PLAYWRIGHT

Playwright is a powerful end-to-end testing library for web applications. End-to-end testing (E2E testing) is a software testing methodology that involves testing the entire application flow from start to finish to ensure that all components and systems work together as expected.

Playwright is an open-source Node.js library developed by Microsoft that enables you to automate browser interactions across multiple browsers, including Chromium, Firefox, and WebKit. With Playwright, you can write tests to simulate user interactions such as clicking buttons, filling forms, navigating pages, and validating content—all programmatically.

Features of Playwright:

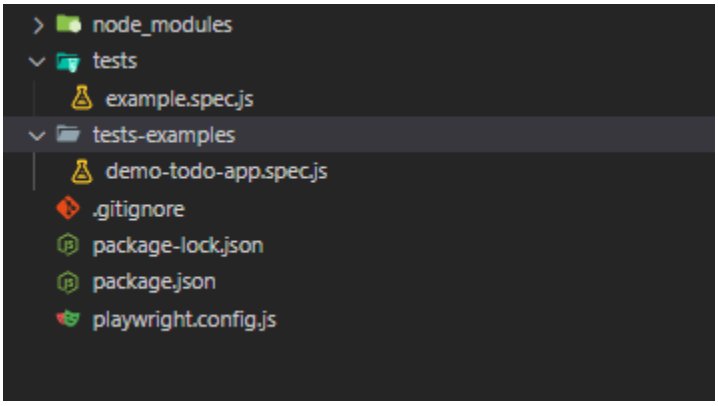
1. **Cross-Browser Support:** Playwright offers native support for automating interactions across popular browsers like Chromium, Firefox, and WebKit, enabling you to write tests that cover a wide range of browser environments, ensuring broad coverage for your web application. Additionally, Playwright provides flexibility by allowing you to configure it to test on other browsers such as Microsoft Edge, Safari, and more directly from the **playwright.config.js** file. This allows you to write tests that are compatible with multiple browser environments, ensuring broad coverage for your web application.
2. **Headless and Headful Mode:** Playwright supports both headless and headful modes, allowing you to run tests in a headless environment for faster execution or in a visible browser window for debugging purposes.
3. **Comprehensive API:** Playwright offers a comprehensive API with a wide range of functions for interacting with web pages. From navigating to URLs and clicking elements to filling forms. Playwright's API provides everything you need to simulate user behavior in your tests.
4. **Wait Mechanisms:** Playwright includes powerful wait mechanisms to ensure reliable and robust tests. You can wait for elements to appear, disappear, or meet specific conditions before proceeding with test execution. This ensures that your tests are resilient to dynamic content and asynchronous operations.
5. **Device Emulation:** With Playwright, you can emulate various device types and screen sizes to test the responsiveness of your web application. Whether you're testing on desktop, tablet, or mobile devices, Playwright's device emulation capabilities ensure accurate simulation of different user environments.

Getting Started with Playwright:

- For working with Playwright , first you need to install Node.js
- Go to https://nodejs.org/en/download
- Download the LTS(Long-term supported) version of Node.js
- In most cases, the Node package manager(npm) also comes with Node.js if you download and install it from official website.
- If npm is not installed , then you can open command line in your system and run following command `` `npm install -g npm` ``.
- Then open your preferred IDE, or VS code.

Compiled By: Khawar Khan

- Open the terminal.
- And initialize a new playwright by running the command `npm init playwright@latest`.
- Then select your preferred language (Javascript or Typescript) and answer other question according to your requirements.
- Answer “Y” or true for this requirement ` Install Playwright browsers (can be done manually via 'npx playwright install')? `.
- The initial folder structure will look like this:



- Now you can start writing test cases in the `tests` folder.

Writing Your First Test case in Playwright:

- Create a file in `tests` folder with suffix `.spec.js` which is used to identify that this file contains test specifications and `.js` represents a Javascript script.

- Import `test` and `expect` from `@playwright/test`, you can import it using one of the following ways:

```
1  const {test, expect} = require('@playwright/test');
2  /////////   OR  ////////////
3  import {test, expect} from '@playwright/test';
```

- Write the test case using ‘test’ function which is used to define individual test cases or test suites.
- And expect is used to define assertions within test cases to verify expected outcomes.
- When defining a test case, provide two paramters, the test case title, and anonymous function.

```
6  test("First Test Case", async({page}) => {
7
8  })
9
10
```

Here the **`page`** parameter represents the Playwright **`Page`** object, which provides methods and properties for interacting with a web page in a test case.

