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

Selenium is an umbrella of tools and libraries used for web based automated testing. Selenium allows the user to emulate interacting with websites and browsers as if a real user was doing so, letting you test your websites across many different browsers easily and efficiently. This lab will guide you through the process of using Selenium to write and execute tests on a Flask-based bookstore web application.

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

1. **Getting Python, Selenium and Required Packages:**

**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/).
  + You can ensure Python is installed and check its version by opening a command line and running:

python --version

**Install Required Packages**:

* + Open a command line interface inside of the ‘WebApp’ folder
  + Run the requirements.txt file to install all required packages

pip install -r requirements.txt

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

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

**Setting Up the Chrome Web Driver**

* + There are 2 ways to get a working Chrome web driver for this lab. The first and simpler option is to simply install [Google Chrome](https://www.google.com/chrome/) at its default location.
  + Alternatively, navigate to [Google Chrome Labs](https://googlechromelabs.github.io/chrome-for-testing/) to download the latest **stable** release of Chrome web browser to use as the Chrome Web Driver. To setup the Chrome Web Driver via this method, follow the instructions [here](https://www.browserstack.com/guide/run-selenium-tests-using-selenium-chromedriver) depending on your operating system.

**Setting Up the Python Virtual Environment**

* + Navigate to the ‘WebApp’ folder and open a command line. Initiate your Python virtual environment by running:

python -m venv env

**Run the Flask Application**:

* Navigate to your ‘WebApp’ folder in the command line and run:

flask --debug --app main run

* If you receive the error ‘flask is not a recognized command’, alternatively run:

**python -m** flask --debug --app main run

* The application should now be running at <http://127.0.0.1:5000/> and will respond to changes you make in the code when you refresh the page. The flask application will need to be running for the lab to interact with it.

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

1. Learning Resources:

**Official Selenium Documentation**:

* + [Selenium Documentation](https://www.selenium.dev/documentation/): The official docs for Selenium and how to use it.

**Community Forums and Support**:

* + [Selenium Public Support Group](https://groups.google.com/g/selenium-users?pli=1): A public Google group for Selenium users to seek help from one another
  + [Stack Overflow](https://stackoverflow.com/questions/tagged/selenium-webdriver): Community Q&A on Selenium-related topics.

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

1. **Writing Selenium Tests**

**Open Your Test File**:

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

**Basic Selenium Test Structure**:

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

# Import driver

from selenium import webdriver

# Start web driver for Chrome

driver = webdriver.Chrome()

# Navigate to the Flask web application's URL.

driver.get("http://127.0.0.1:5000/")

# Define the test function

def test\_loading\_webpage():

    # Test to ensure the main page loads correctly.

    print("Starting test: test\_loading\_webpage")

    try:

# Assert result

        assert "County Bookstore" in driver.title

        print("Passed: test\_loading\_webpage")

    except AssertionError:

        print("Failed: test\_loading\_webpage. Expected 'County Bookstore' in page title.")

# Call test function

test\_loading\_webpage()

# Close the driver when done

driver.close()

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

1. **Test Examples:**

**Test 1 – Validate that the new book “Pages” and “Release Year” fields only accept integers**:

This test enters a single letter into the “Pages” and “Release Year” options in the new book field and asserts that it failed to enter them.

print("Starting test: test\_001\_nonint\_input")

    try:

        # Show the new book box

        new\_book\_button = driver.find\_element(By.ID, "newBookToggle")

        new\_book\_button.click()

        # Wait half a second for the box to appear

        driver.implicitly\_wait(0.5)

        # Grab the inputs

        pages\_input = driver.find\_element(By.ID, "pages")

        release\_year\_input = driver.find\_element(By.ID, "releaseYear")

        # Attempt to put a character in them

        pages\_input.send\_keys("a")

        release\_year\_input.send\_keys("a")

        # Get the current contents of the input

        pages\_input\_content = driver.find\_element(By.ID, "pages").get\_attribute("value")

        pages\_inrelease\_year\_input\_content = driver.find\_element(By.ID, "releaseYear").get\_attribute("value")

        # Assert results

        assert pages\_input\_content == ""

        assert pages\_inrelease\_year\_input\_content == ""

        print("Passed: test\_001\_nonint\_input\n")

    except Exception as e:

        print("Failed: test\_001\_nonint\_input. Error: " + str(e) + '\n')

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

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

# Send inputs

        driver.find\_element(By.ID, "newTitle").send\_keys("Fake Title")

        driver.find\_element(By.ID, "newAuthor").send\_keys("Fake Author")

        driver.find\_element(By.ID, "newGenres").send\_keys("Fiction")

        driver.find\_element(By.ID, "pages").send\_keys("100")

        driver.find\_element(By.ID, "releaseYear").send\_keys("2023")

        # Submit new book

        driver.find\_element(By.ID, "submitNewBook").click()

        # Check to see if the book title is present in the page

        # 2 seconds of wait added to allow for the page to refresh

        driver.implicitly\_wait(2)

        assert new\_title in driver.page\_source

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

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 “Non-int input” test, the page and release year boxes in the ‘New Book’ section won’t be able to accept a non-numeric input.
* 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 either assert the present of a displayed error or other handling.

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

1. **Running the Tests**

Execute your tests using the command:

python selenium\_test\_suite.py

Observe the results to ensure all tests are functioning as expected. Your command line should indicate “Passed: \_\_” or “Failed: \_\_” for each test.

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

1. **Conclusion**

This lab should provide a good insight into the workings of Selenium Web Driver, how to automate testing, and the beneficial impact that something like automated web-testing can provide in a development environment. Through working with Selenium to implement these tests, we can assure that functionality does not break in the future when further additions are made to our program by simply running the tests again.