LATHA MATHAVAN ENGINEERING COLLEGE

KIDARIPATTI, ALAGARKOIL, MADURAI-625301

CCS366-SOFTWARE TESTING AND AUTOMATION LABORATORY

Regulation: 2021

Branch : B.E.CSE

Year : IIIrd Year

Semester : VI

LATHA MATHAVAN ENGINEERING COLLEGE

KIDARIPATTI, ALAGARKOIL, MADURAI-625301

Department of _____Engineering Laboratory Record

Name: _____ CLASS _____

REGISTER NO: _____

Certified that this is bona fide record of work done by the above student of the

CCS366-SOFTWARE TESTING AND AUTOMATION LABORATORY
during the **year 2023-24 (Even semester)**

Signature of Lab-in-Charge Signature of Head of the Department

Submitted for the Practical Examination Held on_____

INTERNAL EXAMINATION EXTERNAL EXAMINATION

TABLE OF CONTENTS

S.NO.	DATE	EXPERIMENT TITLE	PAGE NO.	MARKS	SIGN.
1.		DEVELOP THE TEST PLAN FOR TESTING AN ECOMMERCE WEB/MOBILE APPLICATION (www.amazon.com)			
2.		DESIGN THE TEST CASE FOR TESTING THE E-COMMERCE APPLICATION			
3.		TEST THE E-COMMERCE APPLICATION AND REPORT THE DEFECTS IN IT			
4.		DEVELOP THE TEST PLAN AND DESIGN THE TEST CASES FOR AN INVENTORY CONTROL SYSTEM			
5.		EXECUTE THE TEST CASES AGAINST A CLIENT SERVER OR DESKTOP APPLICATION AND IDENTIFY THE DEFECTS			
6.		TEST THE PERFORMANCE OF THE ECOMMERCE APPLICATION			
7.		AUTOMATE THE TESTING OF E-COMMERCE APPLICATION USING SELENIUM			
8.		INTEGRATE TESTNG WITH ABOVE TEST AUTOMATION			
9.a		BUILD A DATA-DRIVEN FRAMEWORK USING SELENIUM AND TESTNG			
9.b		BUILD PAGE OBJECT MODEL USING SELENIUM AND TESTNG			
9.c		BUILD A BDD FRAMEWORK USING SELENIUM, TESTNG AND CUCUMBER			

EX. NO: 1 **DEVELOP THE TEST PLAN FOR TESTING AN E-COMMERCE**

DATE: **WEB/MOBILE APPLICATION (www.amazon.com)**

AIM

To Develop the test plan for testing an e-commerce web/mobile application (www.amazon.com)

PROCEDURE

1. Test Objectives:

- Verify that users can browse and search for products.
- Confirm that products are displayed accurately with correct details.
- Test the user registration and login process.
- Validate the shopping cart and checkout process.
- Test payment processing and order placement.
- Check for responsiveness and usability on mobile devices.
- Verify security aspects such as data encryption and session management.

2. Test Environment:

- Web Browsers: Chrome, Firefox, Safari, Edge.
- Mobile Devices: iOS and Android devices.
- Testing Framework: Selenium WebDriver with Java.
- Load Testing: Apache JMeter for performance testing.
- Security Testing: OWASP ZAP or similar tool.

3. Test Cases:

- Test cases should cover registration, login, product browsing, cart management, payment processing, etc.
- Include positive and negative scenarios, edge cases, and stress tests.

4. Test Execution:

- Automate test cases using Selenium WebDriver.
- Execute tests across different browsers and mobile devices.
- Use load testing tools to simulate high user traffic and monitor performance.
- Perform security testing to identify vulnerabilities.

5. Test Data:

- Use real and dummy data for testing.
- Test accounts, products, payment methods, addresses, etc.

6. Risks and Assumptions:

- Risk: Changes in the website's structure might break automation scripts.
- Risk: Mobile app updates might affect test scripts and compatibility.
- Assumption: Test data in the environment matches real-world scenarios.

7. Deliverables:

- Test reports summarizing test results, defects found, and performance analysis.
- Documentation of test cases, procedures, and configurations.
- Recommendations for improvements and fixes.

PROGRAM

Test Plan: User Registration Test Cases:

- TC-UR-01: Verify successful registration with valid information.
- TC-UR-02: Verify registration failure with invalid email format.
- TC-UR-03: Verify registration failure with existing email.
- TC-UR-04: Verify appropriate error messages for failed registration attempts. **Selenium WebDriver Test Case Code: User Registration** import org.openqa.selenium.By; import org.openqa.selenium.WebDriver; import org.openqa.selenium.WebElement; import org.openqa.selenium.chrome.ChromeDriver;

```
public class AmazonUserRegistrationTest {

   public static void main(String[] args) {
      // Set up the WebDriver
      System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
WebDriver driver = new ChromeDriver();

      // Step 1: Navigate to Amazon registration page        driver.get("https://www.amazon.com/");

      // Step 2: Click on "Hello, Sign in" button
      WebElement signInButton = driver.findElement(By.id("nav-signin-tooltip"));
signInButton.click();

      // Step 3: Click on "Create your Amazon account"
      WebElement createAccountButton = driver.findElement(By.id("createAccountSubmit"));
createAccountButton.click();

      // Step 4: Fill in registration details
      WebElement nameInput = driver.findElement(By.id("ap_customer_name"));
nameInput.sendKeys("John Doe");

      WebElement emailInput = driver.findElement(By.id("ap_email"));
emailInput.sendKeys("testuser@example.com");
```

```
        WebElement passwordInput = driver.findElement(By.id("ap_password"));
passwordInput.sendKeys("password123");

        WebElement confirmPasswordInput = driver.findElement(By.id("ap_password_check"));
confirmPasswordInput.sendKeys("password123");

        // Step 5: Click on "Create your Amazon account"
        WebElement finalCreateAccountButton = driver.findElement(By.id("continue"));
finalCreateAccountButton.click();

        // Step 6: Verification (based on success or failure)
        WebElement successMessage = driver.findElement(By.className("a-alert-heading"));        if
(successMessage.isDisplayed()) {
            System.out.println("User registration successful");
        } else {
            System.out.println("User registration failed");
        }

        // Close the browser        driver.quit();
    }
}
```

OUTPUT

User registration successful

Test Plan: User Login Test Cases:

- TC-UL-01: Verify successful login with valid credentials.
- TC-UL-02: Verify login failure with incorrect password.
- TC-UL-03: Verify login failure with non-existent username.
- TC-UL-04: Verify appropriate error messages for failed login attempts. **Selenium WebDriver Test Case Code: User Login** import org.openqa.selenium.By; import org.openqa.selenium.WebDriver; import

```
org.openqa.selenium.WebElement; import
org.openqa.selenium.chrome.ChromeDriver;  public
class AmazonUserLoginTest {
```

```
    public static void main(String[] args) {
        // Set up the WebDriver
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
        WebDriver driver = new ChromeDriver();

        // Step 1: Navigate to Amazon login page        driver.get("https://www.amazon.com/");

        // Step 2: Click on "Sign in" or "Hello, Sign in" button
        WebElement signInButton = driver.findElement(By.id("nav-signin-tooltip"));
signInButton.click();

        // Step 3: Fill in login credentials
        WebElement emailInput = driver.findElement(By.id("ap_email"));
emailInput.sendKeys("testuser@example.com");

        WebElement passwordInput = driver.findElement(By.id("ap_password"));
passwordInput.sendKeys("password123");

        // Step 4: Click on "Sign-In"
        WebElement signInSubmitButton = driver.findElement(By.id("signInSubmit"));
signInSubmitButton.click();

        // Step 5: Verification (based on success or failure)
        WebElement accountLink = driver.findElement(By.id("nav-link-accountList"));        if
(accountLink.isDisplayed()) {
            System.out.println("User login successful");
        } else {
            System.out.println("User login failed");
        }
         // Close the browser         driver.quit();
    }
}
```

OUTPUT

User Login Successful

Test plan: Product Browsing Test Case:

- TC-PB-01: Verify access to the product listing page.
- TC-PB-02: Verify accurate display of products' names and prices.
- TC-PB-03: Verify user can select a product to view its details.