- Then go to your website's url.

```
test("First Test Case", async({page}) => {
    await page.goto('http://localhost/web%20Applications%20with%20PHP/finalModule_Course1/login.
    php');
})
```

- Then you can interact with the page objects fill the form and try to submit.

```
6   test("First Test Case", async({page}) => {
7       await page.goto('http://localhost/web%20Applications%20with%20PHP/finalModule_Course1/login.
        php');
8     //set username
9     await page.fill('input[name="who"]', 'Jack');
10    //set password field, inalid password
11    await page.fill('input[name="pass"]', 'pass123'); //wrong password
12    // Now click the login button
13    await page.click('button[value="LogIn"]');
14    //wait for navigation
15    // await page.waitForSelector('text=Log In', {timeout: 5000});
16    await page.waitForURL('http://localhost/web%20Applications%20with%20PHP/finalModule_Course1/
      checkingLogin.php');
17  })
```

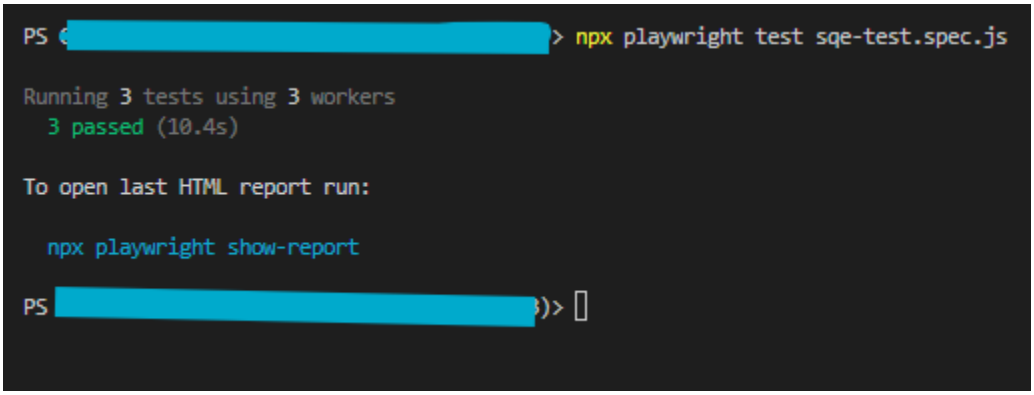
- Then you can insert some assertions using expect function to compare the actual response of the page with the expected response.

```
6   test("First Test Case", async({page}) => {
7       await page.goto('http://localhost/web%20Applications%20with%20PHP/finalModule_Course1/login.
        php');
8     //set username
9     await page.fill('input[name="who"]', 'Jack');
10    //set password field, inalid password
11    await page.fill('input[name="pass"]', 'pass123'); //wrong password
12    // Now click the login button
13    await page.click('button[value="LogIn"]');
14    //wait for navigation
15    // await page.waitForSelector('text=Log In', {timeout: 5000});
16    await page.waitForURL('http://localhost/web%20Applications%20with%20PHP/finalModule_Course1/
      checkingLogin.php');
17
18    //inserting assertions
19    await expect(page.locator('p[style="color: red;"]')).toHaveText(
20      "Incorrect Password."
21    );
22  })
```

Running Your First Test Case:

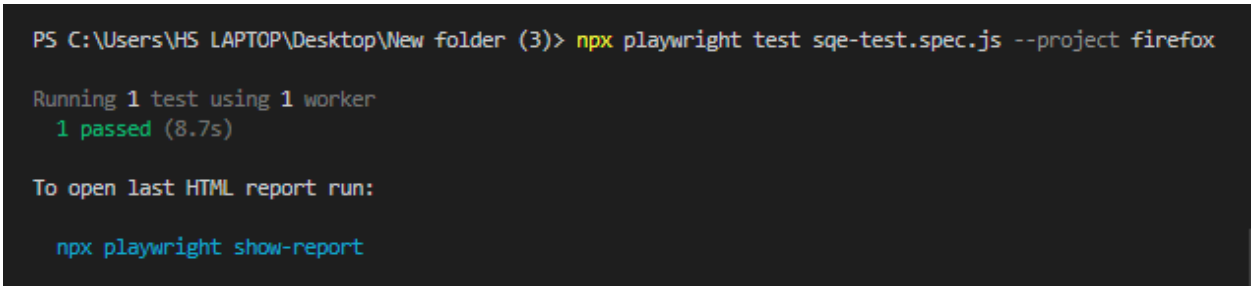
- Open Terminal.
- Run the following command to run the test case.

