Contents

[1. My Test Plan objective: 2](#_Toc198158382)

[**1a. My main focus of Test Plan** 2](#_Toc198158383)

[**1b. Testing Levels** 2](#_Toc198158384)

[**1c. Rationale for Testing Approach** 3](#_Toc198158385)

[2. Functionalities I would prioritize: 4](#_Toc198158386)

[3. Unexpected behaviors I noticed in the sauce application 5](#_Toc198158387)

[4 If you encounter any bugs or unexpected behavior: 6](#_Toc198158388)

[4.1 Document them in detail (steps to reproduce, expected vs. actual results) 6](#_Toc198158389)

[4.2 Explain how you would adjust your testing approach due to these issues 7](#_Toc198158390)

[5. Rationalize for selecting test tools 8](#_Toc198158391)

[6 Tests in automation testing 9](#_Toc198158392)

[Test 1: Login Test 9](#_Toc198158393)

[Test 2: Login, items to shopping cart, payment and complete purchase 12](#_Toc198158394)

[Test 3 : Broken links in the sauce webpage 13](#_Toc198158395)

[Test 4 : Add and remove items from the cart 15](#_Toc198158396)

[Test 5 : **Cart Persistence & Logout Verification** 17](#_Toc198158397)

# My Test Plan objective:

Task for the website: https://www.saucedemo.com/v1/1

As a test engineer firstly, I will thoroughly understand the requirements thoroughly, project objectives and write/contribute to the Test Plan.

To evaluate and validate the core functionality of the Swag Labs application (saucedemo), ensuring a seamless and reliable user experience. The goal is to identify functional issues and establish that the core features work as intended and that users have a positive experience.

## **1a. My main focus of Test Plan**

* **WHAT** needs to be tested in this Saucedemo website – scope, out of scope, levels of testing
* **WHY** it needs to be tested? What the business value am giving to the program
* **WHEN** testing should occur? I will go through the project plan and align test dates to it.
* **WHERE** the testing will be carried out – What type of test environment, data etc

The Test Plan will also contain

* Acceptance criteria (Entry and Exit criteria for each test phase)
* Environments, Test data, Resources and support
* Key risks, dependencies, assumptions, and constraints

## **1b. Testing Levels**

**Static Testing (Shift-Left Approach) –** Early involvement of testers  
Early you detect the bugs, the cheaper they are to fix

The primary goal is to detect defects (accuracy, clarity & consistence) as early as possible in the development lifecycle (before coding begins), reducing the risk of issues emerging in the production environment.

**API testing:** Prior to User Interface development, I will explore on the opportunity to do API testing using rest assured postman do CRUD operations (get, put. Post, protect).

**System Testing** - Validating the functionality of the application in an end-to-end environment. E.g.; Login, Logout, incorrect login, adding and removing from cart. The items alphabetical order adding product to the cart, (name A to Z & Z to A), Price Low to High and High to Low, validation of transaction success message (thank y**ou).**

**Integration Testing**:   
Focuses on verifying the interfaces and data flow between modules (e.g., from the web portal to dashboards), ensuring there are no mismatches or loss of data.

E.g.: Payment process using credit card/debit card, PayPal etc, Checkout, address details capturing.

**Database Testing**:   
Ensures data integrity for key transactions such as payments and order processing (e.g., add to cart, delete, checkout, etc.).

**Non-Functional Testing**

**Performance Testing**: Measures system behavior under varying loads to identify performance bottlenecks. It includes:

* + Response time analysis
  + CPU, memory, disk I/O, and network traffic evaluation
  + Load testing – increasing user load incrementally starting from one user, 5 users, 10 users, 25 users concurrently
  + Volume testing – handling large data sets or transactions

**Tools I would recommend:** Load Runner, JMeter

**E.g.:** In saucedemo, calculating the response time with 1 user(monitor cpu, disk, I/o, network), calculating the response time with 5 users, 10 users, 50 users concurrently & monitor the system utilization

Give the customer the analysis of the load on the server when multiple transactions mix is happening with multiple users. Doing orders and payments by multiple users at the same point of time

**Operational Acceptance Testing (OAT) – This helps to avoid defects in production**

I will plan these tests in preprod which are mimic to real productions. I will write OAC (operation acceptance criteria) and coordinate with network, storage, dba and compute resources to execute the below tests. Assesses the system’s operational readiness, including:

* + - Failover mechanisms
    - Jumbo frame testing (packet size testing)
    - Backup and recovery procedures
    - Alerting and monitoring systems
    - Job scheduling and automation

**Security Testing**: Ensures the application is protected from threats and vulnerabilities.

* + - Authentication and authorization checks (e.g., for the Sauce Labs demo site)
    - Penetration testing to identify and address vulnerabilities
    - Monitoring using tools like Splunk.