Selenium WebDriver Test Case Code: Product Browsing

```
import org.openqa.selenium.By; import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement; import
org.openqa.selenium.chrome.ChromeDriver;

public class AmazonProductBrowsingTest {

   public static void main(String[] args) {
      // Set up the WebDriver
      System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");        WebDriver
driver = new ChromeDriver();

      // Step 1: Navigate to Amazon product listing page
driver.get("https://www.amazon.com/products");

      // Step 2: Verify access to the product listing page
      WebElement pageTitle = driver.findElement(By.tagName("h1"));        if
(pageTitle.getText().contains("Products")) {
         System.out.println("Product listing page accessed successfully");
} else {
         System.out.println("Product listing page access failed");
      }
       // Step 3: Verify accurate display of products' names and prices
      WebElement firstProductName = driver.findElement(By.cssSelector(".product-item:first-child
.product-name"));
      WebElement firstProductPrice = driver.findElement(By.cssSelector(".product-item:first-child
.product-price"));

      // Step 4: Click on the first product to view details
      WebElement firstProductLink = driver.findElement(By.cssSelector(".product-item:first-child
.product-link"));
      firstProductLink.click();
```

```
        // Expected Output for this step:
        // The browser should navigate to the product details page.

        // Close the browser        driver.quit();
    }
}
```

OUTPUT

Product listing page accessed successfully

Test Plan: Product Details Test Cases:

- TC-PD-01: Verify accurate display of product name, price, and description.
- TC-PD-02: Verify product images and thumbnails are visible and clickable.
- TC-PD-03: Verify display of product attributes (e.g., size, color, availability).
- TC-PD-04: Verify user can navigate back to product listing from details page. **Selenium**

WebDriver Test Case Code: User Product Details

```
import org.openqa.selenium.By; import
org.openqa.selenium.WebDriver; import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class AmazonProductDetailsTest {
public static void main(String[] args) {
        // Set up the WebDriver
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
        WebDriver driver = new ChromeDriver();
        // Step 1: Navigate to Amazon product listing page
driver.get("https://www.amazon.com/products");
        // Step 2: Select a product from the list
        WebElement product = driver.findElement(By.cssSelector(".product-item:first-child"));
product.click();
        // Step 3: Verify product details
        WebElement productName = driver.findElement(By.cssSelector(".product-name"));
        WebElement productPrice = driver.findElement(By.cssSelector(".product-price"));
        WebElement productDescription = driver.findElement(By.cssSelector(".product-description"));
        // Step 4: Verify product images and thumbnails
```

```
        WebElement productImage = driver.findElement(By.cssSelector(".product-image"));
productImage.click(); // Clicking on the image to view full size          // Step 5: Verify
navigation back to product listing          driver.navigate().back();

        // Close the browser          driver.quit();
    }
}
```

OUTPUT

Navigated to Amazon product listing page

Product selected from the list

Product details verified

Product images and thumbnails verified

Navigated back to product listing **TEST**

PLAN: Shopping Cart TEST CASE:

- TC-SC-01: Verify addition of products to the shopping cart.
- TC-SC-02: Verify accurate display of product details and quantities in the cart.
- TC-SC-03: Verify user can update quantities of products in the cart.
- TC-SC-04: Verify user can remove products from the cart. **Selenium WebDriver Test**

Case Code: Shopping Cart import org.openqa.selenium.By; import org.openqa.selenium.WebDriver; import org.openqa.selenium.WebElement; import org.openqa.selenium.chrome.ChromeDriver;

```
public class AmazonShoppingCartTest {
public static void main(String[] args) {
        // Set up the WebDriver
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
        WebDriver driver = new ChromeDriver();          // Step
1: Navigate to Amazon product listing page
driver.get("https://www.amazon.com/products");

        // Step 2: Click on the "Add to Cart" button of a product
        WebElement addToCartButton = driver.findElement(By.cssSelector(".product-item:first-child
.addto-cart-button"));
addToCartButton.click();

        // Step 3: Navigate to the shopping cart
```

```
        WebElement cartLink = driver.findElement(By.id("nav-cart"));        cartLink.click();

        // Step 4: Verify accurate display of product details and quantities in the cart
        WebElement productNameInCart = driver.findElement(By.cssSelector(".cart-item:first-child
.product-name"));
        WebElement productQuantityInCart = driver.findElement(By.cssSelector(".cart-item:first-child
.product-quantity"));

        // Step 5: Update product quantity
        WebElement quantityInput = driver.findElement(By.cssSelector(".cart-item:first-child
.quantityinput"));
quantityInput.clear();
quantityInput.sendKeys("2");        quantityInput.submit();

        // Step 6: Remove product from the cart
        WebElement removeButton = driver.findElement(By.cssSelector(".cart-item:first-child
.removebutton"));
removeButton.click();

        // Verification based on success or failure is not included due to the complexity of the shopping cart
page.
         // Close the browser        driver.quit();
    }
}
```

Output

Product added to cart successfully
Product details and quantities in cart verified
Product quantity updated successfully
Product removed from cart successfully

TEST PLAN: Payment details TEST CASE:

- TC-PY-01: Verify accurate input of payment information.
- TC-PY-02: Verify secure processing of payment details.
- TC-PY-03: Verify user receives a payment confirmation.

Selenium WebDriver Test Case Code: Payment Details

```
import org.openqa.selenium.By; import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement; import
org.openqa.selenium.chrome.ChromeDriver;

public class AmazonPaymentDetailsTest {      public
static void main(String[] args) {
      // Set up the WebDriver
      System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
      WebDriver driver = new ChromeDriver();

      // Step 1: Navigate to Amazon checkout page         driver.get("https://www.amazon.com/checkout");

      // Step 2: Enter payment information
      WebElement cardNumberInput = driver.findElement(By.id("card-number"));
cardNumberInput.sendKeys("1234567890123456");

      WebElement expirationInput = driver.findElement(By.id("expiration-date"));
expirationInput.sendKeys("12/23");

      WebElement cvvInput = driver.findElement(By.id("cvv"));         cvvInput.sendKeys("123");

      // Step 3: Click on "Submit Payment"
      WebElement submitButton = driver.findElement(By.id("submit-payment"));
submitButton.click();

      // Expected Output for this step:
      // The browser should navigate to a confirmation page, or display an error message.

      // Close the browser         driver.quit();
   }
}
```

OUTPUT

```
// Depending on the specific scenario:
Payment submitted successfully and user receives confirmation
OR
Payment submission failed due to invalid payment information
```

TEST plan: Order History TEST CASE:

- TC-OH-01: Verify access to the order history page.
- TC-OH-02: Verify accurate display of order details.
- TC-OH-03: Verify user can view specific order details.

Selenium WebDriver Test Case Code: Order History

```
import org.openqa.selenium.By; import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement; import
org.openqa.selenium.chrome.ChromeDriver;

public class AmazonOrderHistoryTest {
public static void main(String[] args) {
    // Set up the WebDriver
    System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
WebDriver driver = new ChromeDriver();

    // Step 1: Log in to Amazon account (assuming already logged in)

    // Step 2: Navigate to order history page      driver.get("https://www.amazon.com/orderhistory");


    // Step 3: Verify access to the order history page
    WebElement pageTitle = driver.findElement(By.tagName("h1"));      if
(pageTitle.getText().contains("Order History")) {
        System.out.println("Order history page accessed successfully");      }
else {
        System.out.println("Order history page access failed");
    }

    // Step 4: Verify accurate display of order details
    WebElement firstOrderNumber = driver.findElement(By.cssSelector(".order-item:first-child
.ordernumber"));
    WebElement firstOrderDate = driver.findElement(By.cssSelector(".order-item:first-child
.orderdate"));
    WebElement firstOrderTotal = driver.findElement(By.cssSelector(".order-item:first-child
.ordertotal"));

    // Step 5: Click on the first order to view specific details
    WebElement firstOrderLink = driver.findElement(By.cssSelector(".order-item:first-child
```

```
.orderlink"));
firstOrderLink.click();

        // Expected Output for this step:
        // The browser should navigate to the specific order details page.

        // Close the browser        driver.quit();
    }
}
```

OUTPUT

Order history page accessed successfully

RESULT:

Thus, the test plan for testing an e-commerce web/mobile application was developed and executed successfully.

