SE333 (6%) - Assignment 5 - UI Testing

Background.

In this assignment, you will learn how to use user interface testing tools, specifically **Playwright**. Specifically, you will learn how to use a UI testing tool, create a UI test report, and create UI tests.

Definition

UI testing is

- Used to verify the reliability of User Interfaces and ensure the UI meets specifications.
- Done through test cases
- Possible to do manually, but much more scalable through automation.

Goals

- To determine if UI behaves according to specifications and in consistent ways

Mechanism

Graphical user interfaces can present problems that will not be readily apparent through regular source code testing such as unit tests. Some of these problems might be subjective and must be tested by human beings to determine their severity. However, it is possible to automate certain aspects of UI testing. For example, it is possible to determine if all the desired and expected elements are present after a web page is done loading.

Tools

Various tools exist to test user interfaces. These tools are usually highly dependent on the kind of UI being tested. Websites can be tested with tools such as Selenium. Tools such as Telerik can be used to test HTML AJAX and Silverlight applications, Cucumber can be used to test system behavior in a variety of languages, and can be used to run other UI tests. In mobile applications, we can use tools such as Espresso to automate UI testing.

Playwright - Features

Playwright is a framework for testing web applications. It is composed of multiple components such as Playwright **codeGen** and **recorder**, which is a browser extension that can record and replay user interactions within a website. **Playwright** framework also contains a WebDriver, which allows programmers to write tests in a variety of programming languages using the **Playwright** API. These tests can then be run using most modern browsers to determine if the UI under test is functioning correctly.

Environment and Requirements

Please make sure that you have Maven and Java installed on your computer.

- OS: macOS/Windows 10
- Java: version 1.8+
- Maven: version 3.8.5+

Note: Only the above environment is tested by us, we are not sure if other versions work well.

Tasks: Simple UI Testing on DePaul Website

Part A *(Adding playwright into your pom.xml)*

- This assignment does not have a starter code. Your job is to create a new maven project and add playwright as dependency in your pom.xml

```
<dependency>
  <groupId>com.microsoft.playwright</groupId>
  <artifactId>playwright</artifactId>
  <version>LATEST</version> // replace with the latest version
</dependency>
```

Part B *(Writing Playwright tests automations)*

The test case you will automate will be a purchase pathway scenario for a specific product in the DePaul University website bookstore. The test case should be performed as follows:

Startup

- Start playwright **codegen** and **record** by typing cmd below:

  ```
  mvn exec:java -e -D exec.mainClass=com.microsoft.playwright.CLI -D
  exec.args="codegen"
  ```

- Go to https://depaul.bncollege.com/ link on the browser.
- Then start recording.

Write Test Cases

0. First create a new package called `playwrightTraditional`, then write following test cases below.
1. **TestCase Bookstore**
   - Enter “earbuds” on the search box in the upper right and press the return key.
   - Click on the “Brand” filter to expand it, then select “JBL”.
   - Click on the “Color” filter to expand it, then select “Black”.
   - Click on the “Price” filter to expand it, then select “Over $50”.
   - Click on the “JBL Quantum True Wireless Noise Cancelling Gaming…” item link.
   - **AssertThat** product name, SKU number, the price, and the product description.
   - Add 1 to the Cart.
   - **AssertThat** “1 Items" in cart (upper-right of page)
     - [You may need to pause here and wait for the cart icon to reflect the change].
   - Click “Cart” (click icon in upper right of page).
2. **TestCase Your Shopping Cart Page:**
   - **AssertThat** you are at cart: “Your Shopping Cart”

- **AssertThat** the product name (JBL Quantum True Wireless Noise Cancelling Gaming Earbuds- Black), quantity (1), and price ($149.98).
- Select “FAST In-Store Pickup”.
- **AssertThat** sidebar subtotal (149.98), handling (2.00), taxes (value is “TBD”), and estimated total (151.98).
- Enter promo code TEST and click APPLY (code doesn’t exist so it should fail).
- **AssertThat** promo code reject message is displayed.
- Click “PROCEED TO CHECKOUT”.

3. **TestCase Create Account Page**
    - **AssertThat** “Create Account” label is present.
    - Select “Proceed as Guest”
4. **TestCase Contact Information Page**
    - **AssertThat** you are at Contact Information page
    - Enter a first name, a last name, an email address, a phone number
    - **AssertThat** sidebar subtotal (149.98), handling (2.00), taxes (value is “TBD”), and estimated total (151.98).
    - Click CONTINUE
5. **TestCase Pickup Information**
    - **AssertThat** Contact Information: name, email, and phone are correct.
    - **AssertThat** Pick Up location (DePaul University Loop Campus & SAIC)
    - **AssertThat** selected Pickup Person (“I’ll pick them up”)
    - **AssertThat** Sidebar order subtotal (149.98), handling (2.00), taxes (value is “TBD”), and estimated total (151.98).
    - **AssertThat** pickup item and price are correct.
    - Click CONTINUE.
6. **TestCase Payment Information**

- **AssertThat** Sidebar order subtotal (149.98), handling (2.00), taxes (15.58), and total (167.56).
- **AssertThat** pickup item and price are correct.
- Click "< BACK TO CART"> (upper-right of page).

7. **TestCase Your Shopping Cart**
    - Delete product from cart.
    - **AssertThat** your cart is empty.
    - Close window.

After you have developed all your test. Record the video of the entire Playwright test run start to finish. To record the video, use the code below:

```
BrowserContext context = browser.newContext(new Browser.NewContextOptions()
        .setRecordVideoDir(Paths.get("videos/"))
        .setRecordVideoSize(1280, 720));
```

To execute the tests, you need to run `mvn test` to ensure that your UI test work as intended and passes all the tests and builds.

> Note: You MUST clear the browser cache between script runs or the cart quantities and the "tax" value will not be listed as TBD as it should do until the user's account information.

Part 3 — Use Github Actions

> Upload your code to GitHub and set up GitHub Actions workflows so that every time code is pushed, your user interface tests are automatically executed. Please refer to the link.

Part 4 — MCP for Playwright LLM Agent

> In traditional UI testing, you write test scripts manually (e.g., in Java using Playwright) and define assertions using code. In this final part, you will use an AI agent powered by Playwright MCP to perform the same UI test cases.

0. First create a new package called `playwrightLLM`, then use Playwright MCP to generate tests.

1. **Install Playwright MCP in VS Code**
    - Follow the instructions here: **https://github.com/microsoft/playwright-mcp**
    - This extension lets AI agents (like ChatGPT connected through MCP) interact with a live browser to generate and run Playwright tests.
2. Generate test using natural language prompts:
    - Use AI to describe your same workflow, for example:

      > *"Test search for earbuds, filter by color, add to cart, and verify the cart shows 1 item."*

    - Ask the agent to **generate Playwright tests** in java and runnable in Junit.
    - Add the generated code into the `playwrightLLM` folder of your codebase.
    - Record **test execution videos** like in Part B.
3. Write a several paragraph reflection comparing:
    - **Manual UI testing**: You wrote it in Java + Playwright
    - **AI-assisted UI testing**: You prompted the MCP agent to generate Playwright tests. You can compare aspects like:
        - Ease of writing and running tests
        - Accuracy and reliability of generated tests
        - Maintenance effort
        - Limitations or issues you encountered

Delivery

- Submit your source code
    - Your test should have two packages to compare the different test generation approaches.
        - `test.java.playwrightTraditional`
        - `test.java.playwrightLLM`
- In the readme.md

  - Please reference GitHub repository link
  - The GitHub action should execute all the tests must pass.
  - Reflection that compares the two approaches.
  - If it does not compile, it is an instant zero.