1. **Cross browser testing**: – Testing the application across multiple browsers (chrome, IE, safari, Opera, Firefox) and on various Os windows and Linux/Unix
2. **Business process testing**: From the user perspective business end to end journeys. Focus on UI and UX, Look and feel

## **1c. Rationale for Testing Approach**

* **Shift-Left Approach:** Combine exploratory testing with early automation to detect issues sooner in the pipeline.**User-Centric Testing:** Simulate realistic end-user behavior to catch usability and logic errors early.

# Functionalities I would prioritize:

**Before I prioritize, I will question and brainstorm on the below:**

* Which aspect of the Souse web application is used by business most e.g.; customer portal
* Which aspect of the element has highest financial impact e.g.: Items, purchasing and returning
* Which aspect of the has metrics used by customers for their day to day business eg:dashboards
* Which aspect of the business application has integration issues
* Which aspect of the application may cause maintainability issues in future
* Which aspect of the application may cause production issues
* Which aspect of the application may cause most customer complaints

1. **User Authentication**
   * Valid attempts with all user ids and passwords provided (e.g.: standard\_user)
   * and invalid login attempts
   * Error handling for missing credentials
   * Locked-out user validation
   * Session expiry ( if you leave the system idle for 15 minutes or more)
2. **Product Catalog**
   * Display and sorting of product listings from the list box on top right hand corner
   * Item details accuracy without any spelling mistakes
   * Add/remove items to/from cart
3. **Shopping Cart Functionality**
   * Cart updates correctly when items are added or removed
   * Product quantity and pricing validation
4. **Checkout Flow**
   * Input validation on user information (first name, last name, postal code)
   * Order summary accuracy
   * Successful and failed order placements
5. **Navigation and Links**
   * Correct routing between pages (e.g., inventory, cart, checkout)
   * Logo/home links, menu, and logout functionalities
6. **Input Validation and Boundary Testing**
   * Text field limits
   * Special characters handling
   * Required field validation
7. **Disability disrimenent act testing** : use tools like narrator to test for certain vulnerable people e.g.: disability, vulnerability, blind, deaf etc
8. **UI/Ux validation**
9. **Back button functionality**

# Unexpected behaviors I noticed in the sauce application

System and integrity bugs:

* Usernames and passwords are available in the home page which is security violation
* The error message formatting is inconsistent when incorrect credentials are entered
* The cart icon does not always update correctly when items are added or removed
* Session functionality not working, leave the system idle for 15 minutes, it should be locked out.
* Payment process is not built
* The images displayed for products do not always match the actual items
* Contact us, Help page in the footer not displayed, as per W3C standard these should be there.
* Clicking "Add to cart" multiple times can cause duplicate items or improper cart count
* Forgot password is not included
* About us is not working
* Add items, logout, then login again – the cart reset unexpectedly.
* Reset App is not working
* In the checkout page, your information, it doesn’t state delivery address
* Single letters are accepted in the information section of customer, no errors thrown
* Numbers are accepted in first name and last name
* The menu icon (☰) overlaps text or doesn’t open on smaller screens Special characters are allowed in post code e.g.: &(#(#) – no error through
* Zoom button is not working on the image
* Description of the product is not correct

**. UI Bugs:**

* The menu icon overlaps text or doesn’t open on smaller screens.
* Some elements overlap or are misaligned, especially on different screen sizes
* Clicking on "About" opens a non-existent or external URL unexpectedly.
* Certain buttons do not appear correctly or are hidden when resizing the window
* UI Zoom button is not working on image
* Low contrast in button text and background.
* Social media links in the footer may lead to dummy or 404 pages.
* Missing unnecessary large images or JS images are loaded upfront

**Performance:**

* Application is not security built
* Observed lagging in the pages sometimes
* Unnecessary large images or JS libraries are loaded upfront.

