

SauceDemo Web Application Test Documentation

Table of Contents

1. Introduction

- 1.1 Purpose of Test Automation
- 1.2 Scope of the Project
- 1.3 Objectives

2. Test Automation Approach

- 2.1 Explanation of the Page Object Model (POM)
- 2.2 Technology Stack
 - Selenium with Java
 - Cucumber BDD Framework

3. Test Scenarios

- 3.1 Scenario Overview
- 3.2 Step-by-Step Description of Scenarios
 - Login to the website
 - Select specific items from the Product page
 - Add items to the cart
 - Proceed to the checkout flow
 - Provide required information on checkout pages
 - Complete the order

4. Test Implementation

- 4.1 Project Structure Overview
 - Explanation of Packages and Class Names
- 4.2 Purpose of Java Classes
 - LoginPage
 - ProductsPage
 - CartPage
 - Checkout Information, Overview, and Complete Pages
- 4.3 Implementation Details
 - Locators Used
 - Actions Performed
 - Custom Methods and Utilities

5. Test Execution

- 5.1 Using the TestRunner.java Class
- 5.2 Running Tests in the Local Environment
 - Prerequisites
 - Step-by-Step Instructions

6. Test Reporting

- **6.1 Reporting Tools or Frameworks Used**
- **6.2 Accessing Test Results**

7. Conclusion

- **7.1 Summary of Automation Outcomes**
- **7.2 Future Enhancements**

8. Annexures

- **8.1 Test Scenarios and Gherkin Scripts**
- **8.2 Sample Code Snippets**
- **8.3 Environment Configuration Details**

1. Introduction

• 1.1 Purpose of Test Automation

The project, **Automated Testing of the SauceDemo E-Commerce Web Application**, aims to validate the core user flows and functionality of the SauceDemo platform. By implementing test automation, the project ensures that critical processes work as expected without manual intervention.

This approach focuses exclusively on **happy path scenarios**—the most common and successful user journeys—to guarantee smooth operations during key interactions with the platform. Automation allows repetitive tests to be run efficiently and reliably, reducing the likelihood of human error.

URL - <https://www.saucedemo.com>

• 1.2 Scope of the Project

^This project is designed to:

- Validate critical functionalities including **Login, Product Catalog, Cart, and Checkout** pages.
- Ensure a consistent and reliable user experience across happy path scenarios.
- Serve as a foundational suite for future automation and regression testing.

• 1.3 Objectives

The objectives of the test automation project are:

1. **Reduce Manual Effort:** Automate repetitive testing tasks to improve efficiency and consistency.
2. **Increase Test Coverage:** Focus on validating end-to-end user flows for the most critical functionalities.
3. **Enhance Reliability:** Ensure that core processes operate as expected in different environments.
4. **Improve Scalability:** Build a framework that can accommodate future test cases and scenarios.

2. Test Automation Approach

- **2.1 Explanation of the Page Object Model (POM)**

The **Page Object Model (POM)** is a design pattern used in this project to enhance the readability, reusability, and maintainability of test automation scripts. In POM, each web page of the application is represented as a separate Java class, encapsulating the UI elements and their interactions.

Key benefits of using POM in this project:

- **Separation of Concerns:** Keeps test scripts clean by separating test logic from page interactions.
- **Reusability:** Common methods, such as filling out forms or clicking buttons, are defined once and can be reused across multiple tests.
- **Maintainability:** Changes to UI elements are localized to the corresponding page class, reducing the effort required to update scripts when the application UI changes.

For the SauceDemo application, each core page (e.g., Login Page, Products Page, Cart Page) is implemented as a Java class. These classes define:

- **Locators:** Identifiers for web elements, such as IDs, XPaths, or CSS selectors.
- **Methods:** Actions that can be performed on the page, such as logging in, selecting products, or completing the checkout process.

- **2.2 Technology Stack**
 - **Selenium with Java**
 - **Cucumber BDD Framework**

This project leverages the following technologies to implement the test automation framework:

- **Selenium with Java:**
 - Selenium is a powerful web automation tool that interacts with web elements on the application.
 - Java is used as the programming language for writing scripts and implementing the Page Object Model.
- **Cucumber BDD Framework:**
 - Cucumber is used to write test cases in a **Behavior-Driven Development (BDD)** format using Gherkin syntax.
 - Gherkin scenarios are written in a natural language style, making them accessible to both technical and non-technical stakeholders.
 - Cucumber integrates seamlessly with Selenium, connecting feature files with Java-based step definitions.

- **IDE:** Eclipse Version 2024-06 (4.32.0)
- **Programming Language:** Java
- **Java Version:** JDK 22
- **Testing Framework:** JUnit 4.13
- **Build Tool:** Maven 3.10.1
- **Automation Tool:** Selenium WebDriver
- **Browser Driver:** ChromeDriver Version 131.0.6778.86
- **BDD Framework:** Cucumber BDD 7.3.4
- **Scripting Language:** Gherkin (for writing feature files)

3. Test Scenarios

• 3.1 Scenario Overview

The test automation project focuses on validating the **happy path scenarios** for the SauceDemo E-Commerce Web Application. These scenarios cover the core functionalities required for a seamless shopping experience, ensuring the platform operates as expected for end-users.

The key scenarios include:

1. Logging into the website with valid credentials.
2. Selecting specific items from the product catalog.
3. Adding the selected items to the shopping cart.
4. Proceeding to the checkout process.
5. Providing the required customer information during checkout.
6. Completing the order successfully.

• 3.2 Step-by-Step Description of Scenarios

Here is a detailed description of each test scenario:

1. **Login to the Website**
 - **Given:** The user is on the SauceDemo login page.
 - **When:** They enter valid credentials (username and password).
 - **And:** They click the "Login" button.
 - **Then:** They are redirected to the Products page.
2. **Select Specific Items from the Product Page**
 - **Given:** The user is on the Products page.
 - **When:** They add specific items (e.g., "Sauce Labs Backpack" and "Sauce Labs Bike Light") to the cart.
 - **Then:** The items are successfully added to the cart.
3. **Add Items to the Cart**
 - **Given:** The user has selected items from the Products page.
 - **When:** They navigate to the Cart page.
 - **Then:** The selected items are displayed in the cart with correct quantities and prices.