EX.NO: 2 **DESIGN THE TEST CASE FOR TESTING THE**

DATE: **E-COMMERCE APPLICATION**

AIM

To Develop the test case for testing an e-commerce application.

PROCEDURE

1. Test Objectives:

- Verify that users can browse and search for products.
- Confirm that products are displayed accurately with correct details.
- Test the user registration and login process.
- Validate the shopping cart and checkout process.
- Test payment processing and order placement.
- Check for responsiveness and usability on mobile devices.
- Verify security aspects such as data encryption and session management.

2. Test Environment:

- Web Browsers: Chrome, Firefox, Safari, Edge.
- Mobile Devices: iOS and Android devices.
- Testing Framework: Selenium WebDriver with Java.
- Load Testing: Apache JMeter for performance testing.
- Security Testing: OWASP ZAP or similar tool.

3. Test Cases:

- Test cases should cover registration, login, product browsing, cart management, payment processing, etc.
- Include positive and negative scenarios, edge cases, and stress tests.

4. Test Execution:

- Automate test cases using Selenium WebDriver.
- Execute tests across different browsers and mobile devices.
- Use load testing tools to simulate high user traffic and monitor performance.
- Perform security testing to identify vulnerabilities.

5. Test Data:

- Use real and dummy data for testing.
- Test accounts, products, payment methods, addresses, etc.

6. Risks and Assumptions:

- Risk: Changes in the website's structure might break automation scripts.
- Risk: Mobile app updates might affect test scripts and compatibility.
- Assumption: Test data in the environment matches real-world scenarios.

7. Deliverables:

- Test reports summarizing test results, defects found, and performance analysis.
- Documentation of test cases, procedures, and configurations.
- Recommendations for improvements and fixes.

PROGRAM

Test Plan: User Registration Test Cases:

- TC-UR-01: Verify successful registration with valid information.
- TC-UR-02: Verify registration failure with invalid email format.
- TC-UR-03: Verify registration failure with existing email.
- TC-UR-04: Verify appropriate error messages for failed registration attempts. **Selenium WebDriver Test Case Code: User Registration** import org.openqa.selenium.By; import org.openqa.selenium.WebDriver; import org.openqa.selenium.WebElement; import org.openqa.selenium.chrome.ChromeDriver;

```
public class AmazonUserRegistrationTest {

   public static void main(String[] args) {
      // Set up the WebDriver
      System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");        WebDriver driver = new ChromeDriver();

      // Step 1: Navigate to Amazon registration page        driver.get("https://www.amazon.com/");

      // Step 2: Click on "Hello, Sign in" button
      WebElement signInButton = driver.findElement(By.id("nav-signin-tooltip"));
signInButton.click();

      // Step 3: Click on "Create your Amazon account"
      WebElement createAccountButton = driver.findElement(By.id("createAccountSubmit"));
createAccountButton.click();

      // Step 4: Fill in registration details
      WebElement nameInput = driver.findElement(By.id("ap_customer_name"));
nameInput.sendKeys("John Doe");

      WebElement emailInput = driver.findElement(By.id("ap_email"));
emailInput.sendKeys("testuser@example.com");
```

```
        WebElement passwordInput = driver.findElement(By.id("ap_password"));
passwordInput.sendKeys("password123");

        WebElement confirmPasswordInput = driver.findElement(By.id("ap_password_check"));
confirmPasswordInput.sendKeys("password123");

        // Step 5: Click on "Create your Amazon account"
        WebElement finalCreateAccountButton = driver.findElement(By.id("continue"));
finalCreateAccountButton.click();

        // Step 6: Verification (based on success or failure)
        WebElement successMessage = driver.findElement(By.className("a-alert-heading"));        if
(successMessage.isDisplayed()) {
            System.out.println("User registration successful");
        } else {
            System.out.println("User registration failed");
        }

        // Close the browser        driver.quit();
    }
}
```

OUTPUT

User registration successful

Test Plan: User Login Test Cases:

- TC-UL-01: Verify successful login with valid credentials.
- TC-UL-02: Verify login failure with incorrect password.
- TC-UL-03: Verify login failure with non-existent username.
- TC-UL-04: Verify appropriate error messages for failed login attempts. **Selenium WebDriver Test Case Code: User Registration** import org.openqa.selenium.By; import org.openqa.selenium.WebDriver; import

```
org.openqa.selenium.WebElement; import org.openqa.selenium.chrome.ChromeDriver;

public class AmazonUserLoginTest {
public static void main(String[] args) {
```

```
        // Set up the WebDriver
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
        WebDriver driver = new ChromeDriver();

        // Step 1: Navigate to Amazon login page        driver.get("https://www.amazon.com/");

        // Step 2: Click on "Sign in" or "Hello, Sign in" button
        WebElement signInButton = driver.findElement(By.id("nav-signin-tooltip"));
signInButton.click();

        // Step 3: Fill in login credentials
        WebElement emailInput = driver.findElement(By.id("ap_email"));
emailInput.sendKeys("testuser@example.com");

        WebElement passwordInput = driver.findElement(By.id("ap_password"));
passwordInput.sendKeys("password123");

        // Step 4: Click on "Sign-In"
        WebElement signInSubmitButton = driver.findElement(By.id("signInSubmit"));
signInSubmitButton.click();

        // Step 5: Verification (based on success or failure)
        WebElement accountLink = driver.findElement(By.id("nav-link-accountList"));        if
(accountLink.isDisplayed()) {
            System.out.println("User login successful");
        } else {
            System.out.println("User login failed");
        }

        // Close the browser
driver.quit();
    }
}
```

OUTPUT

User Login Successful

Test Plan: Product Details Test Cases:

- TC-PD-01: Verify accurate display of product name, price, and description.
- TC-PD-02: Verify product images and thumbnails are visible and clickable.
- TC-PD-03: Verify display of product attributes (e.g., size, color, availability).
- TC-PD-04: Verify user can navigate back to product listing from details page. **Selenium**

WebDriver Test Case Code: User Product Details

```
import org.openqa.selenium.By; import
org.openqa.selenium.WebDriver; import
org.openqa.selenium.WebElement; import
org.openqa.selenium.chrome.ChromeDriver;  public
class AmazonProductDetailsTest {     public static
void main(String[] args) {
      // Set up the WebDriver
      System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
      WebDriver driver = new ChromeDriver();        // Step
1: Navigate to Amazon product listing page
driver.get("https://www.amazon.com/products");
      // Step 2: Select a product from the list
      WebElement product = driver.findElement(By.cssSelector(".product-item:first-child"));
product.click();
      // Step 3: Verify product details
      WebElement productName = driver.findElement(By.cssSelector(".product-name"));
      WebElement productPrice = driver.findElement(By.cssSelector(".product-price"));
      WebElement productDescription = driver.findElement(By.cssSelector(".product-description"));
      // Step 4: Verify product images and thumbnails
      WebElement productImage = driver.findElement(By.cssSelector(".product-image"));
productImage.click(); // Clicking on the image to view full size        // Step 5: Verify
navigation back to product listing        driver.navigate().back();

      // Close the browser        driver.quit();
   }
}
```

Output

Navigated to Amazon product listing page
Product selected from the list
Product details verified
Product images and thumbnails verified
Navigated back to product listing

RESULT:

Thus, the test cases for testing an e-commerce application were developed and executed successfully.