# 4 If you encounter any bugs or unexpected behavior:

4.1 Document them in detail (steps to reproduce, expected vs. actual results)

**Defect ID: usually auto generated by the test tool, if manually we need to give one number as agree with test manager/lead**

**Status: New/Open/fixed/Rested/closed/Deferred**

**Defect description:**

**Criticality:** How impact to the business if it occurs in production. Select from the below as per the nature of ht defect

**CRITICAL, MAJOR, MINOR, NO IMPACT**

**PRIORTY**: How important to fix the bug? Select from the below as per the nature of the defect wile reporting

**Showstopper, HIGH. MEDIUM and LOW**

**Expected:**

**Actual result happened:**

**Module/functionality Name:**

**Test case id:**

**Screensho**t

**Attachments if any:**

**Time stamp - this is very crucial to check the logs by development team**

**Test phase detected:**

**Date raised:**

**Steps to reproduce:** Provide a clear sequence of actions leading to the issue.

**Test environment** – ST, SIT, UAT or performance etc - Include browser version, OS, Test data used

**Build TAR version deployed**

**Defect assigned to**

**Defect raised by**

**Root cause** : <after fixing the defect>

## 4.2 Explain how you would adjust your testing approach due to these issues

Test issues could impact test coverage, timelines, and focus. The below key steps I would do:

* Communication: I would ensure ongoing discussions with scrum team and communicate on daily basis to the progress of the defect to the test team, key stakeholders
* Understanding Root Cause Analysis –Will try to investigate deeper to understand why the issue occurred, asking dev teams what steps they are taking to prevent similar defects in future releases.
* Risk Assessment with team– Brainstorm and try to understand the bug’s impact on other functionalities to determine if broader testing adjustments are needed.
* Stop furthers tests in the affected area: I would stop deeper testing in that affected area until the bug is properly addressed.
* Focus on critical Test Scenarios: I would see test edge cases to uncover related defects.
* Prioritize High-Impact Bugs: I would focus on issues affecting core functionality first.
* Re-test After Fixes & regression testing : Try to validate that resolved issues do not introduce new
* Update Test Cases: If the bug leads to a spec change or identifies missing validation, I would update or add new test cases.
* Adjust test timelines: if critical bug delays in getting fix, I would re align testing plans and communicate delays to the project team.
* Regression Testing Strategy – Identify areas affected by changes and conduct thorough regression testing to maintain system stability.

# 5. Rationalize for selecting test tools

These tools collectively ensure a structured, scalable, and efficient test automation framework

* **Java (8+) –** 
  + - A powerful object-oriented language with strong community support for test automation. Compatible with modern CI/CD pipelines and cloud-based testing tools (e.g., Browser Stack, Selenium Grid).
    - Offers functional programming features (from Java 8+) like lambdas, which can simplify code.
* **Selenium WebDriver** 
  + - Most widely used open source, we can easily automate web based applications.
    - Allows for advanced user interactions like mouse hover, drag-and-drop, JavaScript execution
    - No need to spend licensing costs.
    - Easily integrates with Java, TestNG, Cucumber, and CI tools like Jenkins.
* Cucumber (BDD) :
  + - Facilitates behavior-driven development (BDD) with readable, user-friendly test scripts.
    - Non technical resources can easily understand the BDD cucumber way of style using Jerkin.
* JUnit (or TestNG) –
  + - Robust frameworks for writing and managing automated test cases efficiently.
* Maven –
  + - A powerful build automation tool that manages dependencies and simplifies project configuring.

# 6 Tests in automation testing

## Test 1: Login Test

Rational: The rationale for selecting **Gherkin language** for test automation is its **clarity, simplicity, and business-readable format**, which bridges the gap between technical teams and non-technical stakeholders.

 Behavior**-Driven Development (BDD)** – Encourages collaboration by defining tests in terms of user behavior rather than technical details.

**Reusability & Maintainability** – Common steps like Given user is on the login page can be reused across multiple scenarios, improving efficiency.

**Automation-Ready** – Works seamlessly with frameworks like Cucumber, enabling automated execution of test cases.

**Clear Test Coverage** – Your scenarios explicitly define both successful and failed login attempts, ensuring core functionality is validated.

