**Assignment-2**

**Step 1: Project Setup**

1. **Create a Maven project**:
   * Use Maven to manage dependencies and build the project.
   * Add required dependencies to the pom.xml:
     + **Selenium WebDriver** for browser automation.
     + **TestNG** for test management and assertions.
     + **WebDriverManager** to manage browser drivers automatically.
     + Optionally, **ExtentReports** for reporting.

Example of the pom.xml dependencies:

<dependencies>

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>4.x.x</version>

</dependency>

<dependency>

<groupId>io.github.bonigarcia</groupId>

<artifactId>webdrivermanager</artifactId>

<version>5.x.x</version>

</dependency>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>7.x.x</version>

<scope>test</scope>

</dependency>

</dependencies>

**Step 2: Test Scenarios Implementation**

1. **Verify the Sorting Order (Z-A) on the “All Items” Page**:
   * Navigate to the **Products** page after login.
   * Select the sorting option “Name (Z to A)” from the dropdown.
   * Fetch the product names and verify if they are sorted in descending alphabetical order.

@Test

public void verifySortingZA() {

WebDriver driver = new ChromeDriver();

driver.get("https://www.saucedemo.com/");

// Login

login("standard\_user", "secret\_sauce");

// Select Z-A Sorting

WebElement sortDropdown = driver.findElement(By.className("product\_sort\_container"));

Select select = new Select(sortDropdown);

select.selectByVisibleText("Name (Z to A)");

// Verify sorting order

List<WebElement> products = driver.findElements(By.className("inventory\_item\_name"));

List<String> productNames = products.stream().map(WebElement::getText).collect(Collectors.toList());

List<String> sortedNames = new ArrayList<>(productNames);

Collections.sort(sortedNames, Collections.reverseOrder());

Assert.assertEquals(productNames, sortedNames, "Products are not sorted Z-A.");

driver.quit();

}

1. **Verify the Price Order (High to Low)**:
   * Select the sorting option “Price (high to low)” from the dropdown.
   * Fetch the product prices and ensure they are sorted in descending order.

@Test

public void verifyPriceHighToLow() {

WebDriver driver = new ChromeDriver();

driver.get("https://www.saucedemo.com/");

// Login and navigate to products

login("standard\_user", "secret\_sauce");

// Select High to Low price sorting

WebElement sortDropdown = driver.findElement(By.className("product\_sort\_container"));

Select select = new Select(sortDropdown);

select.selectByVisibleText("Price (high to low)");

// Verify price sorting

List<WebElement> productPrices = driver.findElements(By.className("inventory\_item\_price"));

List<Double> prices = productPrices.stream().map(e -> Double.valueOf(e.getText().replace("$", "")))

.collect(Collectors.toList());

List<Double> sortedPrices = new ArrayList<>(prices);

Collections.sort(sortedPrices, Collections.reverseOrder());

Assert.assertEquals(prices, sortedPrices, "Prices are not sorted high to low.");

driver.quit();

}

1. **Add Multiple Items to Cart and Validate the Checkout Journey**:
   * Add multiple items to the cart, proceed to checkout, and validate the total amount.
   * Perform validation of the checkout flow, including user information and order completion.

@Test

public void verifyCheckoutJourney() {

WebDriver driver = new ChromeDriver();

driver.get("https://www.saucedemo.com/");

// Login and add items to the cart

login("standard\_user", "secret\_sauce");

driver.findElement(By.id("add-to-cart-sauce-labs-backpack")).click();

driver.findElement(By.id("add-to-cart-sauce-labs-bike-light")).click();

// Proceed to checkout

driver.findElement(By.className("shopping\_cart\_link")).click();

driver.findElement(By.id("checkout")).click();

// Enter user information

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();

// Validate total price

String totalPrice = driver.findElement(By.className("summary\_total\_label")).getText();

Assert.assertTrue(totalPrice.contains("Total: $"), "Total price is not displayed correctly.");

// Finish the checkout process

driver.findElement(By.id("finish")).click();

String completionMessage = driver.findElement(By.className("complete-header")).getText();

Assert.assertEquals(completionMessage, "THANK YOU FOR YOUR ORDER", "Checkout not completed successfully.");

driver.quit();

}

**Step 3: Bonus Points - Visual Testing and Accessibility Testing**

1. **Automated Visual Testing**:
   * Capture screenshots before and after specific actions (e.g., page load, sort action).
   * Use tools like **Applitools** or **Selenium’s TakeScreenshot** to compare the current UI with a baseline image.

java

File screenshot = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(screenshot, new File("screenshots/products\_page.png"));

1. **Automated Accessibility Testing**:
   * Use **axe-core** to test for accessibility issues on the page.
   * Include verification for alt text, ARIA labels, and other key accessibility attributes.

java

// Using Axe-core for accessibility testing

AxeBuilder axe = new AxeBuilder();

Results results = axe.analyze(driver);

Assert.assertEquals(results.getViolations().size(), 0, "Accessibility issues found: " + results.getViolations());

**Step 4: README.md**

Provide the following details in the README file:

1. **Project Overview**:
   * Briefly describe the scenarios tested.
2. **Prerequisites**:
   * Java 8+
   * Maven
   * Chrome/Firefox
3. **Setup and Execution**:
   * Instructions on how to clone the repository and run tests locally.

bash

mvn test

1. **Report Generation**:
   * Mention the location of the reports/logs after test execution (e.g., target/surefire-reports).

**Step 5: Test Execution Logs/Reports and Recording**

1. **TestNG Reports**:
   * Ensure TestNG generates default HTML reports. You can also configure **ExtentReports** for better reporting.
2. **Execution Logs**:
   * Use Log4j or another logging framework to capture execution details.
3. **Recording**:
   * Record test executions for both headed and headless modes, saving the videos in the videos/ directory.

**Final Checklist:**

* Tests for sorting Z-A and price high-low.
* Cart and checkout flow validation.
* Bonus tests: visual and accessibility.
* README.md with setup instructions.
* Execution logs and TestNG reports.
* Headed and headless execution recordings.