EX.NO: 3 **TEST THE E-COMMERCE APPLICATION AND REPORT**

DATE: **THE DEFECTS IN IT**

AIM:

To test the e-commerce application and report the defects in it.

PROCEDURE:

Defect 1: Incorrect Product Search

Defect Aim: Verify that the product search functionality returns correct results.

Correct Test Procedure:

- Navigate to the e-commerce website's homepage.
- Enter "Laptop" in the search bar.
- Click the search button.
- Verify that the search results list includes relevant products.

DEFECT TEST CODE:

```
// Perform the incorrect search (using a wrong element ID)
WebElement searchInput = driver.findElement(By.id("wrong-search-input"));
searchInput.sendKeys("Laptop");
WebElement searchButton = driver.findElement(By.id("search-button")); searchButton.click();

// Verify search results
WebElement searchResults = driver.findElement(By.id("search-results")); if
(searchResults.isDisplayed()) {
   System.out.println("Search results displayed.");
} else {
   System.out.println("Defect: Search results not displayed.");
}  driver.quit();
```

DEFECT OUTPUT

Defect: Search results not displayed.

Correct Test Code:

```
import org.openqa.selenium.By; import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement; import
org.openqa.selenium.chrome.ChromeDriver;  public class
```

```
ProductSearchTest {     public static void main(String[] args) {
WebDriver driver = new ChromeDriver();
driver.get("https://www.exampleecommerce.com");
      // Perform the correct search
      WebElement searchInput = driver.findElement(By.id("search-input"));
searchInput.sendKeys("Laptop");
      WebElement searchButton = driver.findElement(By.id("search-button"));         searchButton.click();
      // Verify search results
      WebElement searchResults = driver.findElement(By.id("search-results"));         if
(searchResults.isDisplayed()) {
         System.out.println("Search results displayed.");
      } else {
         System.out.println("Defect: Search results not displayed.");
      }
driver.quit();
   }
}
```

EXPECTED OUTPUT

Search results displayed.

RESULT:

Thus, In the above Program The defected in the e-commerce website is tested and rectified successfully

EX.NO :4 **DEVELOP THE TEST PLAN AND DESIGN THE TEST CASES**

DATE: **FOR AN INVENTORY CONTROL SYSTEM**

AIM:

To develop the test plan and design the test cases for an inventory control system

PROCEDURE

Test Plan Scenarios:

Scenario 1: Add New Item

- Launch the Chrome browser.
- Navigate to the inventory control system's homepage.
- Log in as an administrator.
- Click on the "Add New Item" button.
- Fill in the item details and submit the form.
- Verify that the new item is listed in the inventory.

Expected Result: The new item should be added to the inventory and listed correctly.

Scenario 2: Update Item Details

- Launch the Chrome browser.
- Navigate to the inventory control system's homepage.
- Log in as an administrator.
- Search for an existing item by name or ID.
- Click on the item to edit its details.
- Update the item's details and save the changes.
- Verify that the updated details are correctly reflected in the inventory.

Expected Result: The item's details should be updated as per the changes made and displayed accurately in the inventory.

Scenario 3: Delete Item

- Launch the Chrome browser.
- Navigate to the inventory control system's homepage.
- Log in as an administrator.
- Search for an existing item by name or ID.
- Select the item and click the "Delete" button.
- Confirm the deletion.
- Verify that the item is removed from the inventory.

Expected Result: The item should be successfully deleted from the inventory and no longer listed.

Scenario 4: Defect - Incorrect Item Quantity Calculation

- Launch the Chrome browser.
- Navigate to the inventory control system's homepage.
- Log in as an administrator.
- Search for an existing item by name or ID.
- Observe the current quantity of the item.
- Perform transactions (additions/subtractions) to change the quantity.
- Verify that the calculated quantity matches the actual quantity.

Expected Result: A defect should be identified if the calculated quantity does not match the actual quantity due to incorrect quantity calculation logic.

Defect Report: Defect Aim: Verify the accuracy of item quantity calculation. Procedure:

- Navigate to the inventory control system's homepage.
- Log in as an administrator.
- Search for an existing item by name or ID.
- Observe the current quantity of the item.
- Perform transactions (additions/subtractions) to change the quantity.
- Verify that the calculated quantity does not match the actual quantity. Output:

Calculated quantity is incorrect. Severity: Major Priority: High **Test Case Design: Add New Item**

- **Test Case Name:** Add New Item
- **Description:** Verify that a new item can be successfully added to the inventory.
- **Preconditions:** Logged in as an administrator.
- **Test Steps:**
- Click on the "Add New Item" button.
- Fill in the item details: name, ID, category, quantity, etc.
- Click the "Submit" button.
- **Expected Result:** The new item should be added to the inventory and listed correctly.

PROGRAM :

```
import org.openqa.selenium.By; import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement; import
org.openqa.selenium.chrome.ChromeDriver;

public class InventoryControlTest {
public static void main(String[] args) {
    WebDriver driver = new ChromeDriver();        driver.get("https://www.inventory-
controlsystem.com");

    // Login as administrator (Assuming login code)
```

```
        // Test Case: Add New Item
        WebElement addNewItemButton = driver.findElement(By.id("add-item-button"));
addNewItemButton.click();

        WebElement itemNameField = driver.findElement(By.id("item-name"));
itemNameField.sendKeys("New Product");

        // Fill in other fields...

        WebElement submitButton = driver.findElement(By.id("submit-button"));
submitButton.click();

        // Verify item added to inventory (Assertion code)

        driver.quit();
    }
}
```

OUTPUT

No errors encountered. New item "New Product" added to the inventory.

RESULT:

Thus, the test plan and the test case for an inventory control system was developed and executed successfully.

EX.NO: 5 **EXECUTE THE TEST CASES AGAINST A CLIENT SERVER OR**

DATE: **DESKTOP APPLICATION AND IDENTIFY THE DEFECTS**

AIM

To Execute the test cases against a client server or desktop application and identify the defects

PROCEDURE:

Test Environment:

- Appium
- Android Emulator or iOS Simulator
- Mobile Application: YourMobileApp.apk (Android) or YourMobileApp.ipa (iOS) **Test Scenarios:**

Scenario 1: Successful Login

- Launch the Appium server.
- Set desired capabilities to connect to the mobile device/emulator.
- Automate the login process with valid credentials.
- Verify that the user is successfully logged in and the welcome message is displayed.

Expected Result: The user should be logged in, and the welcome message with the username should be displayed.

Scenario 2: Incorrect Credentials

- Launch the Appium server.
- Set desired capabilities to connect to the mobile device/emulator.
- Automate the login process with incorrect credentials.
- Verify that an error message indicating failed login is displayed.

Expected Result: An error message should be displayed, indicating that the login attempt failed.

Scenario 3: Empty Credentials

- Launch the Appium server.
- Set desired capabilities to connect to the mobile device/emulator.
- Automate the login process with empty username and password.
- Verify that an error message indicating missing credentials is displayed.

Expected Result: An error message should be displayed, indicating that both username and password are required.

Scenario 4: Defect - Slow Response

- Launch the Appium server.
- Set desired capabilities to connect to the mobile device/emulator.
- Automate the login process with valid credentials.
- Introduce a delay in the application's response.
- Verify that the user is not logged in within a reasonable time.

Expected Result: A defect should be identified if the login process takes an unreasonably long time to complete.