**Folder structure is below**

src

├── test

│ ├── java

│ │ ├── stepDefinitions

│ │ ├── runners

│ │ └── pages

│ ├── resources

│ │ └── features

Pom.xml Dependencies

* Selenium-java – 4.12.1
* Cucumber-java – 7.14.0
* Cucmber-junit – 7.14.0
* Junit – 4.13.2
* Java 21 version

1. Login.feature (in resources/features)

|  |
| --- |
| Gherkin language  Feature: Login functionality of Swag Labs  Scenario: Login with valid credentials  Given user is on the login page ‘https://www.saucedemo.com/v1/index.html’  When user logs in with username "standard\_user" and password "secret\_sauce"  Then user should be redirected to the products page  Scenario: Login with invalid credentials  Given user is on the login page  When user logs in with username "invalid\_user" and password "wrong\_pass"  Then an error message should be displayed |

2. LoginSteps.java (in stepDefinitions)

|  |
| --- |
| packagestepDefinitions;  importorg.openqa.selenium.WebDriver;  importorg.openqa.selenium.chrome.ChromeDriver;  importpages.LoginPage;  importio.cucumber.java.en.\*;  importstaticorg.junit.Assert.\*;  publicclassLoginSteps {  WebDriver driver;  LoginPage login;  @Given("user is on the login page")  publicvoidopenLoginPage() {  driver = newChromeDriver();  driver.get("https://www.saucedemo.com/v1/");  login = newLoginPage(driver);  }  @When("user logs in with username {string} and password {string}")  publicvoidloginWithCredentials(String username, String password) {  login.login(username, password);  }  @Then("user should be redirected to the products page")  publicvoidverifyProductsPage() {  assertTrue(driver.getCurrentUrl().contains("inventory.html"));  driver.quit();  }  @Then("an error message should be displayed")  publicvoidverifyErrorMessage() {  assertTrue(login.getErrorMessage().contains("Username and password do not match"));  driver.quit();  }  } |

3. LoginPage.java (in pages)

|  |
| --- |
| package pages;  importorg.openqa.selenium.By;  importorg.openqa.selenium.WebDriver;  publicclassLoginPage {  WebDriver driver;  privateByusernameField= By.id("user-name");  privateBypasswordField= By.id("password");  privateByloginButton= By.id("login-button");  privateByerrorMessage=By.cssSelector("[data-test='error']");  publicLoginPage(WebDriver driver) {  this.driver = driver;  }  publicvoidlogin(String username, String password) {  driver.findElement(usernameField).sendKeys(username);  driver.findElement(passwordField).sendKeys(password);  driver.findElement(loginButton).click();  }  public String getErrorMessage() {  returndriver.findElement(errorMessage).getText();  }  } |

TEST RUNNER

|  |
| --- |
| 4. TestRunner.java (in runners)  package runners;  importio.cucumber.junit.Cucumber;  importio.cucumber.junit.CucumberOptions;  importorg.junit.runner.RunWith;  @RunWith(Cucumber.class)  @CucumberOptions(  features = "src/test/resources/features",  glue = "stepDefinitions",  plugin = {"pretty", "html:target/cucumber-reports.html"},  monochrome = true  )  publicclassTestRunner {  } |

## Test 2: Login, items to shopping cart, payment and complete purchase

The **checkout and payment test** is **crucial** because it validates the **core transaction flow**, ensuring users can **successfully purchase items** in Swag Labs. Here’s why it was selected:

* **Business-Critical Functionality – If checkout fails, users cannot complete purchases, directly impacting revenue and user satisfaction.**
* **Data Integrity Checks – Ensures that the correct product, price, and user details persist throughout the transaction process.**
* **Payment Processing Validation – Confirms input validation, payment details handling, and potential security risks.**

The reason to take this test it’s crucial

|  |
| --- |
| Feature: Purchase Items and Checkout in Swag Labs  Scenario: Select an item and complete checkout  Given user is logged in to Swag Labs  When user adds "Sauce Labs Backpack" to the cart  And user proceeds to checkout  And user enters payment details and confirms order  Then user should see the order confirmation message |