4. Proceed to the Checkout Flow

- **Given:** The user is on the Cart page with items added.
- **When:** They click the "Checkout" button.
- **Then:** They are redirected to the "Checkout Information" page.

5. Provide Required Information on Checkout Pages

- **Given:** The user is on the "Checkout Information" page.
- **When:** They enter their details (e.g., first name, last name, and postal code).
- **And:** They proceed to the "Checkout Overview" page.
- **Then:** The order details are displayed, including item summary and total cost.

6. Complete the Order

- **Given:** The user is on the "Checkout Overview" page.
- **When:** They confirm the order by clicking the "Finish" button.
- **Then:** They are redirected to the "Checkout Complete" page with a success message.

4. Test Implementation

• 4.1 Project Structure Overview

○ Explanation of Packages and Class Names



- **4.2 Purpose of Java Classes**

The following Java classes are included in the **saucedemoPages** package to represent the Page Object Model (POM). Each class encapsulates the web elements and actions for its respective page:

- **LoginPage.java**

- Represents the **Login Page** functionality.
- **Locators:** Fields for username, password, and login button.

```
8
9 public class LoginPage {
10
11     private WebDriver driver;
12
13     //Locators for user name, password, and login button
14     @FindBy(id = "user-name")
15     private WebElement usernameInput;
16
17     @FindBy(id = "password")
18     private WebElement passwordInput;
19
20     @FindBy(id = "login-button")
21     private WebElement loginButton;
22
```

- **Actions:**
 - Entering credentials.
 - Clicking the login button.
- **Purpose:** Provides methods for automating login interactions.

```
37
38 //Enter the user name
39 public void usernameInput(String username) {
40     usernameInput.clear();
41     usernameInput.sendKeys(username);
42     System.out.println("Username entered: " + username);
43 }
44
45 //Enter the password
46 public void passwordInput(String password) {
47     passwordInput.clear();
48     passwordInput.sendKeys(password);
49     System.out.println("Password entered: " + password);
50 }
51
52 //Click Login button
53 public void clickLoginButton() {
54     loginButton.click();
55     System.out.println("Login button clicked.");
56 }
57 }
58
59
```


- **ProductsPage.java**

- Represents the **Products Page** functionality.
- **Locators:** Product names, add-to-cart buttons, and navigation to the cart.

```
13
14 public class ProductPage {
15
16     private WebDriver driver;
17
18     //Locators for WebElements
19     @FindBy(className = "inventory_list")
20     private WebElement inventoryList;
21
22     @FindBy(className = "shopping-cart-badge")
23     private WebElement shopping_cart_link;
24
25     @FindBy(xpath = "//button[@id='add-to-cart-sauce-labs-bolt-t-shirt']")
26     private WebElement boltTShirtButton;
27
28     @FindBy(xpath = "//*[@id='add-to-cart-test.allthethings()-t-shirt-(red)']")
29     private WebElement addToCartBoltTShirt;
30
31     @FindBy(id = "shopping_cart_container")
32     private WebElement cartButton;
33 }
```

- **Actions:**
 - Selecting products.
 - Navigating to the cart.
- **Purpose:** Allows interaction with the product catalog.

```
39
40 //get inventory list
41 public WebElement getInventoryList() {
42     return inventoryList;
43 }
44
45 //add Item to cart
46 public void addItemToCart(String itemId) {
47
48     //Add an item to the cart using explicit wait
49     WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(1000));
50     WebElement addToCartButton = wait.until(ExpectedConditions.elementToBeClickable(By.id(itemId)));
51     addToCartButton.click();
52 }
53
54 //Click Cart Button
55 public void clickCartButton() {
56     cartButton.click();
57     System.out.println("Navigated to the cart");
58 }
59 }
60 }
```

- **CartPage.java**

- Represents the **Cart Page** functionality.
- **Locators:** Items in the cart, quantity, and checkout button.

```
12
13 public class CartPage {
14
15     private WebDriver driver;
16     private WebDriverWait wait;
17
18     //Locators for WebElements
19     @FindBy(id = "checkout")
20     private WebElement checkoutButton;
21
22     @FindBy(className = "cart_item")
23     private List<WebElement> cartItems;
24
```

- **Actions:**
 - Validating items in the cart.
 - Proceeding to checkout.
- **Purpose:** Provides methods for validating and interacting with cart contents.

```
31
32 //Click CheckoutButton
33 public void clickCheckoutButton() {
34     // wait.until(ExpectedConditions.elementToBeClickable(checkoutButton)).click();
35     checkoutButton.click();
36 }
37
38 //Verify if on the cart page
39 public boolean isCartPage() {
40     return driver.getCurrentUrl().contains("/cart.html");
41 }
42
43 //Get the count of items in the cart
44 public int getCartItemCount() {
45     wait.until(ExpectedConditions.visibilityOfAllElements(cartItems));
46     return cartItems.size();
47 }
48
49 // Check if the cart is empty
50 public boolean isCartEmpty() {
51     return getCartItemCount() == 0;
52 }
53 }
54
```

- **CheckoutInformationPage.java**

- Represents the **Checkout Information Page** functionality.
- **Locators:** Input fields for first name, last name, postal code, and continue button.

```

11
12 public class CheckoutInformationPage {
13     private WebDriver driver;
14     private WebDriverWait wait;
15
16     //Locators for WebElement
17     @FindBy(id="first-name")
18     private WebElement firstNameInput;
19
20     @FindBy(id = "last-name")
21     private WebElement lastNameInput;
22
23     @FindBy(id = "postal-code")
24     private WebElement postalCodeInput;
25
26     @FindBy(id="continue")
27     private WebElement continueInfoButton;
28

```

- **Actions:**
 - Filling in customer information.
 - Proceeding to the order overview page.

```

34
35 //Enter Customer Details with Explicit time
36 public void enterCustomerInformation(String firstName, String lastName, String zipCode) {
37     wait.until(ExpectedConditions.visibilityOf(firstNameInput));
38     firstNameInput.clear();
39     firstNameInput.sendKeys(firstName);
40
41     wait.until(ExpectedConditions.visibilityOf(lastNameInput));
42     lastNameInput.clear();
43     lastNameInput.sendKeys(lastName);
44
45     wait.until(ExpectedConditions.visibilityOf(postalCodeInput));
46     postalCodeInput.clear();
47     postalCodeInput.sendKeys(zipCode);
48
49     System.out.println("Customer information entered: " + firstName + " " + lastName + " " + zipCode);
50 }
51
52 //Click Continue Button
53 public void clickContinue() {
54     continueInfoButton.click();
55 }
56 }
57