`npx playwright test  <your test file name>`.



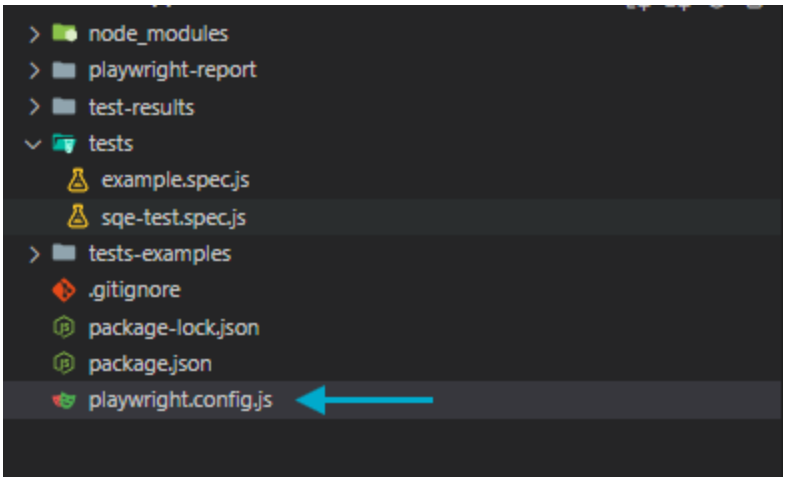
- By default, it is running the test case on Chrome, WebKit, and Firefox, that is why it is running three tests, it is running a single test on three browsers.
- If you want to run the test case on specific browser, then you can run the following command.
    - For Chrome: `npx playwright test <your test file name> --project chromium`
    - For Firefox: `npx playwright test <your test file name> --project firefox`

  And in the same way you can run tests on a specific browser, if you do not want to run tests on all browsers.



  You can see that now it is running the test only one time, on Firefox browser.

- If you want to see the real-time execution of your test cases and observe the interactions with the web page, then you add `--headed` at the last of you command. For example:
    - `npx playwright test <your test file name> --headed`
    - `npx playwright test <your test file name> --project chromium --headed`
- Now if you want to run your test cases on other browsers or devices , then go to playwright.config.js file in your root directory.



And locate projects in this file for configuring other devices and browsers.

Initially it look like this:

```
projects: [
  {
    name: 'chromium',
    use: { ...devices['Desktop Chrome'] },
  },

  {
    name: 'firefox',
    use: { ...devices['Desktop Firefox'] },
  },

  {
    name: 'webkit',
    use: { ...devices['Desktop Safari'] },
  },
```

You can add mobile viewports:

```
  {
    name: 'Mobile Chrome',
    use: { ...devices['Pixel 5'] },
  },
  {
    name: 'Mobile Safari',
    use: { ...devices['iPhone 12'] },
  },
```

And for adding other browsers such as Microsoft Edge.

```
{
  name: 'Microsoft Edge',
  use: { ...devices['Desktop Edge'], channel: 'msedge' },
},
```

Checking Report of Test Execution:

For seeing test execution report you can run command `npx playwright show-report`.

```
PS [redacted] npx playwright show-report

  Serving HTML report at http://localhost:9323. Press Ctrl+C to quit.
```

Now you can view the report on the provided URL.