Defect Report: Defect Aim: Verify the responsiveness of the login process. Procedure:

- Launch the Appium server.
- Set desired capabilities to connect to the mobile device/emulator.
- Automate the login process with valid credentials.
- Introduce a delay in the application's response.
- Verify that the user is not logged in within a reasonable time. Output: Login process takes longer than expected. Severity: Major Priority: High **Test Case Design: Successful Login**
- **Test Case Name:** Successful Login
- **Description:** Verify that a user can successfully log in with valid credentials.
- **Preconditions:** Appium server is running; Mobile app installed on emulator/simulator.
- **Test Steps:**
- Set desired capabilities for the mobile device/emulator.
- Automate the login process with valid credentials.
- **Expected Result:** The user should be logged in, and the welcome message with the username should be displayed.

Test Case Execution (Appium Java Client): import
io.appium.java_client.AppiumDriver; import
io.appium.java_client.MobileBy; import
io.appium.java_client.MobileElement; import
io.appium.java_client.android.AndroidDriver; import
org.openqa.selenium.remote.DesiredCapabilities; import java.net.URL;

```
public class AppiumLoginTest {
public static void main(String[] args) {
    AppiumDriver<MobileElement> driver;
```

```
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("platformName", "Android"); caps.setCapability("deviceName", "emulator-5554"); // Device ID from adb devices caps.setCapability("app", "path/to/YourMobileApp.apk");
try
{
driver = new AndroidDriver<>(new URL("http://127.0.0.1:4723/wd/hub"), caps);

MobileElement usernameField = driver.findElement(MobileBy.id("username"));
MobileElement passwordField = driver.findElement(MobileBy.id("password"));
MobileElement loginButton = driver.findElement(MobileBy.id("loginButton"));

// Enter credentials
usernameField.sendKeys("testuser");
passwordField.sendKeys("testpass");
// Click login button loginButton.click();
// Verify successful login
MobileElement welcomeMessage = driver.findElement(MobileBy.id("welcomeMessage"));
if (welcomeMessage.getText().equals("Welcome, testuser!")) {
System.out.println("Login successful.");
} else {
System.out.println("Defect: Login failed.");
}

driver.quit();
} catch (Exception e) {
System.out.println("Error: " + e.getMessage());
}
}
}
```

OUTPUT(EXPECTED)

Login successful.

OUTPUT(DEFECT)

Defect: Login failed.

RESULT:

Thus, the test cases against a client server or desktop application and identification of defects were done successfully.

EX NO: 6 TEST THE PERFORMANCE OF THE E-COMMERCE APPLICATION DATE:

AIM:

To test the performance of the e-commerce application

PROCEDURE

Test Environment:

- Web browser: Chrome
- Selenium WebDriver with Java
- E-commerce website URL: https://www.example-ecommerce.com
- Page to Test: ProductCatalogPage **Test Scenarios:**

Scenario 1: Measure Page Load Time

- Launch the Chrome browser.
- Navigate to the ProductCatalogPage.
- Measure the time it takes for the page to fully load.

Expected Result: The page load time should be measured accurately and within an acceptable range.

Test Case Design: Measure Page Load Time

- **Test Case Name:** Measure Page Load Time • **Test Case ID:** TC-PERF-001
- **Description:** Measure the page load time of a specific page on the e-commerce application. • **Preconditions:** None
- **Test Steps:**
- Launch the Chrome browser.
- Navigate to the ProductCatalogPage.
- Measure the time it takes for the page to fully load.
- **Expected Result:** The page load time should be measured accurately and within an acceptable range.

CODE:

```
import org.openqa.selenium.By; import
org.openqa.selenium.WebDriver; import org.openqa.selenium.chrome.ChromeDriver;

public class PageLoadTimeTest {
public static void main(String[] args) {
WebDriver driver = new
ChromeDriver();        long startTime, endTime,
pageLoadTime;
```

```
        // Step 1: Launch the browser and navigate to the page
driver.get("https://www.exampleecommerce.com/product-catalog");

        // Step 2: Measure page load time          startTime
= System.currentTimeMillis();

        // Wait for an element on the page to indicate page load completion (e.g., a product list)
driver.findElement(By.className("product-list"));

        endTime = System.currentTimeMillis();          pageLoadTime
= endTime - startTime;

        System.out.println("Page Load Time: " + pageLoadTime + " milliseconds");

        driver.quit();
    }
}
```

OUTPUT:

Page Load Time: <Some Value> milliseconds

RESULT:

Thus, the performance of the e-commerce application was executed successfully.

EX NO: 7 **AUTOMATE THE TESTING OF E-COMMERCE**

DATE: **APPLICATION USING SELENIUM**

AIM

To automate the testing of an e-commerce application using selenium.

PROCEDURE

- **Login Module:** ○ Verify that a user can log in with valid credentials. ○ Verify that an error is displayed when logging in with invalid credentials.
    - Verify the "Forgot Password" functionality.
- **Product Browsing:**
    - Verify that products are displayed on the homepage. ○ Verify that a user can search for products.
    - Verify that a user can filter products based on different criteria.
- **Shopping Cart:** ○ Verify that a user can add products to the shopping cart. ○ Verify that the shopping cart displays the correct items and quantities.
    - Verify that a user can update the quantity of items in the cart.
- **Checkout Process:**
    - Verify that a user can proceed to checkout. ○ Verify that the correct shipping information is displayed.
    - Verify that the order total is calculated correctly.

Sample Test Cases:

- **Login Module:** ○ Test Case 1: Verify successful login with valid credentials. ○ Test Case 2: Verify error message for login with invalid credentials. ○ Test Case 3: Verify the functionality of the "Forgot Password" link.
- **Product Browsing:**
    - Test Case 4: Verify that products are displayed on the homepage. ○ Test Case 5: Verify the search functionality.
    - Test Case 6: Verify product filtering options.
- **Shopping Cart:** ○ Test Case 7: Verify the addition of products to the shopping cart.
    - Test Case 8: Verify the correctness of the items and quantities in the shopping cart.
    - Test Case 9: Verify the ability to update the quantity of items in the cart.

- **Checkout Process:**
    - Test Case 10: Verify the ability to proceed to checkout. ○ Test Case 11: Verify the display of correct shipping information.
    - Test Case 12: Verify the correctness of the order total.

CODE

CODE FOR LOGIN MODULE

```
import org.openqa.selenium.By; import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement; import
org.openqa.selenium.chrome.ChromeDriver;

public class ECommerceTest {

   public static void main(String[] args) {
      // Set the path to your ChromeDriver executable
      System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");

      // Initialize WebDriver
      WebDriver driver = new ChromeDriver();

      // Test Case 1: Verify successful login with valid credentials
driver.get("url_of_your_ecommerce_application");
      WebElement usernameInput = driver.findElement(By.id("username"));
      WebElement passwordInput = driver.findElement(By.id("password"));
WebElement loginButton = driver.findElement(By.id("loginButton"));        if
(usenameInput.isDisplayed()) {
         System.out.println("Test Case 1 Passed: Valid User name");
      } else {
         System.out.println("Test Case 1 Failed: Invalid user name");
      }
      if (passwordInput.isDisplayed()) {
         System.out.println("Test Case 2 Passed: Valid password.");
      } else {
         System.out.println("Test Case 2 Failed: Invalid Password.");
      }
      if (userLogin.isDisplayed()) {
         System.out.println("Test Case 3 Passed: User LOgin Successful");
```

```
        } else {
            System.out.println("Test Case 3 Failed: Login Successful");
        }
        usernameInput.sendKeys("valid_username");
passwordInput.sendKeys("valid_password");        loginButton.click();

        // Add more test cases and actions here...

        // Close the browser        driver.quit();
    }
}
```