```

- **CheckoutOverViewPage.java**

- Represents the **Checkout Overview Page** functionality.
- **Locators:** Item details, total cost, and finish button.

```

10
11
12 public class CheckOutOverViewPage {
13     private WebDriver driver;
14     private WebDriverWait wait;
15
16     @FindBy(id="finish")
17     private WebElement finishButton;
18

```

- **Actions:**

- Verifying order summary.
- Completing the order.

```

24
25
26 public void clickfinish() {
27     finishButton.click();
28 }
29
30
31 }
32

```

- **CheckoutCompletePage.java**

- Represents the **Checkout Complete Page** functionality.
- **Locators:** Success message and order confirmation details.

```

11
12 public class CheckoutCompletePage {
13
14     private WebDriver driver;
15
16     //Locators for WebElements
17     @FindBy(className = "title")
18     private WebElement completetitle;
19
20     @FindBy(id = "back-to-products")
21     private WebElement backHomeButton;
22
23     @FindBy (className = "title")
24     private WebElement displayproductpage;
25

```

- **Actions:**
 - Verifying the success message.
- **Purpose:** Ensures the checkout process completes successfully.

```

26 public CheckoutCompletePage(WebDriver driver) {
27     this.driver = driver;
28     PageFactory.initElements(driver, this);
29 }
30
31 //Click Home Page Button
32 public void clickBackHomeButton() {
33     backHomeButton.click();
34 }
35
36 //Display Product Page
37 public void displayproductpage() {
38     displayproductpage.click();
39 }
40 }
41

```

5. Test Execution

• 5.1 Using the TestRunner.java Class

The TestRunner.java class in the **TestRunner** package is used to execute the test cases defined in the Cucumber feature files. This class integrates the following components:

- **Cucumber Options:** Specifies feature file locations, step definition paths, and reporting formats.
- **JUnit Runner:** Executes the tests using JUnit.

```

J Runner.java ×
1 package TestRunner;
2
3
4 import org.junit.runner.RunWith;
5 import io.cucumber.junit.Cucumber;
6 import io.cucumber.junit.CucumberOptions;
7
8 @RunWith(Cucumber.class)
9 @CucumberOptions(
10     features = "src/test/resources/Features",
11     glue = {"StepsDefinition","utils"},
12     plugin = {"pretty",
13         "html:target/cucumber-reports/report.html",
14         "json:target/cucumber-reports/cucumber.json",
15         "junit:target/cucumber-reports/cucumber.xml",
16         "rerun:target/cucumber-reports/rerun.txt"
17     },
18     monochrome = true,
19     publish = false
20 )
21 public class Runner {
22
23 }
24
25

```

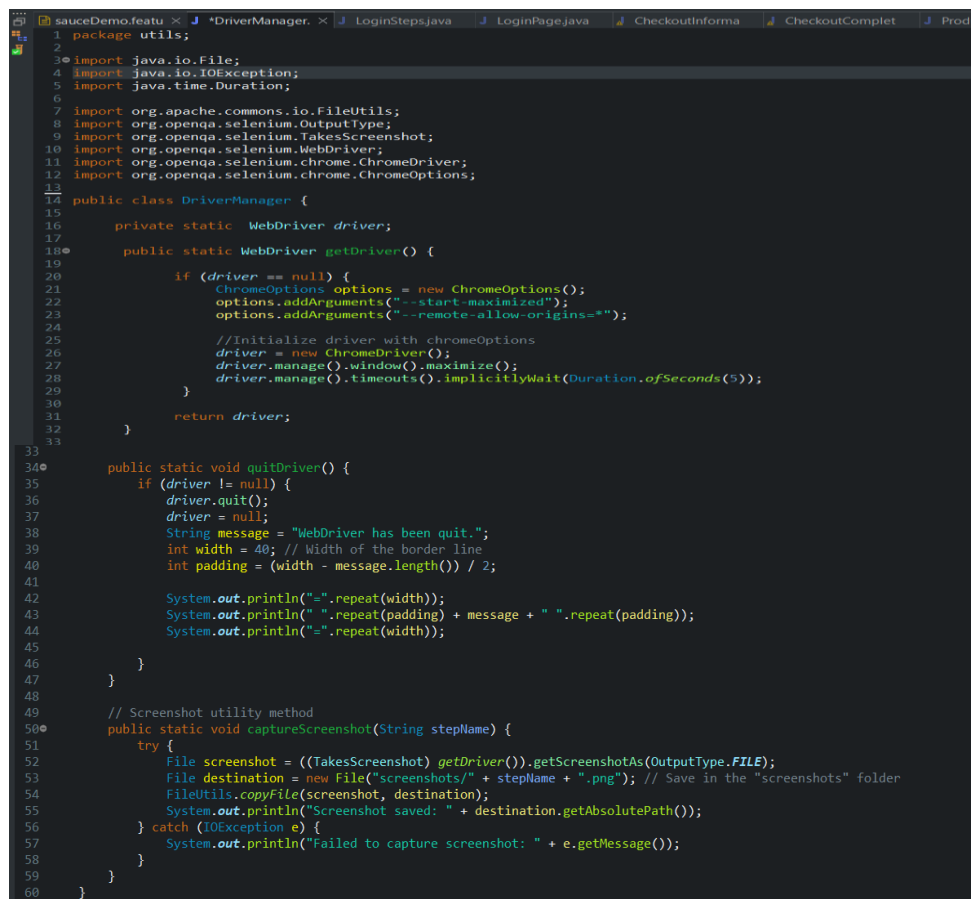
DriverManager

The **DriverManager** is a utility class responsible for creating, configuring, and managing instances of the Selenium **WebDriver** (like **ChromeDriver**).

- **Centralized WebDriver Handling:** All WebDriver initialization logic is in one place, improving maintainability.
- **Reusability:** Avoid duplicating WebDriver setup code in every test file.
- **Flexibility:** Easily switch between different browsers or configurations by modifying the DriverManager class.
- **Singleton Pattern:** Ensures only one WebDriver instance per test, avoiding conflicts in multi-threaded environments.