BONUS TIP: For Locating different Elements on a Web Page

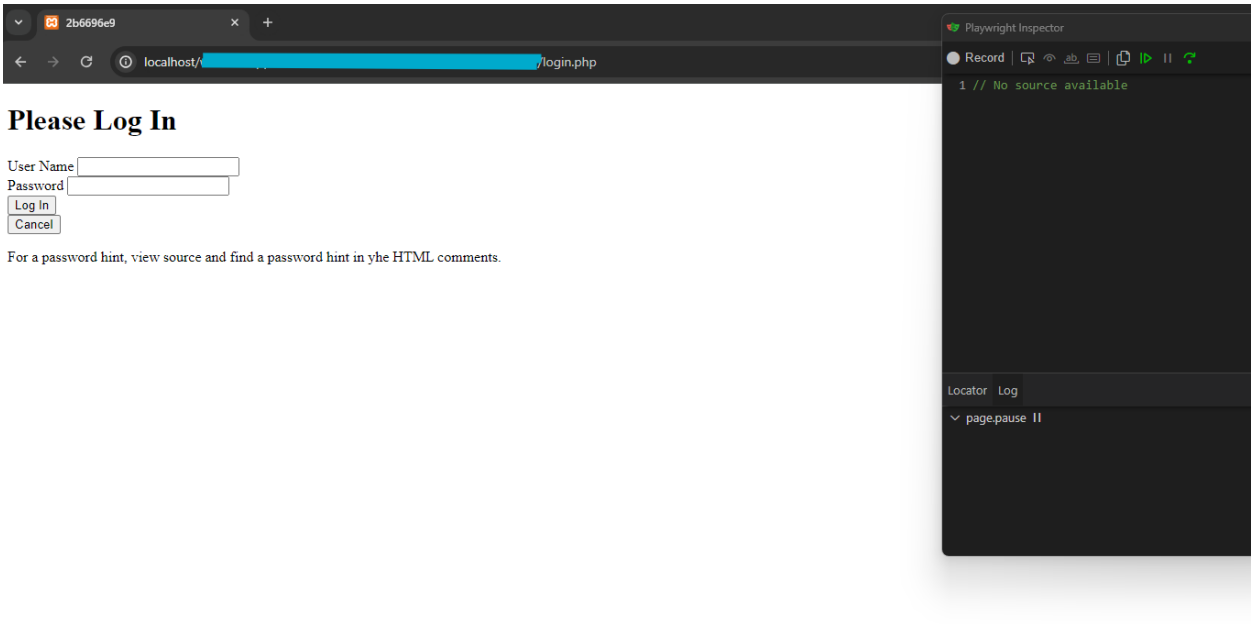
If you want to test a webpage, but you do not have much knowledge about the codebase, then locating different elements or interacting with different elements of the webpage will be quite difficult.
For locating elements on a webpage using Playwright, you have to initialize a test case then go to the url of your webpage , then add command to pause the webpage.

```
6  test("First Test Case", async({page}) => {
7      await page.goto('http://localhost/
       web%20Applications%20with%20PHP/finalModule_Course1/login.
       php');
8      await page.pause();
9  })
```

Then you can run the command in the terminal to run the test case with visualization.
`npx playwright test <your test file name> --project <desired browser> --headed`

```
)> npx playwright test sqe-test.spec.js --project chromium --headed
```

This will open the browser with Playwright IDE.

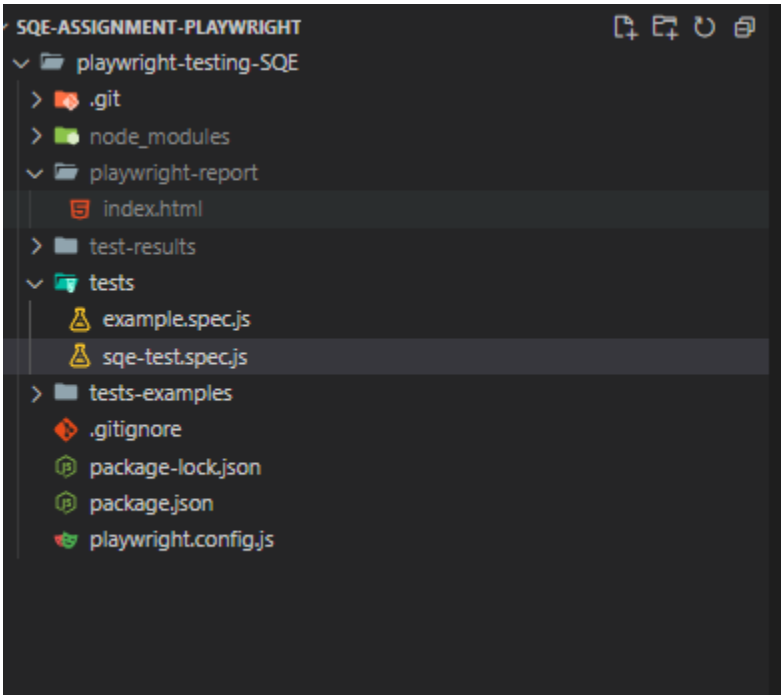


Now you can select `Pick locator` option from the options available on the top of Playwright IDE. Then you can drag your mouse cursor on different elements of the website to see their locators which then help in interacting with these elements during test case specification.

