**How to use Playwright**

First thing, make a folder you want to start. Then open that folder in VSCode. Open up your terminal and type git clone <https://github.com/CSC-256-Group-Project-9/Group-9-Playwright-Lab.git>. If done right, you should see this. Then cd to the lab. A screenshot of a computer

Description automatically generated

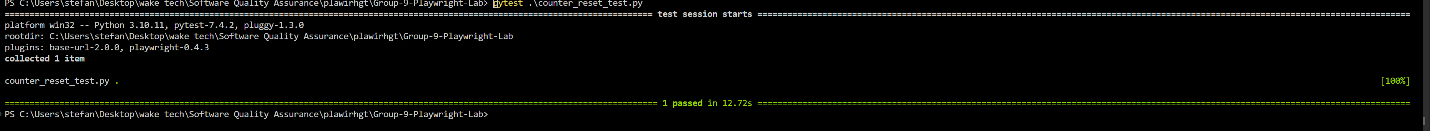
Then install Playwright, Playwright install Chromium, and Pytest. You can just enter these in your terminal one by one.

* pip install playwright
* playwright install chromium
* pip install pytest

If done right, you will be able to test all of the python files. We will be testing only one of the files so type the following command into the terminal.

* Pytest counter\_reset\_test.py

The test should pass here is a screenshot of what it should look like



Let's take a closer look at the code to see how it works. We will be using counter\_reset\_test.py as an example.

# Imports all the required packages

**import** re, pytest

**from** playwright.sync\_api **import** Page, expect

This section imports the necessary packages/modules for the script:

* **re**: Regular expression module for pattern matching.
* **pytest**: The pytest framework for testing.
* **Page**: Part of the Playwright library, representing a browser page.
* **expect** Helper function for asserting expectations on Playwright elements.

@pytest.fixture(*scope***=**"function", *autouse***=**True)

*def* test\_get\_page(page: *Page*):

    page.goto("https://group-9-webapp-official.vercel.app/testing")

    # Expect a title "to contain" a substring.

    expect(page).to\_have\_title(re.compile("Testing"))

* This is a pytest fixture, which is a function that sets up or provides resources for other tests. In this case, it sets up a Playwright page and navigates to a specific URL.
* expect(page).to\_have\_title(re.compile("Testing")): It asserts that the page title contains the substring "Testing".

*def* test\_validation(page: *Page*):

    # Checks if default value is zero

    expect(page.locator('#test2-output')).to\_have\_text('0')

    # Clicks the counter button

    page.locator('#test2-button').click()

    # Checks if output value changed (expected value is 1)

    expect(page.locator('#test2-output')).to\_have\_text('1')

    # Clicks the reset button in the test 3 box

    page.locator('#test3-button').click()

    # Checks if output value changed (expected value is 0)

    expect(page.locator('#test2-output')).to\_have\_text('0')

    # Clicks the button twice more

    page.locator('#test2-button').click()

    page.locator('#test2-button').click()

    # Checks the value before clicking the reset button

    expect(page.locator('#test2-output')).to\_have\_text('2')

    # Clicks the reset button in the test 3 box

    page.locator('#test3-button').click()

    # Checks if output value changed (expected value is 0)

    expect(page.locator('#test2-output')).to\_have\_text('0')

* This is the main test function, testing the functionality of a web page.
* expect(page.locator('#test2-output')).to\_have\_text('0'): Asserts that the element with the id 'test2-output' has text content '0'.
* page.locator('#test2-button').click(): Clicks the element with the id 'test2-button'.
* Subsequent lines check and perform various actions on the web page, using Playwright locators and assertions.

Conclusion: Overall, the script navigates to a webpage, performs interactions, and checks if the expected outcomes match the actual outcomes using Playwright and Pytest.