Common Features of DriverManager:

- Initialize WebDriver.
- Configure browser-specific properties (e.g., paths for ChromeDriver).
- Set browser options (e.g., headless mode, window size).
- Quit WebDriver after test execution.

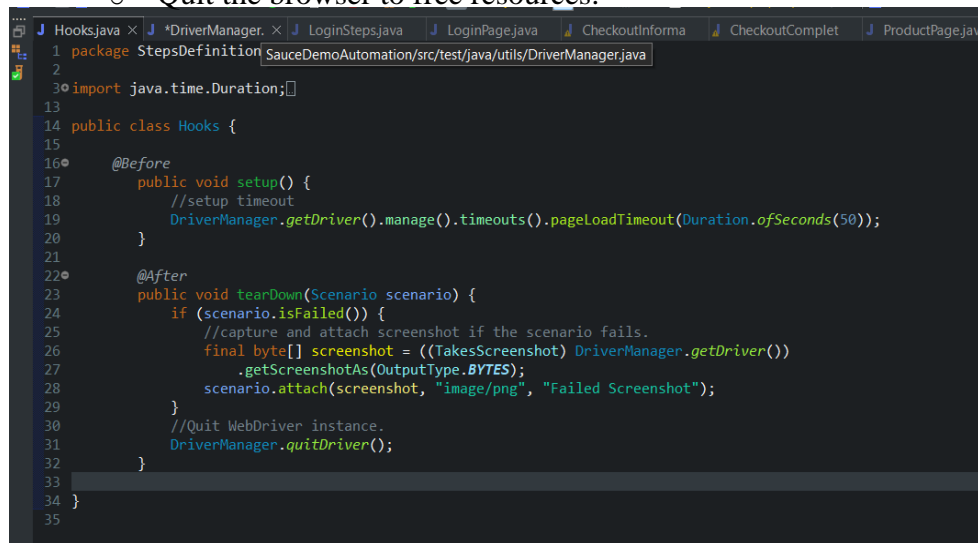


```
1 package utils;
2
3 import java.io.File;
4 import java.io.IOException;
5 import java.time.Duration;
6
7 import org.apache.commons.io.FileUtils;
8 import org.openqa.selenium.OutputType;
9 import org.openqa.selenium.TakesScreenshot;
10 import org.openqa.selenium.WebDriver;
11 import org.openqa.selenium.chrome.ChromeDriver;
12 import org.openqa.selenium.chrome.ChromeOptions;
13
14 public class DriverManager {
15
16     private static WebDriver driver;
17
18     public static WebDriver getDriver() {
19         if (driver == null) {
20             ChromeOptions options = new ChromeOptions();
21             options.addArguments("--start-maximized");
22             options.addArguments("--remote-allow-origins=*");
23             //Initialize driver with chromeOptions
24             driver = new ChromeDriver(options);
25             driver.manage().window().maximize();
26             driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(5));
27         }
28         return driver;
29     }
30
31     public static void quitDriver() {
32         if (driver != null) {
33             driver.quit();
34             driver = null;
35             String message = "WebDriver has been quit.";
36             int width = 40; // Width of the border line
37             int padding = (width - message.length()) / 2;
38
39             System.out.println("=".repeat(width));
40             System.out.println(" ".repeat(padding) + message + " ".repeat(padding));
41             System.out.println("=".repeat(width));
42         }
43     }
44
45     // Screenshot utility method
46     public static void captureScreenshot(String stepName) {
47         try {
48             File screenshot = ((TakesScreenshot) getDriver()).getScreenshotAs(OutputType.FILE);
49             File destination = new File("screenshots/" + stepName + ".png"); // Save in the "screenshots" folder
50             FileUtils.copyFile(screenshot, destination);
51             System.out.println("Screenshot saved: " + destination.getAbsolutePath());
52         } catch (IOException e) {
53             System.out.println("Failed to capture screenshot: " + e.getMessage());
54         }
55     }
56 }
57
58
59
60
```

Hook

Hooks, particularly in **Cucumber**, are special methods annotated with `@Before` and `@After`. They allow you to define **setup** and **cleanup** logic that runs before and after each scenario or test.

- **@Before Hook:**
 - Launch the browser.
 - Set timeouts.
 - Navigate to the starting page of the test.
- **@After Hook:**
 - Take a screenshot if a test fails.
 - Quit the browser to free resources.



```
1 package StepsDefinition;
2
3 import java.time.Duration;
4
5 public class Hooks {
6
7     @Before
8     public void setup() {
9         //setup timeout
10        DriverManager.getDriver().manage().timeouts().pageLoadTimeout(Duration.ofSeconds(50));
11    }
12
13    @After
14    public void tearDown(Scenario scenario) {
15        if (scenario.isFailed()) {
16            //capture and attach screenshot if the scenario fails.
17            final byte[] screenshot = ((TakesScreenshot) DriverManager.getDriver())
18                .getScreenshotAs(OutputType.BYTES);
19            scenario.attach(screenshot, "image/png", "Failed Screenshot");
20        }
21        //Quit WebDriver instance.
22        DriverManager.quitDriver();
23    }
24 }
25
```

5.2 Running Tests in the Local Environment

Prerequisites

- **Java JDK:** Ensure JDK 22 is installed and configured.
- **Maven:** Install Maven (version 3.10.1) and ensure it is added to the system path.
- **ChromeDriver:** Verify that the correct version of ChromeDriver (131.0.6778.86) is installed.
- **Eclipse IDE:** Use Eclipse 2024-06 (4.32.0) to manage and run the project.
- **Dependencies:** Install required dependencies via `pom.xml`.

