Amazon Automation Testing Project Report

**1.** **Introduction**

This report details the automation testing activities carried out on the Amazon.in website using Selenium WebDriver and Java. The objective was to simulate and verify critical e-commerce functionalities such as login, product search, filtering, cart handling, and payment simulation.

**2. Background**

This project was undertaken to gain real-time experience in functional and UI automation testing. By choosing a high-traffic, dynamic e-commerce platform like Amazon, the goal was to work with complex scenarios such as dynamic elements, multiple windows, filtering logic, and cart total validations.

**3. Learning Objectives**

Implement end-to-end test automation using Selenium WebDriver and TestNG.

Practice advanced XPath strategies for locating elements.

Enforce input validation using regex.

Use Page Object Model (POM) to structure test code efficiently.

Handle cart states, window switching, and page validations dynamically.

Introduce screenshot capture for assertion failures.

4. **Activities and Tasks**

* Login Validation

Objective: Verify username validation rules.

Logic Implemented:

String username = "TestUser1";

boolean isValid = username.matches("^[a-zA-Z0-9]{1,10}$") && !username.matches(".\*[ACGIacgi].\*");

Assert.assertTrue(isValid, "Username must be alphanumeric, max 10 chars, and not contain A, C, G, I");

* Product Search and Filtering

Objective: Search a product, ignore products starting with A–D and avoid Electronics category.

Code Example:

List<WebElement> titles = driver.findElements(By.xpath("//span[@class='a-size-medium a-color-base a-text-normal']"));

for (WebElement title : titles) {

String productTitle = title.getText();

if (!productTitle.startsWith("A") && !productTitle.startsWith("B") &&

!productTitle.startsWith("C") && !productTitle.startsWith("D")) {

title.click();

break;

}

}

* Product Page Verification

Verifications:

Product image is displayed.

Price is available.

Ratings and review count are visible.

XPath Example:

WebElement price = driver.findElement(By.xpath("//span[@class='a-price']//span[@class='a-offscreen']"));

Assert.assertTrue(price.isDisplayed(), "Price not displayed");

* Add to Cart and Cart Validation

Add to Cart:

WebElement addToCartBtn = driver.findElement(By.id("add-to-cart-button"));

addToCartBtn.click();

Cart Total Validation (> ₹2000):

String priceText = driver.findElement(By.xpath("//span[@id='sc-subtotal-amount-activecart']")).getText();

double totalPrice = Double.parseDouble(priceText.replace("₹", "").replace(",", "").trim());

Assert.assertTrue(totalPrice > 2000, "Cart value should be greater than ₹2000");

* Clear Cart Before Test Execution

Utility Method:

List<WebElement> deleteButtons = driver.findElements(By.xpath("//input[@value='Delete']"));

for (WebElement btn : deleteButtons) {

btn.click();

}

* Payment Flow Simulation

Payment Page Validations:

WebElement totalAmount = driver.findElement(By.xpath("//span[@id='subtotals-marketplace-table']//span[@class='a-size-medium a-color-base sc-price sc-white-space-nowrap']"));

double amount = Double.parseDouble(totalAmount.getText().replace("₹", "").replace(",", "").trim());

Assert.assertTrue(amount > 500, "Payment amount should be greater than ₹500");

* Screenshot Capture on Failure

Reusable Method:

public void captureScreenshot(String testName) {

TakesScreenshot ts = (TakesScreenshot) driver;

File src = ts.getScreenshotAs(OutputType.FILE);

File dest = new File("./screenshots/" + testName + ".png");

FileUtils.copyFile(src, dest);

}

**5. Skills and Competencies Developed**

* Strong command over Selenium WebDriver and TestNG.
* Expertise in writing complex XPath expressions.
* Developed a structured Page Object Model (POM) framework.
* Proficient in applying validation logic using Java.
* Improved debugging skills with automatic screenshot capture.
* Experienced in dynamic product filtering, window handling, and cart state management.

**6. Feedback and Evidence**

All key test flows were successfully implemented and verified.

Test execution reports were generated via TestNG.

Screenshots were automatically saved for all failed tests.

Proper assertion messages ensured traceability and faster debugging.

Complex scenarios (e.g., multi-window handling and data validations) were resolved using custom utility functions.

**7. Challenges and Solutions**

| Challenge | Solution |
| --- | --- |
| Handling popups and new windows | Used driver.getWindowHandles() and switched using driver.switchTo().window() logic. |
| Username validation constraints | Combined multiple regex patterns to exclude specific characters and limit string length. |
| Avoiding disallowed products | Used Java string checks and startsWith() method during iteration of search results. |
| Cart cleanup | Automated deletion of cart items using button locator strategy to maintain test state. |

**8. Outcomes and Impact**

Developed an efficient, end-to-end automation suite for Amazon’s core user journey.

Gained hands-on experience in solving real-world e-commerce testing problems.

Improved the ability to write modular and scalable test code.

Enhanced confidence in automation scripting, validations, and result reporting.

Built reusable utilities for common testing actions such as element checks and screenshot capture.

**9. Conclusion**

This Amazon automation project provided deep insights into automation challenges and taught practical strategies to overcome them. Through structured design, accurate validations, and efficient error handling, a production-like automation framework was developed. The skills acquired during this project form a strong foundation for professional roles in software testing.