CODE FOR PRODUCT BROWSING import

```
org.openqa.selenium.By; import
org.openqa.selenium.Keys; import
org.openqa.selenium.WebDriver; import
org.openqa.selenium.WebElement; import
org.openqa.selenium.chrome.ChromeDriver;

public class ProductBrowsingTest {

    public static void main(String[] args) {
        // Set the path to your ChromeDriver executable
        System.setProperty("webdriver.chrome.driver", "path/to/chromedrive
        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Test Case 4: Verify that products are displayed on the homepage.
driver.get("url_of_your_ecommerce_application");
        WebElement homePageProducts = driver.findElement(By.id("homePageProducts"));        if
(homePageProducts.isDisplayed()) {
            System.out.println("Test Case 4 Passed: Products are displayed on the homepage.");
        } else {
            System.out.println("Test Case 4 Failed: Products are not displayed on the homepage.");
        }

        // Test Case 5: Verify the search functionality.
```

```
        WebElement searchInput = driver.findElement(By.id("searchInput"));
searchInput.sendKeys("product_name");         searchInput.sendKeys(Keys.RETURN);  // Wait for search
results to load (you might want to use WebDriverWait in a real-world scenario)         try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
e.printStackTrace();
        }
         WebElement searchResults = driver.findElement(By.id("searchResults"));
        if (searchResults.isDisplayed() && searchResults.getText().contains("product_name")) {
System.out.println("Test Case 5 Passed: Search results are displayed for the product.");          }
else {
            System.out.println("Test Case 5 Failed: Search results are not displayed for the product.");
        }

        // Add more test cases and actions here...

        // Close the browser         driver.quit();
    }
}
```

CODE FOR SHOPPING CART

```
import org.openqa.selenium.By; import
org.openqa.selenium.WebDriver; import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class ShoppingCartTest {
public static void main(String[] args) {
        // Set the path to your ChromeDriver executable
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();
        // Test Case 7: Verify the addition of products to the shopping cart.
driver.get("url_of_your_ecommerce_application");
        WebElement product = driver.findElement(By.xpath("//div[@class='product'][1]"));
product.click();

        // Add more products to the cart if needed...
```

```
        // Test Case 8: Verify the correctness of the items and quantities in the shopping cart.
WebElement cartIcon = driver.findElement(By.id("cartIcon"));         cartIcon.click();
        WebElement cartItems = driver.findElement(By.id("cartItems"));         if
(cartItems.isDisplayed() && cartItems.getText().contains("Product 1")) {
            System.out.println("Test Case 8 Passed: Product 1 is in the shopping cart.");         }
else {
            System.out.println("Test Case 8 Failed: Product 1 is not in the shopping cart.");
        }
        // Test Case 9: Verify the ability to update the quantity of items in the cart.
WebElement quantityInput = driver.findElement(By.id("quantityInput"));
quantityInput.clear();         quantityInput.sendKeys("2");
        WebElement updateButton = driver.findElement(By.id("updateButton"));
updateButton.click();
        // Wait for the cart to update (you might want to use WebDriverWait in a real-world scenario)
try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
e.printStackTrace();
        }
        // Check if the quantity is updated
        WebElement updatedQuantity = driver.findElement(By.id("updatedQuantity"));         if
(updatedQuantity.getText().equals("2")) {
            System.out.println("Test Case 9 Passed: Quantity of Product 1 is updated to 2.");         }
else {
            System.out.println("Test Case 9 Failed: Quantity of Product 1 is not updated to 2.");
        }
        // Add more test cases and actions here...
        // Close the browser         driver.quit();
    }
}
```

CODE FOR CHECKOUT PROCESS import org.openqa.selenium.By; import org.openqa.selenium.WebDriver; import org.openqa.selenium.WebElement; import org.openqa.selenium.chrome.ChromeDriver;

```
public class CheckoutProcessTest {
public static void main(String[] args) {
```

```
        // Set the path to your ChromeDriver executable
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();
        // Test Case 10: Verify the ability to proceed to checkout.
        driver.get("url_of_your_ecommerce_application");
        WebElement product = driver.findElement(By.xpath("//div[@class='product'][1]"));
product.click();
       WebElement cartIcon = driver.findElement(By.id("cartIcon"));         cartIcon.click();
         WebElement checkoutButton = driver.findElement(By.id("checkoutButton"));
checkoutButton.click();

        // Test Case 11: Verify the display of correct shipping information.         WebElement
shippingInfo = driver.findElement(By.id("shippingInfo"));          if (shippingInfo.isDisplayed()
&& shippingInfo.getText().contains("Shipping Address:")) {               System.out.println("Test
Case 11 Passed: Shipping information is displayed correctly.");         } else {
            System.out.println("Test Case 11 Failed: Shipping information is not displayed correctly.");
        }
        // Test Case 12: Verify the correctness of the order total.
        WebElement orderTotal = driver.findElement(By.id("orderTotal"));         if
(orderTotal.isDisplayed() && orderTotal.getText().equals("Total: $100.00")) {
System.out.println("Test Case 12 Passed: Order total is correct.");         } else {
            System.out.println("Test Case 12 Failed: Order total is not correct.");
        }
    // Close the browser         driver.quit();
   }
}
```

OUTPUT:

```
Test Case 1 Passed: Valid Username
Test Case 2 Passed: Valid Password
Test Case 3 Passed: User Login successful
Test Case 4 Passed: Products are displayed on the homepage. Test
Case 5 Passed: Search results are displayed for the product Test Case
9 Passed: Quantity of Product 1 is updated to 2.
Test Case 11 Passed: Shipping information is displayed correctly.
Test Case 12 Passed: Order total is correct.
```

RESULT

Thus, the automation of an e-commerce application using selenium has been executed successfully.

EX NO: 8 INTEGRATE TESTNG WITH ABOVE TEST AUTOMATION

DATE:

AIM:

To integrate TestNG for testing an e-commerce application with test automation

PROCEDURE:

- **Login Module:** ○ Test Case 1: Verify successful login with valid credentials. ○ Test Case 2: Verify error message for login with invalid credentials. ○ Test Case 3: Verify the functionality of the "Forgot Password" link.
- **Product Browsing:**
  - Test Case 4: Verify that products are displayed on the homepage. ○ Test Case 5: Verify the search functionality.
  - Test Case 6: Verify product filtering options.
- **Shopping Cart:** ○ Test Case 7: Verify the addition of products to the shopping cart. ○ Test Case 8: Verify the correctness of the items and quantities in the shopping cart.
  - Test Case 9: Verify the ability to update the quantity of items in the cart.
- **Checkout Process:**
  - Test Case 10: Verify the ability to proceed to checkout. ○ Test Case 11: Verify the display of correct shipping information.
  - Test Case 12: Verify the correctness of the order total.