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>SauceDemoAutomation</groupId>
  <artifactId>SauceDemoAutomation</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
  </properties>

  <dependencies>
    <!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-java -->
    <dependency>
      <groupId>io.cucumber</groupId>
      <artifactId>cucumber-java</artifactId>
      <version>7.3.4</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>4.25.0</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/junit/junit -->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.13.2</version>
      <scope>test</scope>
    </dependency>

    <!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-junit -->
    <dependency>
      <groupId>io.cucumber</groupId>
      <artifactId>cucumber-junit</artifactId>
      <version>7.20.1</version>
      <scope>test</scope>
    </dependency>

    <!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-core -->
    <dependency>
      <groupId>io.cucumber</groupId>
      <artifactId>cucumber-core</artifactId>
      <version>7.20.1</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/io.github.bonigarcia/webdrivermanager -->
    <dependency>
      <groupId>io.github.bonigarcia</groupId>
      <artifactId>webdrivermanager</artifactId>
      <version>5.9.2</version>
    </dependency>
  </dependencies>

```



```

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.7.32</version>
</dependency>

<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.2.6</version>
</dependency>

<dependency>
  <groupId>com.aventstack</groupId>
  <artifactId>extentreports</artifactId>
  <version>5.0.9</version>
</dependency>

  <!-- https://mvnrepository.com/artifact/tech.grasshopper/htmlextentreporter -->
<dependency>
  <groupId>tech.grasshopper</groupId>
  <artifactId>htmlextentreporter</artifactId>
  <version>1.1.0</version>
</dependency>

<dependency>
  <groupId>net.masterthought</groupId>
  <artifactId>cucumber-reporting</artifactId>
  <version>5.7.1</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-devtools-
v130 -->
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-devtools-v130</artifactId>
  <version>4.26.0</version>
</dependency>

</dependencies>
<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.10.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
        <encoding>UTF-8</encoding>
      </configuration>
    </plugin>

    <plugin>
      <groupId>net.masterthought</groupId>
      <artifactId>maven-cucumber-reporting</artifactId>
      <version>5.6.0</version>
      <executions>
        <execution>

```

```

        <id>execution</id>
        <phase>verify</phase>
        <goals>
            <goal>generate</goal>
        </goals>
    </execution>
</executions>
</plugin>
</plugins>
</build>
</project>

```

Step-by-Step Instructions

- Open the project in **Eclipse IDE**.
- Right-click on the `TestRunner.java` class in the **TestRunner** package.
- Select **Run As > JUnit Test**.
- Alternatively, run the tests via Maven using the command:

```
mvn test
```

```

C:\Windows\System32\cmd.exe X + v
Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\asus\eclipse-workspace\SauceDemoAutomation>mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] -----< SauceDemoAutomation:SauceDemoAutomation >-----
[INFO] Building SauceDemoAutomation 0.0.1-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ SauceDemoAutomation ---
[INFO] Copying 0 resource from src\main\resources to target\classes
[INFO]
[INFO] --- compiler:3.10.1:compile (default-compile) @ SauceDemoAutomation ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 6 source files to C:\Users\asus\eclipse-workspace\SauceDemoAutomation\target\classes
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ SauceDemoAutomation ---
[INFO] Copying 1 resource from src\test\resources to target\test-classes
[INFO]
[INFO] --- compiler:3.10.1:testCompile (default-testCompile) @ SauceDemoAutomation ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 9 source files to C:\Users\asus\eclipse-workspace\SauceDemoAutomation\target\test-classes
[INFO]
[INFO] --- surefire:3.2.5:test (default-test) @ SauceDemoAutomation ---
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 6.861 s
[INFO] Finished at: 2024-11-27T07:57:53+05:30
[INFO]
C:\Users\asus\eclipse-workspace\SauceDemoAutomation>

```

- View the execution progress and results in the console output or generated reports.

6. Test Reporting

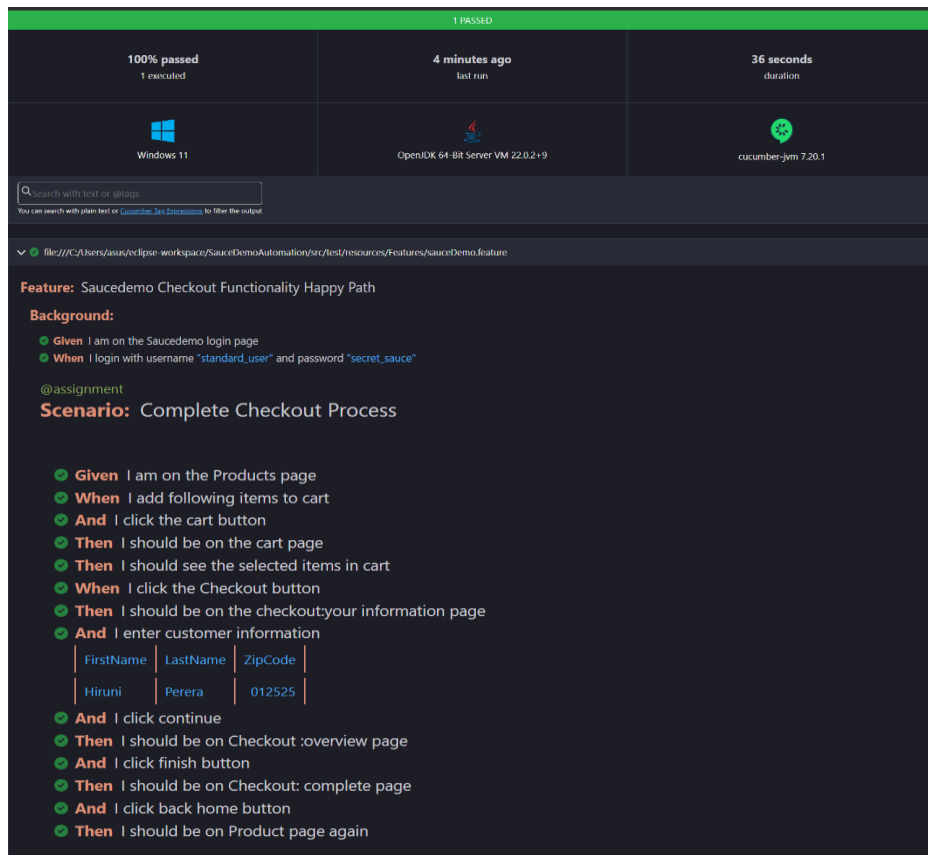
6.1 Reporting Tools or Frameworks Used

The SauceDemo Test Automation project incorporates Cucumber's built-in reporting features along with JUnit reporting to provide comprehensive insights into test execution.

Cucumber Reporting:

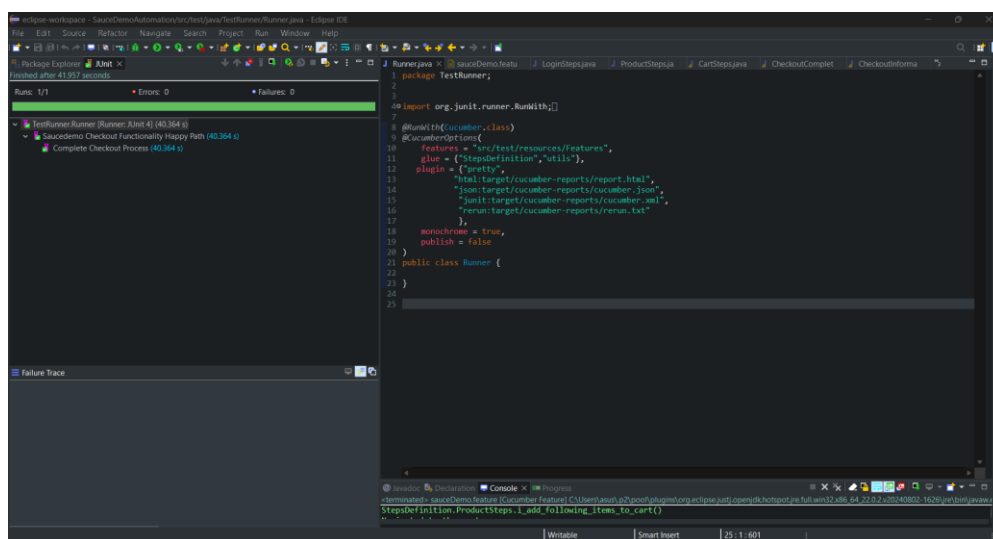
- Generates detailed reports in multiple formats, including HTML, JSON, and XML.

- These reports provide insights into scenario execution, including steps passed, failed, or skipped.



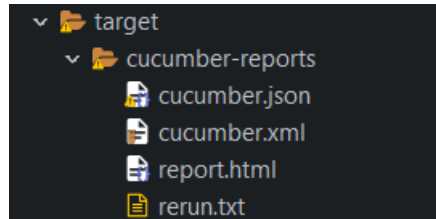
JUnit Reporting:

- Captures test execution results in XML format, suitable for integrating with CI/CD tools.



6.2 Accessing Test Results

- After execution, reports are generated in the target directory of the project.
- Open the **target/cucumber-reports.html** file in a browser to view the detailed execution summary.



7. Conclusion

7.1 Summary of Automation Outcomes

The test automation suite achieves the following outcomes:

- Validates critical functionalities (Login, Product, Cart, Checkout) under **happy path** scenarios.
- Ensures a seamless user experience through efficient and repeatable test scripts.
- Lays the foundation for scaling the test automation framework for future needs.

7.2 Future Enhancements

- Add automation for negative test cases (e.g., invalid login, empty cart).
- Implement cross-browser testing using Selenium Grid.
- Integrate with CI/CD pipelines for continuous testing.

8. Annexures

8.1 Test Scenarios and Gherkin Scripts

Feature	Scenario	Step	Outcome
Saucedemo Checkout Functionality Happy Path	Complete Checkout Process	Given I am on the Saucedemo login page	User is on the login page
Saucedemo Checkout Functionality Happy Path	Complete Checkout Process	When I login with username 'standard_user' and password 'secret_sauce'	Login successful with standard_user
Saucedemo Checkout Functionality Happy Path	Complete Checkout Process	Given I am on the Products page	User navigates to the Products page
Saucedemo Checkout Functionality Happy Path	Complete Checkout Process	When I add following items to cart	Items successfully added to cart
Saucedemo Checkout Functionality Happy Path	Complete Checkout Process	And I click the cart button	Cart button clicked, navigating to the cart page
Saucedemo Checkout Functionality Happy Path	Complete Checkout Process	Then I should be on the cart page	User is on the cart page
Saucedemo Checkout Functionality Happy Path	Complete Checkout Process	Then I should see the selected items in cart	Selected items are displayed in the cart
Saucedemo Checkout Functionality Happy Path	Complete Checkout Process	When I click the Checkout button	Checkout button clicked, navigating to the next page
Saucedemo Checkout Functionality Happy Path	Complete Checkout Process	Then I should be on the checkout: your information page	User lands on the checkout: your information page
Saucedemo Checkout Functionality Happy Path	Complete Checkout Process	And I enter customer information: FirstName: Hiruni, LastName: Perera, ZipCode: 012525	Customer information entered successfully
Saucedemo Checkout Functionality Happy Path	Complete Checkout Process	Then I should be on the product page again	User is on Product Page

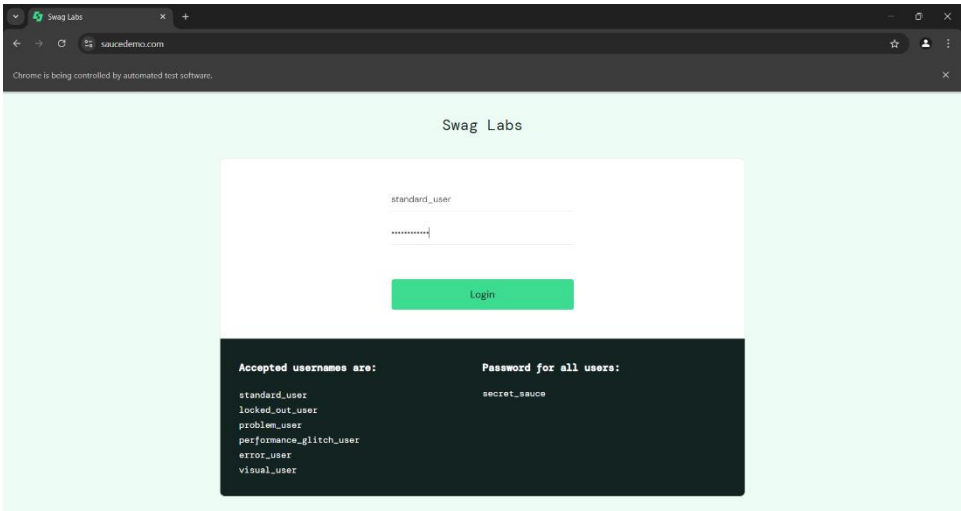
```

