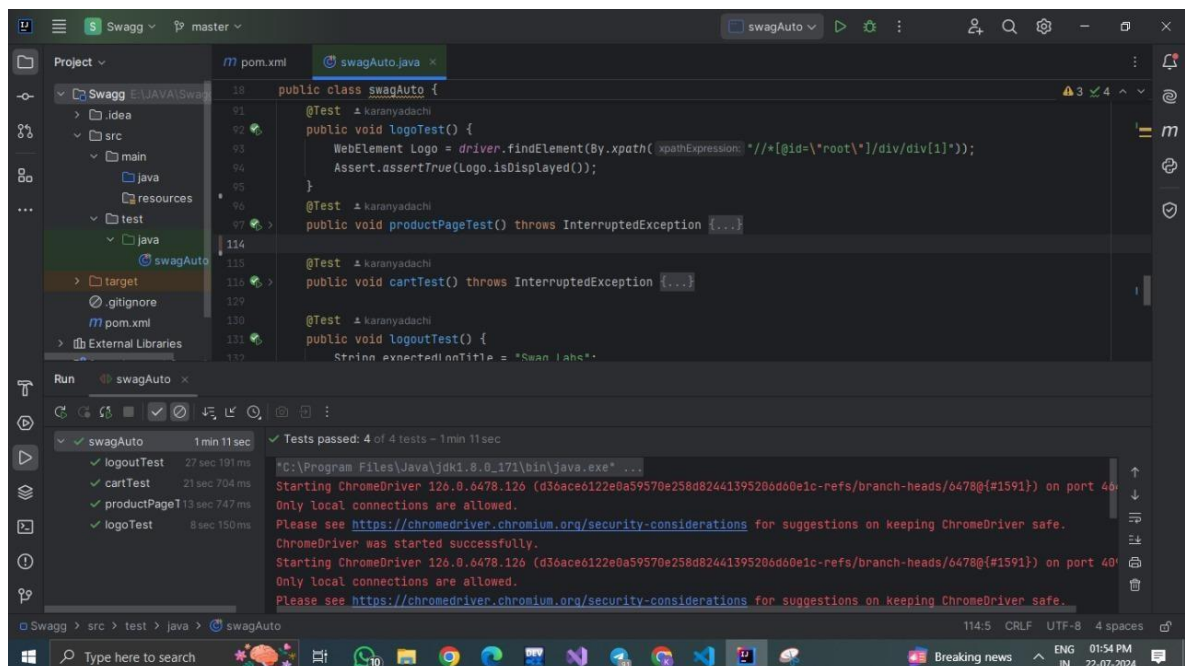


Fig 5.6 Test cases

CHAPTER 6

RESULTS AND DISCUSSION

Results:

1. **logoutTest():**
 - **Status:** Passed
 - **Execution Time:** 8 sec
2. **cartTest():**
 - **Status:** Passed
 - **Execution Time:** 7 sec
3. **productPageTest():**
 - **Status:** Passed
 - **Execution Time:** 7 sec
4. **logoTest():**
 - **Status:** Passed
 - **Execution Time:** 8 sec

Discussion:

The automation testing suite for Swag Labs executed a total of 4 test cases, each of which passed successfully. The tests were efficient, with execution times ranging from 7 to 8 seconds.

- **Class Coverage:** Achieving 100% class coverage (1/1) indicates that all classes within the scope of testing were executed during the test run. This implies that the entire code base was considered during the testing phase.
- **Method Coverage:** The method coverage of 100% (11/11) suggests that every method in the classes under test was invoked during the test execution. This ensures that all functionalities provided by the methods were verified.
- **Lines of Code Coverage:** The 100% line coverage (54/54) demonstrates that every line of code in the tested classes was executed during the tests. This level of coverage ensures that no part of the code was left untested, increasing the reliability of the application.
- **Branch Coverage:** Although the branch coverage is listed as 100%, it shows 0/0, indicating that there were no conditional branches in the code to be tested. This might imply that the tested methods did not contain conditional statements or branches.

REFERENCES

1. Selenium WebDriver 3 Practical Guide, Second Edition:

This book provides end-to-end automation testing solutions for web and mobile browsers using Selenium WebDriver 3. It covers topics like cross-browser testing, advanced actions (e.g., drag-and-drop), and using Java 8 API with Selenium 3.

2. Selenium with Java – A Beginner’s Guide:

Written by Pallavi Sharma, this ebook focuses on web browser automation for testing using Selenium with Java. It’s suitable for beginners and covers essential concepts.

3. Java Testing with Selenium: A Comprehensive Syntax Guide for Automation:

This comprehensive guide helps both beginners and experienced testers build robust and maintainable test suites using Selenium with Java. It emphasizes quality and reliability for web applications.

4. Hands-On Selenium WebDriver with Java:

Author Boni Garcia takes Java developers through Selenium’s main features, including web navigation, browser manipulation, and web element interaction. The book includes ready-to-be-executed test examples.

5. Mastering Selenium WebDriver 3.0 - Second Edition:

While not an ebook, this book is worth mentioning. It provides practical guidance on using Selenium WebDriver 3.0 with Java for automating web applications.

6. A workshop conducted by the department of Information Science and Engineering, CITECH- Bangaluru.