Testing website with Playwright:

I will test a PHP web application which code can be cloned from this Github repository (https://github.com/KhawarGit/Rock-Paper-Scissor-PHP).

Once the application is setup and listening on localhost, you can start developing test cases.



TEST SCRIPT
(Repository URL: https://github.com/KhawarGit/playwright-testing-SQE)

sqe-test.spec.js

```
// @ts-check
const { test, expect} = require('@playwright/test');
```

```
test('check login - Invalid Password - gives error', async (
{page} ) => {
  await
page.goto('http://localhost/web%20Applications%20with%20PHP/fin
alModule_Course1/login.php');

  //set username
  await page.fill('input[name="who"]', 'Jack');
  //set password field, inalid password
  await page.fill('input[name="pass"]', 'pass123'); //wrong
password

  // Now click the login button
  await page.click('button[value="LogIn"]');
  //wait for navigation
  // await page.waitForSelector('text=Log In', {timeout:
5000});
  await
page.waitForURL('http://localhost/web%20Applications%20with%20P
HP/finalModule_Course1/checkingLogin.php');

  await expect(page.locator('p[style="color:
red;"]')).toHaveText(
    "Incorrect Password."
  );
});

test('check login - without username - gives error', async (
{page} ) => {
  await
page.goto('http://localhost/web%20Applications%20with%20PHP/fin
alModule_Course1/login.php');
  //Not providing any username and password.

  // Now click the login button
  await page.click('button[value="LogIn"]');
  //wait for navigation
```

```
  // await page.waitForSelector('text=Log In', {timeout:
5000});
  await
page.waitForURL('http://localhost/web%20Applications%20with%20P
HP/finalModule_Course1/checkingLogin.php');
  await expect(page.locator('p[style="color:
red;"]')).toHaveText(
    "User Name and Password are required."
  );
});

test('check login - with valid username and password', async({
page }) => {
  await
page.goto('http://localhost/web%20Applications%20with%20PHP/fin
alModule_Course1/Login.php');

  //set username
  await page.fill('input[name="who"]', 'Khan');
  //set password field, inalid password
  await page.fill('input[name="pass"]', 'php123'); //correct
password
  // Now click the login button
  await page.click('button[value="LogIn"]');
  //wait for navigation
  await
page.waitForURL('http://localhost/web%20Applications%20with%20P
HP/finalModule_Course1/game.php?name=Khan');

  await expect(page.getByRole('heading', { name: 'Rock Paper
Scissors' })).toHaveText("Rock Paper Scissors");
});

test('after login - shows correct username', async({page}) => {
  await
page.goto('http://localhost/web%20Applications%20with%20PHP/fin
alModule_Course1/game.php?name=Khan');
```

```
  await expect(page.getByText('Welcome :
Khan')).toHaveText('Welcome : Khan');
});

test('Verifying all options of rock paper scissor',
async({page})=> {
  await
page.goto('http://localhost/web%20Applications%20with%20PHP/fin
alModule_Course1/game.php?name=Khan');

  //selecting option.
  await page.getByRole('combobox').selectOption('test');
  //clicking the Play button
  await page.getByRole('button', { name: 'Play' }).click();

  await expect(page.getByText('Human=rock
Computer=rock')).toHaveText(`
  Human=rock   Computer=rock   Result=Tie
  Human=rock   Computer=paper   Result=You lose
  Human=rock   Computer=scissor   Result=You win
  Human=paper   Computer=rock   Result=You win
  Human=paper   Computer=paper   Result=Tie
  Human=paper   Computer=scissor   Result=You lose
  Human=scissor   Computer=rock   Result=You lose
  Human=scissor   Computer=paper   Result=You win
  Human=scissor   Computer=scissor   Result=Tie`);

});

test('Verify logout', async({page}) => {
  await
page.goto('http://localhost/web%20Applications%20with%20PHP/fin
alModule_Course1/game.php?name=Khan');

  //helps in getting locators of variables.
  // await page.pause();
  //click on logout button.
  await page.getByRole('button', { name: 'Logout' }).click();
```