CODE

```
import org.openqa.selenium.By; import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver; import
org.openqa.selenium.WebElement; import
org.openqa.selenium.chrome.ChromeDriver;

public class ECommerceTestSuite {
public static void main(String[] args) {
    // Set the path to your ChromeDriver executable
    System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
    // Initialize WebDriver
    WebDriver driver = new ChromeDriver();
    // Run test cases
runLoginModuleTests(driver);
runProductBrowsingTests(driver);
runShoppingCartTests(driver);      runCheckoutProcessTests(driver);
```

```
    // Close the browser        driver.quit();
  }
private static void runLoginModuleTests(WebDriver driver) {
      // Test Case 1: Verify successful login with valid credentials.
      driver.get("url_of_your_ecommerce_application");
      WebElement usernameInput = driver.findElement(By.id("username"));
      WebElement passwordInput = driver.findElement(By.id("password"));        WebElement
loginButton = driver.findElement(By.id("loginButton"));
usernameInput.sendKeys("valid_username");
passwordInput.sendKeys("valid_password");        loginButton.click();
// Add more login module test cases...
  }

  private static void runProductBrowsingTests(WebDriver driver) {
// Test Case 4: Verify that products are displayed on the homepage.
      WebElement homePageProducts = driver.findElement(By.id("homePageProducts"));        if
(homePageProducts.isDisplayed()) {
          System.out.println("Test Case 4 Passed: Products are displayed on the homepage.");
      } else {
          System.out.println("Test Case 4 Failed: Products are not displayed on the homepage.");
      }
      // Test Case 5: Verify the search functionality.
      WebElement      searchInput      =      driver.findElement(By.id("searchInput"));
searchInput.sendKeys("product_name");        searchInput.sendKeys(Keys.RETURN);
try {
          Thread.sleep(2000);
      } catch (InterruptedException e) {
e.printStackTrace();
      }

      WebElement searchResults = driver.findElement(By.id("searchResults"));
      if (searchResults.isDisplayed() && searchResults.getText().contains("product_name")) {
System.out.println("Test Case 5 Passed: Search results are displayed for the product.");        }
else {
          System.out.println("Test Case 5 Failed: Search results are not displayed for the product.");
}
  }
```

```
    private static void runShoppingCartTests(WebDriver driver) {
// Test Case 7: Verify the addition of products to the shopping cart.
        WebElement product = driver.findElement(By.xpath("//div[@class='product'][1]"));
product.click();
   // Test Case 8: Verify the correctness of the items and quantities in the shopping cart.
WebElement cartIcon = driver.findElement(By.id("cartIcon"));         cartIcon.click();
        WebElement cartItems = driver.findElement(By.id("cartItems"));         if
(cartItems.isDisplayed() && cartItems.getText().contains("Product 1")) {
            System.out.println("Test Case 8 Passed: Product 1 is in the shopping cart.");         }
else {
            System.out.println("Test Case 8 Failed: Product 1 is not in the shopping cart.");
        }
        // Test Case 9: Verify the ability to update the quantity of items in the cart.
WebElement    quantityInput    =    driver.findElement(By.id("quantityInput"));
quantityInput.clear();         quantityInput.sendKeys("2");
        WebElement updateButton = driver.findElement(By.id("updateButton"));
updateButton.click();

        // Wait for the cart to update (you might want to use WebDriverWait in a real-world scenario)
try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
e.printStackTrace();
        }
        // Check if the quantity is updated
        WebElement updatedQuantity = driver.findElement(By.id("updatedQuantity"));
if (updatedQuantity.getText().equals("2")) {
            System.out.println("Test Case 9 Passed: Quantity of Product 1 is updated to 2.");         }
else {
            System.out.println("Test Case 9 Failed: Quantity of Product 1 is not updated to 2.");
        }
    }

    private static void runCheckoutProcessTests(WebDriver driver) {         //
Test Case 10: Verify the ability to proceed to checkout.
```

```
        WebElement product = driver.findElement(By.xpath("//div[@class='product'][1]"));
product.click();
        WebElement cartIcon = driver.findElement(By.id("cartIcon"));         cartIcon.click();
         WebElement checkoutButton = driver.findElement(By.id("checkoutButton"));
checkoutButton.click();

        // Test Case 11: Verify the display of correct shipping information.         WebElement
shippingInfo = driver.findElement(By.id("shippingInfo"));          if (shippingInfo.isDisplayed()
&& shippingInfo.getText().contains("Shipping Address:")) {              System.out.println("Test
Case 11 Passed: Shipping information is displayed correctly.");         } else {
            System.out.println("Test Case 11 Failed: Shipping information is not displayed correctly.");
        }
        // Test Case 12: Verify the correctness of the order total.
        WebElement orderTotal = driver.findElement(By.id("orderTotal"));         if
(orderTotal.isDisplayed() && orderTotal.getText().equals("Total: $100.00")) {
System.out.println("Test Case 12 Passed: Order total is correct.");         } else {
            System.out.println("Test Case 12 Failed: Order total is not correct.");
        }
    }
}
```

OUTPUT:

```
Test Case 1 Passed: Valid Username
Test Case 2 Passed: Valid Password
Test Case 3 Passed: User Login successful
Test Case 4 Passed: Products are displayed on the homepage.
Test Case 5 Passed: Search results are displayed for the product Test Case
9 Passed: Quantity of Product 1 is updated to 2.
Test Case 11 Passed: Shipping information is displayed correctly.
Test Case 12 Passed: Order total is correct.
```

RESULT:

Thus, the automation of TestNG for testing an e-commerce application using selenium was executed successfully.

EX NO:9. a **BUILD A DATA-DRIVEN FRAMEWORK USING**

DATE: **SELENIUM AND TESTNG**

AIM

To Build a data-driven framework using selenium and TestNG.

PROCEDURE

- Create a CSV file (testdata.csv):
- Create a CSV file with columns "Username" and "Password" containing test data.
- Create a TestNG Test Class (LoginTest.java):
- Use TestNG annotations to define the test methods.
- Use @DataProvider to read test data from the CSV file.
- Configure TestNG XML File (testng.xml):
- Create a TestNG XML file to configure the execution of your test class.
- Run the Test:
- Execute the TestNG XML file to run your data-driven test.

Test Cases:

- Login with valid credentials:
- Login with invalid credentials: **CODE**

```
import org.openqa.selenium.By; import
org.openqa.selenium.WebDriver; import
org.openqa.selenium.WebElement; import
org.openqa.selenium.chrome.ChromeDriver; import
org.testng.Assert;  import
org.testng.annotations.DataProvider; import
org.testng.annotations.Test;

import java.io.BufferedReader; import
java.io.FileReader; import
java.io.IOException; import
java.util.ArrayList; import
java.util.List;

public class LoginTest {
 @Test(dataProvider = "loginData")
```

```
    public void testLoginWithValidCredentials(String username, String password) {
        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();
        // Open the login page
        driver.get("url_of_your_application_login_page");
        // Locate username and password fields
        WebElement usernameField = driver.findElement(By.id("username"));
        WebElement passwordField = driver.findElement(By.id("password"));
        WebElement loginButton = driver.findElement(By.id("loginButton"));
        // Enter valid username and password         usernameField.sendKeys(username);
passwordField.sendKeys(password);
        // Click on the login button         loginButton.click();
        // Verify successful login
        WebElement welcomeMessage = driver.findElement(By.id("welcomeMessage"));
        Assert.assertTrue(welcomeMessage.isDisplayed(), "Login failed for user: " + username);
        // Close the browser         driver.quit();
    }
    @Test(dataProvider = "loginData")      public void
testLoginWithInvalidCredentials(String username, String password) {
        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();
        // Open the login page
        driver.get("url_of_your_application_login_page");
        // Locate username and password fields
        WebElement usernameField = driver.findElement(By.id("username"));
        WebElement passwordField = driver.findElement(By.id("password"));
        WebElement loginButton = driver.findElement(By.id("loginButton"));
        // Enter invalid username and password         usernameField.sendKeys(username);
passwordField.sendKeys(password);

        // Click on the login button         loginButton.click();
        // Verify error message
        WebElement errorMessage = driver.findElement(By.id("errorMessage"));
Assert.assertTrue(errorMessage.isDisplayed(), "Error message not displayed for user: " + username);        //
Close the browser         driver.quit();
    }
    @DataProvider(name = "loginData")     public
Object[][] readData() {
```

```
        List<Object[]> testData = new ArrayList<>();         String
csvFile = "path/to/testdata.csv";
        try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {
            String line;
            while ((line = br.readLine()) != null) {
String[] data = line.split(",");                testData.add(data);
            }
        } catch (IOException e) {
e.printStackTrace();
        }
        Object[][] result = new Object[testData.size()][2];         for
(int i = 0; i < testData.size(); i++) {              result[i]
= testData.get(i);
}         return
result;
    }
}
```

Configure TestNG XML File (testng.xml):