|  |
| --- |
| package stepDefinitions;  import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.chrome.ChromeDriver;  import io.cucumber.java.en.\*;  public class Checkout Steps {  WebDriver driver;  @Given("user is logged in to Swag Labs")  public void user\_is\_logged\_in() {  driver = new ChromeDriver();  driver.manage().window().maximize();  driver.get("https://www.saucedemo.com/v1/index.html");    // Login Process  driver.findElement(By.id("user-name")).sendKeys("standard\_user");  driver.findElement(By.id("password")).sendKeys("secret\_sauce");  driver.findElement(By.id("login-button")).click();  }  @When("user adds {string} to the cart")  public void user\_adds\_item\_to\_cart(String itemName) {  driver.findElement(By.xpath("//div[text()='" + itemName + "']/../../following-sibling::div/button")).click();  driver.findElement(By.className("shopping\_cart\_link")).click();  }  @And("user proceeds to checkout")  public void user\_proceeds\_to\_checkout() {  driver.findElement(By.id("checkout")).click();  }  @And("user enters payment details and confirms order")  public void user\_enters\_payment\_details() {  driver.findElement(By.id("first-name")).sendKeys("John");  driver.findElement(By.id("last-name")).sendKeys("Doe");  driver.findElement(By.id("postal-code")).sendKeys("12345");  driver.findElement(By.id("continue")).click();  driver.findElement(By.id("finish")).click();  }  @Then("user should see the order confirmation message")  public void user\_should\_see\_confirmation\_message() {  boolean isDisplayed = driver.findElement(By.className("complete-header")).isDisplayed();  assert isDisplayed;  driver.quit();  }  } |

|  |
| --- |
| package runner;  import org.junit.runner.RunWith;  import io.cucumber.junit.Cucumber;  import io.cucumber.junit.CucumberOptions;  @RunWith(Cucumber.class)  @CucumberOptions(features="src/test/resources/features", glue={"stepDefinitions"})  public class TestRunner {  } |

## Test 3 : Broken links in the sauce webpage

Rational I selected reason below :

Broken link testing is crucial because it directly impacts user experience and website credibility. Here’s why I chose this test:

Enhances User Experience – Nothing frustrates users more than clicking on a link that leads to a 404 error. It disrupts navigation and lowers engagement.

Improves SEO Rankings – Search engines like Google penalize sites with broken links, reducing visibility in search results.

Maintains Site Reliability – If important pages (like product details, payment pages, or help centers) contain broken links, users lose trust and may abandon the site.

Prevents Business Losses – For e-commerce platforms like the one you're testing, broken links on checkout/payment pages can lead to lost sales and missed opportunities.

|  |
| --- |
| Feature: Broken Links Verification  Scenario: Verify that all links on the page are working  Given I navigate to "https://www.saucedemo.com/v1/index.html"  When I extract all links from the page  Then all links should return a valid status code |

|  |
| --- |
| package stepDefinitions;  Import io.cucumber.java.en.\*;  import org.junit.Assert;  import org.openqa.selenium.\*;  import org.openqa.selenium.chrome.ChromeDriver;  import java.io.IOException;  import java.net.HttpURLConnection;  import java.net.URL;  import java.util.List;  public class BrokenLinksSteps {  WebDriver driver;  @Given("I navigate to {string}")  public void iNavigateTo(String url) {  driver = new ChromeDriver();  driver.get(url);  }  @When("I extract all links from the page")  public void iExtractAllLinks() {  List<WebElement> links = driver.findElements(By.tagName("a"));    for (WebElement link : links) {  String url = link.getAttribute("href");  if (url != null && !url.isEmpty()) {  verifyLink(url);  }  }  }  @Then("all links should return a valid status code")  public void allLinksShouldReturnAValidStatusCode() {  driver.quit();  }  public void verifyLink(String url) {  try {  HttpURLConnection connection = (HttpURLConnection) new URL(url).openConnection();  connection.setRequestMethod("HEAD");  connection.connect();  int responseCode = connection.getResponseCode();  System.out.println(url + " -> " + responseCode);  Assert.assertTrue("Broken link found: " + url, responseCode < 400);  } catch (IOException e) {  System.out.println("Error checking link: " + url);  }  }  } |

|  |
| --- |
| package runners;  import org.junit.runner.RunWith;  import io.cucumber.junit.Cucumber;  import io.cucumber.junit.CucumberOptions;  @RunWith(Cucumber.class)  @CucumberOptions(  features = "src/test/resources/features",  glue = "stepDefinitions",  plugin = {"pretty", "html:target/cucumber-reports"}  )  public class TestRunner {  } |

## Test 4 : Add and remove items from the cart

Why I chose this test, the reason below

Testing the add & remove items functionality is critical because it ensures a seamless shopping experience. Here’s why this test is important:

**Data Accuracy & Price Calculation** – The test ensures correct item counts and that the total price updates properly when products are added or removed.