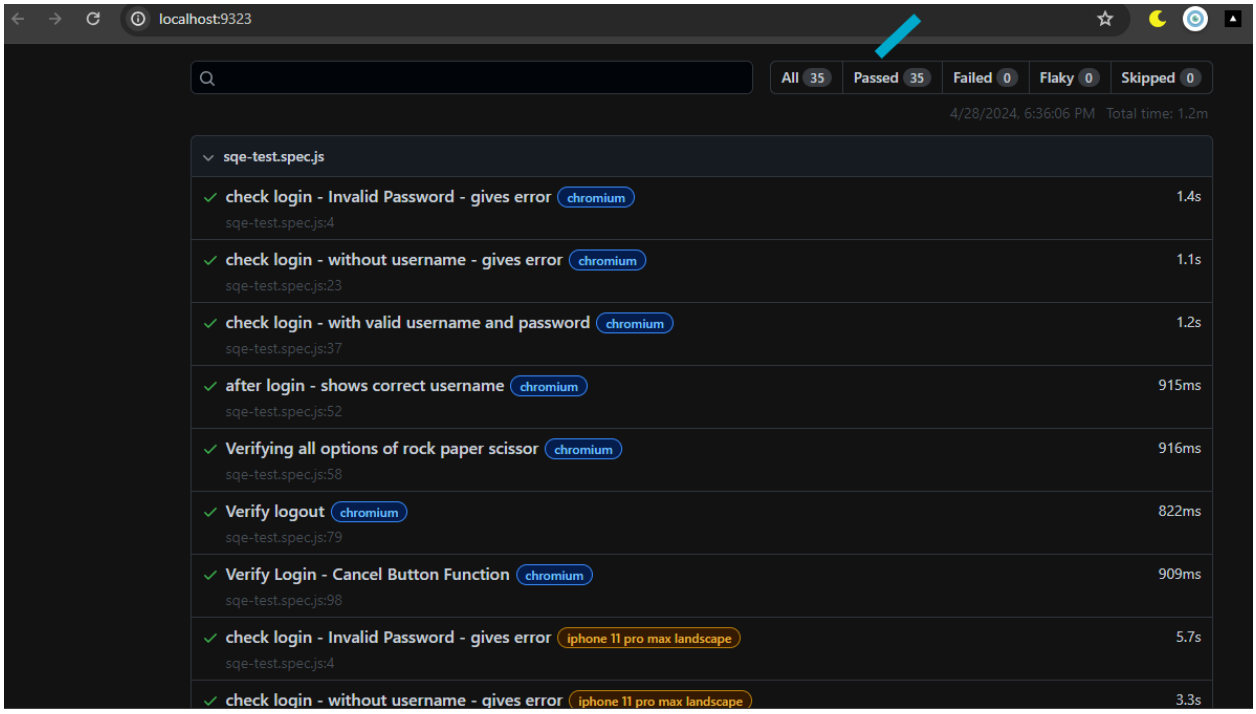
```
  // wait for this URL
  await
page.waitForURL('http://localhost/web%20Applications%20with%20P
HP/finalModule_Course1/Logout.php');

  //check the if the login page is opened.
  await expect(page.getByRole('heading', { name: 'Please Log
In' })).toHaveText('Please Log In');
  await
expect(page.locator('input[name="who"]')).toHaveText('');
  await
expect(page.locator('input[name="pass"]')).toHaveText('');


});

test('Verify Login -  Cancel Button Function', async({page})=>{
  await
page.goto('http://localhost/web%20Applications%20with%20PHP/fin
alModule_Course1/login.php');

  //set username
  await page.fill('input[name="who"]', 'Jack');
  //set password field, inalid password
  await page.fill('input[name="pass"]', 'php123');
 //await page.pause();
  //clicking Cancel button
  await page.getByRole('button', { name: 'Cancel' }).click();
  //now wait for URL to navigate
  await
page.waitForURL('http://localhost/web%20Applications%20with%20P
HP/finalModule_Course1/cancelLogin.php');

  //now check that the fields are now empty again
  await
expect(page.locator('input[name="who"]')).toHaveText('');
  await
expect(page.locator('input[name="pass"]')).toHaveText('');
```