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="TestSuite">
    <test name="DataDrivenTest">
        <classes>
            <class name="LoginTest"/>
        </classes>
    </test>
</suite>
```

OUTPUT

Username : John

Password : 123

Login Successful

RESULT:

Thus the data driven framework using selenium and TestNG was executed successfully.

EX NO:9. b BUILD PAGE OBJECT MODEL USING SELENIUM AND TESTNG

DATE:

AIM

To build a page object model using Selenium and TestNG.

PROCEDURE

- Create Page Object Classes:
- Create a class for each web page. Include web elements and methods representing the actions on that page.
- Create a Test Class using TestNG:
- Use TestNG annotations to define the test methods.
- Instantiate the Page Object classes and use them in your test methods.
- Run the Test:
- Execute the TestNG XML file or run the test class directly to run your tests.

CODE

```
import org.openqa.selenium.By; import
org.openqa.selenium.WebDriver; import org.openqa.selenium.WebElement;

public class LoginPage {  private final
WebDriver driver;     public
LoginPage(WebDriver driver) {         this.driver
= driver;
   }

   public void openLoginPage(String url) {
driver.get(url);
   }

   public void enterUsername(String username) {
      WebElement usernameField = driver.findElement(By.id("username"));
usernameField.sendKeys(username);
   }

   public void enterPassword(String password) {
      WebElement passwordField = driver.findElement(By.id("password"));
passwordField.sendKeys(password);
   }
```

```
    public void clickLoginButton() {
        WebElement loginButton = driver.findElement(By.id("loginButton"));         loginButton.click();
    }

    public boolean isErrorMessageDisplayed() {
        WebElement errorMessage = driver.findElement(By.id("errorMessage"));         return
errorMessage.isDisplayed();
    }
}
```

CODE TO CREATE A TEST CLASS

```
import org.openqa.selenium.WebDriver; import
org.openqa.selenium.chrome.ChromeDriver; import org.testng.Assert;
import org.testng.annotations.AfterMethod; import
org.testng.annotations.BeforeMethod; import org.testng.annotations.Test;

public class LoginTest {

    private WebDriver driver;
private LoginPage loginPage;

    @BeforeMethod     public
void setUp() {
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");         driver
= new ChromeDriver();         loginPage = new LoginPage(driver);
    }

    @Test      public void testLoginWithValidCredentials() {
loginPage.openLoginPage("url_of_your_application_login_page");
loginPage.enterUsername("valid_username");         loginPage.enterPassword("valid_password");
loginPage.clickLoginButton();

        // Add assertion based on your application behavior
    }

    @Test      public void
testLoginWithInvalidCredentials() {
```

```
        loginPage.openLoginPage("url_of_your_application_login_page");
loginPage.enterUsername("invalid_username");
loginPage.enterPassword("invalid_password");         loginPage.clickLoginButton();

        Assert.assertTrue(loginPage.isErrorMessageDisplayed(), "Error message not displayed for invalid
login");
    }

    @AfterMethod     public void
tearDown() {         driver.quit();
    }
}
```

OUTPUT :

User Login Successful

RESULT

Thus, the page object model using selenium and TestNG has been built successfully.

EX NO:9.c **BUILD A BDD FRAMEWORK USING SELENIUM,**

DATE: **TESTNG AND CUCUMBER**

AIM

To build a BDD framework using selenium,testng and cucumber.

PROCEDURE:

1. Create a Maven Project:
   - Use your IDE (Eclipse, IntelliJ, etc.) to create a new Maven project.
2. Add Dependencies:
   - Add dependencies for Selenium, TestNG, and Cucumber in the `pom.xml` file.
3. Create Page Object Classes:
   - Create a `pages` package and implement the `LoginPage` class for the login page.
4. Create Feature Files:
   - Create a `resources` directory and a `features` sub-directory.
   - Write Gherkin-style scenarios in a `login.feature` file.
5. Create Step Definition Classes:
   - Create a `step_definitions` directory.
   - Implement the `StepDefinitions` class with step definition methods.
6. Set Up TestNG XML File:
   - Create a TestNG XML file (e.g., `testng.xml`) and configure the Cucumber runner class.
7. Write Selenium Code in Step Definitions:
   - Use Selenium code in the `StepDefinitions` class to interact with web elements.
   - Utilize methods from the `LoginPage` class.
8. Run the Tests:
   - Run the TestNG XML file to execute the Cucumber tests.
9. Adjust Paths and Locators:
   - Adapt paths, element locators, and assertions based on your application's structure.
10. Enhance the Framework:
    - Add features like reporting, parallel execution, and additional steps.
    - Install Cucumber plugins for your IDE to aid in feature file writing.

CODE

Dependencies (Maven):

<!-- Selenium -->

<dependency>

```
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.141.59</version>
</dependency>

<!-- TestNG -->
<dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>7.3.0</version>
    <scope>test</scope>
</dependency>

<!-- Cucumber -->
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>7.2.10</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-testng</artifactId>
    <version>7.2.10</version>
    <scope>test</scope>
</dependency>
```

LoginPage.java (Page Object Class):

```
package pages; import
org.openqa.selenium.By; import
org.openqa.selenium.WebDriver; import
org.openqa.selenium.WebElement;
public class LoginPage {    private final
WebDriver driver;

    public LoginPage(WebDriver driver) {        this.driver
= driver;
    }
    public void openLoginPage(String url) {        driver.get(url);
```

```
    }
    public void enterUsername(String username) {
        WebElement usernameField = driver.findElement(By.id("username"));
usernameField.sendKeys(username);
    }
    public void enterPassword(String password) {
        WebElement passwordField = driver.findElement(By.id("password"));
passwordField.sendKeys(password);
    }

    public void clickLoginButton() {
        WebElement loginButton = driver.findElement(By.id("loginButton"));         loginButton.click();
    }

    public boolean isErrorMessageDisplayed() {
        WebElement errorMessage = driver.findElement(By.id("errorMessage"));         return
errorMessage.isDisplayed();
    }
}
```

login.feature (Cucumber Feature File):

```
Feature: Login functionality

  Scenario: Login with valid credentials
    Given User is on the login page
    When User enters valid username and password
    And User clicks on the login button
    Then User should be logged in successfully

  Scenario: Login with invalid credentials
    Given User is on the login page
    When User enters invalid username and password
    And User clicks on the login button
    Then User should see an error message
```

StepDefinitions.java (Cucumber Step Definitions):

```
package step_definitions;

import io.cucumber.java.en.Given; import
io.cucumber.java.en.When; import io.cucumber.java.en.Then;
```

```
import org.openqa.selenium.WebDriver; import
org.openqa.selenium.chrome.ChromeDriver; import
org.testng.Assert; import pages.LoginPage;

public class StepDefinitions {
private WebDriver driver;     private
LoginPage loginPage;

   @Given("User is on the login page")     public
void user_is_on_the_login_page() {
       System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
driver = new ChromeDriver();              loginPage = new LoginPage(driver);
loginPage.openLoginPage("url_of_your_application_login_page");
   }

   @When("User enters valid username and password")     public
void user_enters_valid_username_and_password() {
loginPage.enterUsername("valid_username");
loginPage.enterPassword("valid_password");
loginPage.clickLoginButton();
   }

   @When("User enters invalid username and password")     public
void user_enters_invalid_username_and_password() {
loginPage.enterUsername("invalid_username");
loginPage.enterPassword("invalid_password");
loginPage.clickLoginButton();
   }

   @Then("User should be logged in successfully")     public
void user_should_be_logged_in_successfully() {
       // Add assertions based on your application behavior
       // For example, assert that a welcome message is displayed
   }

   @Then("User should see an error message")     public
void user_should_see_an_error_message() {
```

```
    Assert.assertTrue(loginPage.isErrorMessageDisplayed(), "Error message not displayed for invalid
login");
  }

  @After    public void
tearDown() {
driver.quit();
  }
}
```

RUNNING THE TESTS

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="TestSuite">
  <test name="CucumberTests">
    <classes>
      <class name="io.cucumber.testng.CucumberRunner"/>
    </classes>
  </test>
</suite>
```

RESULT

Thus, the BDD framework with Selenium,TestNG and Cucumber was executed successfully