**Playwright Lab: Testing a Bookstore Web Application**

Playwright is a powerful framework for automating browsers, allowing developers to simulate user interactions with web applications. It supports multiple browsers, making it ideal for cross-browser testing. Playwright is primarily used for end-to-end testing, which tests the flow of an application from start to finish. It ensures that the integrated components of an application function as expected. This lab will guide you through the process of using Playwright to write and execute tests on a Flask-based bookstore web application.

-----------------------------------------------------------------------------

1. **Getting Started with Playwright**

**Install Python**:

* + Ensure Python (version 3.7 or higher) is installed on your system. You can download it from [python.org](https://www.python.org/downloads/).

**Install Playwright**:

* + Open a command line interface and install Playwright using pip:

pip install playwright

* + After installation, run the following command to install the necessary browsers:

playwright install

-----------------------------------------------------------------------------

1. **Set Up Your Testing Environment**:

* Create a new folder on your computer for this project.
* Download the provided main.py (the Flask web application) and the incomplete playwright\_test\_suite.py file. Place these in this folder.

**Run the Flask Application**:

* Navigate to your project folder in the command line and run:

python main.py

* The application should now be running at <http://127.0.0.1:5000/>.

-----------------------------------------------------------------------------

1. Learning Resources:

**Official Playwright Documentation**:

* + [Playwright Docs](https://playwright.dev/docs/intro): Comprehensive guide and reference to Playwright features and API.

**Online Tutorials and Courses**:

* + [Automate Testing with Playwright](https://www.udemy.com/course/automated-software-testing-with-playwright/): A course on Udemy covering basics to advanced concepts.
  + [Playwright Tutorial Series on YouTube](https://www.youtube.com/playlist?list=PL4cUxeGkcC9jx2TTZk3IGWKSbtugYdrlu): A series of video tutorials for visual learners.

**Community Forums and Support**:

* + [Playwright GitHub](https://github.com/microsoft/playwright): For contributions, issues, and discussions related to Playwright.
  + [Stack Overflow](https://stackoverflow.com/questions/tagged/playwright): Community Q&A on Playwright-related topics.

-----------------------------------------------------------------------------

1. **Writing Playwright Tests**

**Open Your Test File**:

* + In your project folder, open the file named playwright\_test\_suite.py.
  + This file will contain all your Playwright tests.

**Basic Playwright Test Structure**:

* + A basic Playwright test structure (included) looks like this:

from playwright.sync\_api import sync\_playwright

def run(playwright):

browser = playwright.chromium.launch(headless=False)

context = browser.new\_context()

# Define your tests here

context.close()

browser.close()

with sync\_playwright() as playwright:

run(playwright)

-----------------------------------------------------------------------------

1. **Test Examples:**

**Test 1 - Loading the Webpage and Checking the Title**:

This test navigates to the main page and checks if the title is as expected.

page = context.new\_page()

page.goto("http://127.0.0.1:5000/")

assert "County Bookstore" in page.title(), "Page title does not match"

**Test 2 - Adding a New Book**:

This test simulates adding a new book by interacting with the web elements.

page.click("text=Add Book") # Click on the 'Add Book' button

page.fill("input[name='title']", "New Book Title") # Fill the title

page.fill("input[name='author']", "Author Name") # Fill the author

page.click("text=Submit") # Submit the new book

assert "New Book Title" in page.content() # Verification logic

-----------------------------------------------------------------------------

1. **Incomplete Test Scenarios (Exercises YOU need to Complete):**

**Test 3 (Incomplete) - Search Functionality**:

* + - Write a test that searches for a book and verifies the search results.
    - Instruction: Fill in the search input, click the search button, and assert that the expected result is present in the page content.

**Test 4 (Incomplete) - Error Handling**:

* + - Write a test for handling an error, like submitting an empty form.
    - Instruction: Try submitting the 'Add Book' form without filling in any details and assert that the appropriate error message is displayed.

-----------------------------------------------------------------------------

1. **Expected Results from Testing**

* Test 1: For the "Loading the Webpage" test, the page title should match "County Bookstore".
* Test 2: In the "Adding a New Book" test, the new book title should be present in the page content after submission.
* Test 3: Expected results will depend on the specific implementation but should generally include successful execution of the search.
* Test 4: Expected results will depend on the specific implementation but should generally include proper error handling.

-----------------------------------------------------------------------------

1. **Running the Tests**

Execute your tests using the command:

python playwright\_test\_suite.py

Observe the results to ensure all tests are functioning as expected.

-----------------------------------------------------------------------------

1. **Conclusion**

This lab provides a comprehensive introduction to automated web testing using Playwright. Through a mix of guided examples and interactive exercises, you will gain practical experience in testing web applications. This hands-on approach is designed to enhance your skills in both Playwright and general web testing methodologies.