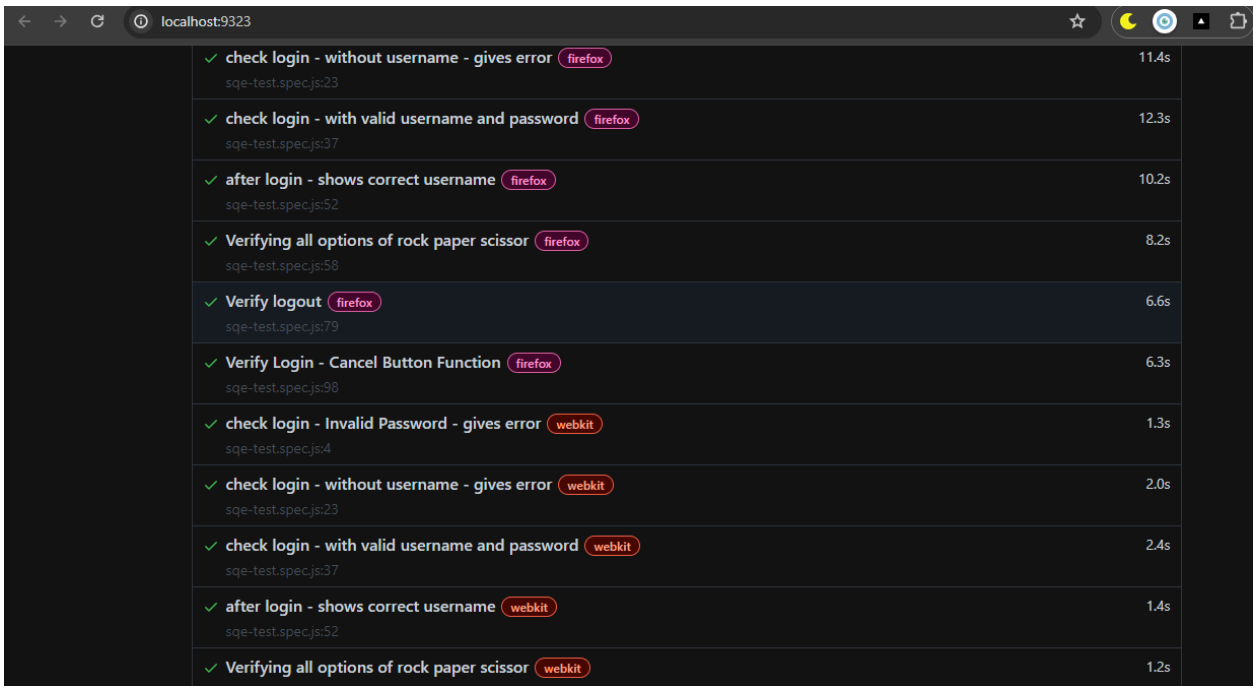
```
});
```

Running this test script

```
PS [redacted]\SQE-assignment-playwright\playwright-testing-SQE> npx playwright test sqe-test.spec.js

Running 35 tests using 4 workers
  35 passed (1.2m)

To open last HTML report run:

  npx playwright show-report

PS [redacted]\SQE-assignment-playwright\playwright-testing-SQE> npx playwright show-report

  Serving HTML report at http://localhost:9323. Press Ctrl+C to quit.
```

Now we can see report on the URL(http://localhost:9323)




localhost:9323
check login - without username - gives error firefox 11.4s
sqe-test.spec.js:23
check login - with valid username and password firefox 12.3s
sqe-test.spec.js:37
after login - shows correct username firefox 10.2s
sqe-test.spec.js:52
Verifying all options of rock paper scissor firefox 8.2s
sqe-test.spec.js:58
Verify logout firefox 6.6s
sqe-test.spec.js:79
Verify Login - Cancel Button Function firefox 6.3s
sqe-test.spec.js:98
check login - Invalid Password - gives error webkit 1.3s
sqe-test.spec.js:4
check login - without username - gives error webkit 2.0s
sqe-test.spec.js:23
check login - with valid username and password webkit 2.4s
sqe-test.spec.js:37
after login - shows correct username webkit 1.4s
sqe-test.spec.js:52
Verifying all options of rock paper scissor webkit 1.2s