1 Feature: Saucedemo Checkout Functionality Happy Path
2
3 Background:
4   Given I am on the Saucedemo login page
5   When I login with username "standard_user" and password "secret_sauce"
6
7 @assignment
8 Scenario: Complete Checkout Process
9   Given I am on the Products page
10  When I add following items to cart
11  And I click the cart button
12  Then I should be on the cart page
13  Then I should see the selected items in cart
14  When I click the Checkout button
15  Then I should be on the checkout: your information page
16  And I enter customer information
17    | FirstName | LastName | ZipCode |
18    | Hiruni   | Perera   | 012525  |
19  And I click continue
20  Then I should be on Checkout :overview page
21  And I click finish button
22  Then I should be on Checkout: complete page
23  And I click back home button
24  Then I should be on Product page again
25
26
27
28
29

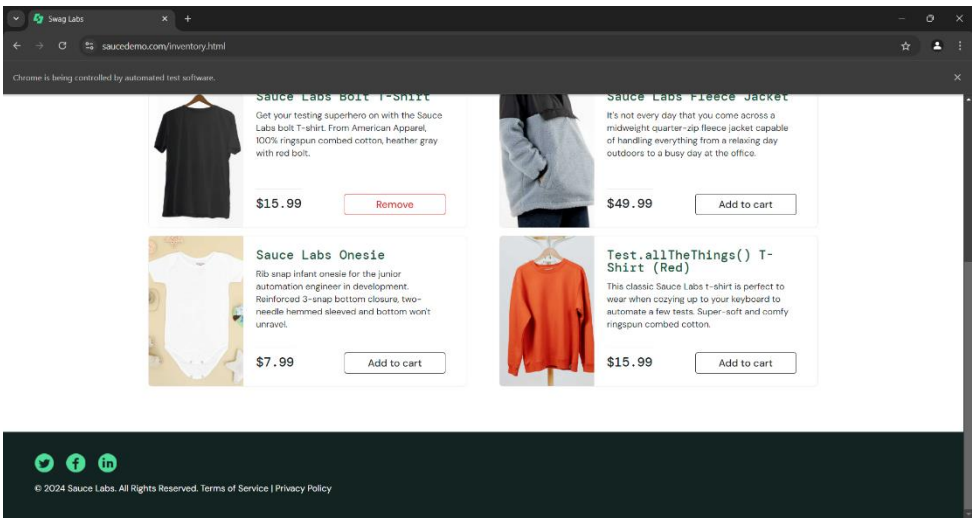
```

- ## 8.2 Sample Code Snippets

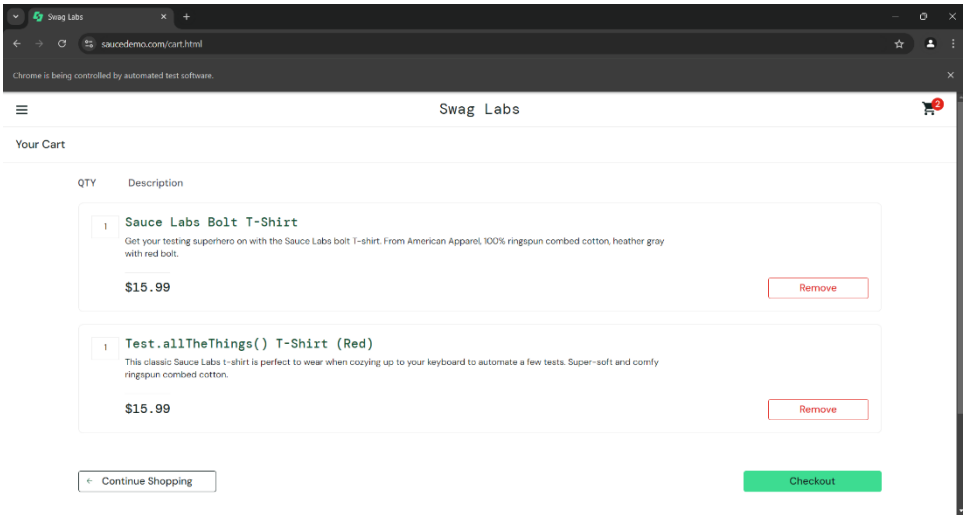
Login page



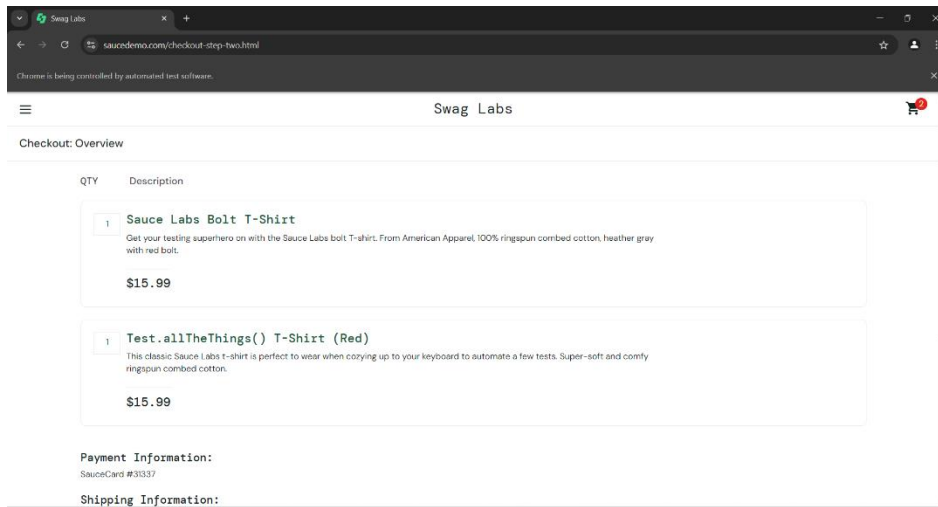
Product Page



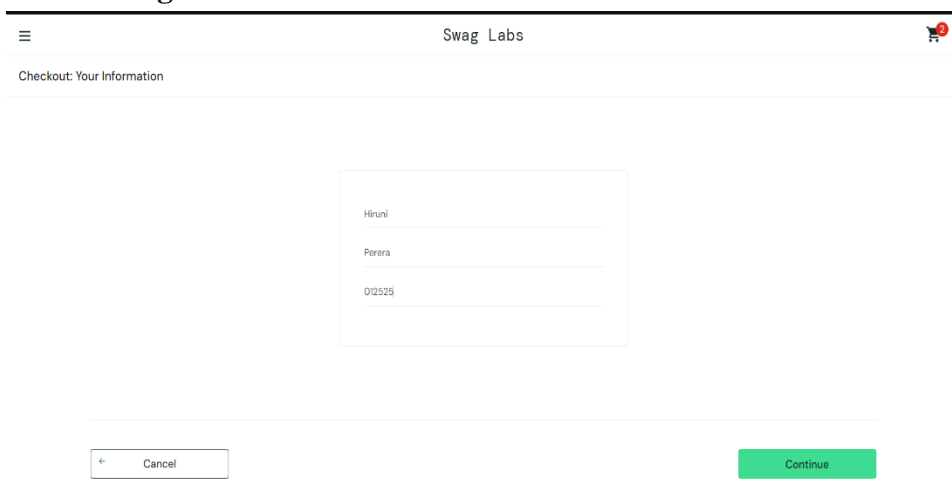
Cart Page



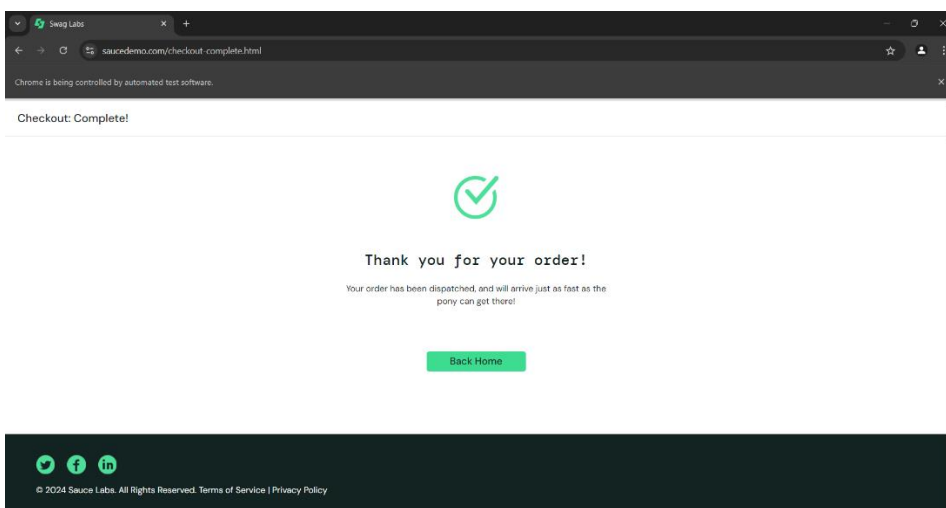
Overview page



Checkout Info Page



Checkout Complete Page



Run As Cucumber Feature