**Session Handling & UI Stability** – Verifying that the cart updates correctly helps detect UI bugs, caching issues, or session-related problems.

**User Convenience** – Shoppers expect a smooth cart interaction. If items don’t add/remove properly, they may abandon their purchase.

**Prevents Revenue Loss** – If a product gets stuck in the cart or removal fails, users may hesitate to complete checkout, leading to lost sales.

|  |
| --- |
| import org.junit.\*;  import org.openqa.selenium.\*;  import org.openqa.selenium.chrome.ChromeDriver;  import java.util.List;  public class CartTest {  WebDriver driver;  @Before  public void setup() {  driver = new ChromeDriver();  driver.get("https://www.saucedemo.com/v1/index.html");    // Login  driver.findElement(By.id("user-name")).sendKeys("standard\_user");  driver.findElement(By.id("password")).sendKeys("secret\_sauce");  driver.findElement(By.id("login-button")).click();  }  @Test  public void addAndRemoveItemsFromCart() {  // Add multiple items to cart  List<WebElement> addButtons = driver.findElements(By.className("btn\_inventory"));  addButtons.get(0).click();  addButtons.get(1).click();  addButtons.get(2).click();  // Navigate to cart  driver.findElement(By.className("shopping\_cart\_link")).click();  // Verify items are added  List<WebElement> cartItems = driver.findElements(By.className("cart\_item"));  Assert.assertEquals("Item count mismatch!", 3, cartItems.size());  // Remove one item  List<WebElement> removeButtons = driver.findElements(By.className("cart\_button"));  removeButtons.get(1).click();  // Verify item is removed  cartItems = driver.findElements(By.className("cart\_item"));  Assert.assertEquals("Item removal failed!", 2, cartItems.size());  // Verify total price updates (sample verification)  WebElement totalPrice = driver.findElement(By.className("summary\_total\_label"));  Assert.assertNotNull("Total price missing!", totalPrice);  }  @After  public void teardown() {  driver.quit();  }  } |
|  |

## Test 5 : **Cart Persistence & Logout Verification**

* Cart Persistence – Ensuring items remain in the cart after a page refresh.
* Logout Functionality – Verifying that logging out redirects the user to the login page.

This test is critical because it ensures two essential aspects of user experience and security:

**User Experience & reliable**: If cart items disappear after a page refresh, users may abandon their purchase, leading to lost sales.

**Session Handling & session cleanup:** Validates whether the cart data is stored correctly in the session or database. Ensures user sessions and cookies are cleared, avoiding potential security risks.

Combining cart persistence and logout verification ensures that shopping experiences remain smooth and secure, reinforcing trust in the platform

|  |
| --- |
| import org.junit.\*;  import org.openqa.selenium.\*;  import org.openqa.selenium.chrome.ChromeDriver;  public class CartPersistenceLogoutTest {  WebDriver driver;  @Before  public void setup()  {  driver = new ChromeDriver();  driver.get("https://www.saucedemo.com/v1/index.html");  **// Login**  driver.findElement(By.id("user-name")).sendKeys("standard\_user");  driver.findElement(By.id("password")).sendKeys("secret\_sauce");  driver.findElement(By.id("login-button")).click();  }  **@Test**  public void verifyCartPersistenceAndLogout() {  // Add items to cart  driver.findElement(By.id("add-to-cart-sauce-labs-backpack")).click();  driver.findElement(By.id("add-to-cart-sauce-labs-bolt-t-shirt")).click();  // Refresh page  driver.navigate().refresh();  // Verify items still in cart  driver.findElement(By.className("shopping\_cart\_link")).click();  Assert.assertTrue("Backpack missing after refresh!", driver.getPageSource().contains("Sauce Labs Backpack"));  Assert.assertTrue("T-Shirt missing after refresh!", driver.getPageSource().contains("Sauce Labs Bolt T-Shirt"));  **// Logout**  driver.findElement(By.id("react-burger-menu-btn")).click();  driver.findElement(By.id("logout\_sidebar\_link")).click();  **// Verify redirection to login page**  Assert.assertTrue("Logout failed!", driver.getCurrentUrl().contains("login"));  }  @After  public void teardown() {  driver.quit();  }  } |