```

Eclipse-workspace - SauceDemoAutomation\src\test\resources\Features\sauceDemo.feature - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Nov 27, 2024 10:24:12 AM cucumber.api.cli.Main run
WARNING: You are using deprecated Main class. Please use io.cucumber.core.cli.Main

@Assignment
Scenario: Complete Checkout Process # src/test/resources/Features/sauceDemo.feature:8
Nov 27, 2024 10:24:18 AM org.openqa.selenium.devtools.CdpVersionFinder.findNearestMatch
WARNING: Unable to find an exact match for CDP version 131, returning the closest version; found: 130; Please update to a Selenium version that supports CDP version 131

=====
Login SauceDemo
=====
Given I am on the SauceDemo login page # StepsDefinition.LoginSteps.i_am_on_the_sauceDemo_login_page()
Username entered: standard_user
Password entered: secret_sauce
Screenshot saved: C:\Users\asus\workspace\SauceDemoAutomation\screenshots\login_page.png
Login button clicked.
Login using : standard_user and password: secret_sauce
When I login with username "standard_user" and password "secret_sauce" # StepsDefinition.LoginSteps.i_login_with_username_and_password(java.lang.String,java.lang.String)
=====
Product Page
=====
Screenshot saved: C:\Users\asus\workspace\SauceDemoAutomation\screenshots\product_page.png
Given I am on the Products page # StepsDefinition.ProductSteps.i_am_on_products_page()
Added item 1: Sauce Labs Bolt T-Shirt
Added item 2: Test.AllTheThings() T-Shirt (Red)
When I add following items to cart # StepsDefinition.ProductSteps.i_add_following_items_to_cart()
Navigated to the cart
And I click the cart button # StepsDefinition.ProductSteps.i_click_the_cart_button()
=====
Your Cart Page
=====
Then I should be on the cart page # StepsDefinition.CartSteps.i_should_be_on_the_cart_page()
Verified that the cart contains 2 item(s).
Screenshot saved: C:\Users\asus\workspace\SauceDemoAutomation\screenshots\cart_page.png
When I should see the selected items in cart # StepsDefinition.CartSteps.i_should_see_the_selected_items_in_cart()
When I click the Checkout button # StepsDefinition.CartSteps.i_click_the_checkout_button()
=====
Checkout : Your Information
=====
Then I should be on the checkout:your information page # StepsDefinition.CheckoutInformationSteps.i_should_be_on_the_checkout_your_information_page()
Customer information entered: Hiruni Perera 012525
Screenshot saved: C:\Users\asus\workspace\SauceDemoAutomation\screenshots\info_page.png
And I enter customer information # StepsDefinition.CheckoutInformationSteps.i_enter_customer_information(io.cucumber.datatable.DataTable)
| FirstName | LastName | ZipCode |
| Hiruni | Perera | 012525 |
And I click continue # StepsDefinition.CheckoutInformationSteps.i_click_continue()
=====
Checkout:Overview Page
=====
Then I should be on Checkout :overview page # StepsDefinition.CheckoutOverviewSteps.i_should_be_on_checkout_overview_page()
And I click finish button # StepsDefinition.CheckoutOverviewSteps.i_click_finish_button()
=====
Checkout:Complete Page
=====
Screenshot saved: C:\Users\asus\workspace\SauceDemoAutomation\screenshots\complete_page.png
Then I should be on Checkout: complete page # StepsDefinition.CheckoutCompleteSteps.i_should_be_on_checkout_complete_page()
And I click back home button # StepsDefinition.CheckoutCompleteSteps.i_click_back_home_button()
Then I should be on Product page again # StepsDefinition.CheckoutCompleteSteps.i_should_be_on_product_page_again()
=====
WebDriver has been quit.
=====

1 Scenarios (1 passed)
16 Steps (16 passed)
0m36.841s

```