**Playwrights Automation**

**Install the Playwright with command:**

npm init playwright@latest

befor writing test case we need to import some thing like

* const {test, expect} = require('@playwright/test');
* import {test, expect} from '@playwright/test'

**test:** for Test case

**expect:** for assertion

test('Home Page',async ({page})=>{

await page.goto(‘www.google.com’)

})

It test case block and it have anonymous function and we need to pass the **fixture** it called Page.

**Fixture:** is contains so many functions like find element and click element and etc.

**JavaScript promise:** async and await is used to next step depend on previous because JavaScript have the parallel execution.

**Run command:** (npx playwright test) it is run all the test cases file.

(npx playwright test Homepage.spec.js) it is run specific the test case file.

(npx playwright test –project=chromium) it is run on only the chromium other wise it is run on all the browsers.

(npx playwright test --headed) it is run the test cases Visually otherwise it is run on headless mode.

(npx playwright test Homepage.spec.js –project=chromium –headed --debug) it is help to debug the test cases.

**Locators:**

* Property
* CSS
* XPath

**For Click:**

* await page.locator(‘locator’).click()
* await page.click (‘locator’)

**Input Box:**

* await page.locator(‘locator’).fill(‘value’)
* await page.locator(‘locator’).type(‘value’)
* await page.fill(‘locator’, ‘value’)
* await page.type(‘locator’, ‘value’)

**Locate multiple web elements:**

* const element = await page.$$(locator)

**Built-in Locators:**

* [page.getByRole()](https://playwright.dev/docs/locators#locate-by-role) to locate by explicit and implicit accessibility attributes.
* [page.getByText()](https://playwright.dev/docs/locators#locate-by-text) to locate by text content.
* [page.getByLabel()](https://playwright.dev/docs/locators#locate-by-label) to locate a form control by associated label's text.
* [page.getByPlaceholder()](https://playwright.dev/docs/locators#locate-by-placeholder) to locate an input by placeholder.
* [page.getByAltText()](https://playwright.dev/docs/locators#locate-by-alt-text) to locate an element, usually image, by its text alternative.
* [page.getByTitle()](https://playwright.dev/docs/locators#locate-by-title) to locate an element by its title attribute.
* [page.getByTestId()](https://playwright.dev/docs/locators#locate-by-test-id) to locate an element based on its data-testid attribute (other attributes can be configured).

**Code Gen:**

* npx playwright codegen
* npx playwright codegen -o tests/mytestfile.spec.js
* npx playwright codegen --device “iphone 13”

**Hard Assertions:**

Hard assertion failed assertion will terminate test execution.

*// Hard Assertions - if the assertion fails, the test will stop and the test will fail*

  await expect(*page*).toHaveTitle('STORE');

  await expect(*page*).toHaveURL('https://demoblaze.com/index.html');

  await expect(*page*.locator('.navbar-brand')).toBeVisible();

**Soft Assertions:**

soft assertions: failed soft assertions do not terminate test execution, but mark the test as failed.

*// Soft Assertions - if the assertion fails, the test will continue and the test will fail*

  await expect.soft(*page*).toHaveTitle('STORE123');

  await expect.soft(*page*).toHaveURL('https://demoblaze.com/index.html');

  await expect.soft(*page*.locator('.navbar-brand')).toBeVisible();

**Input Box:**

*// input first name*

  await expect(*page*.locator('#name')).toBeVisible();

  await expect(*page*.locator('#name')).toBeEmpty();

  await expect(*page*.locator('#name')).toBeEditable();

  await expect(*page*.locator('#name')).toBeEnabled();

  await *page*.locator('#name').fill('John');

  await expect(*page*.locator('#name')).toHaveValue('John');

**Radio Button:**

*// radio button*

  await *page*.locator('#male').check();

  await expect(*page*.locator('#male')).toBeChecked();

  await expect(*page*.locator('#male').isChecked()).toBeTruthy();

*// check female is unchecked*

  await expect(*page*.locator('#female')).not.toBeChecked();

  await expect(*page*.locator('#male').isVisible()).toBeTruthy();

  await expect(*page*.locator('#male').isEnabled()).toBeTruthy();

  await expect(*page*.locator('#male').isEditable()).toBeTruthy();

*// check male is not disabled and not hidden*

  const isDisabled = await *page*.locator('#male').isDisabled();

  expect(isDisabled).toBeFalsy();

  const isHidden = await *page*.locator('#male').isHidden();

  expect(isHidden).toBeFalsy();

**CheckBox:**

await page.locator(‘locator’).check()

await page.locator(‘locator’).toBeChecked()

await page.locator(‘locator’).isChecked() return True False{.toBeTruthy(), .toBeFalsy()}

await page.locator(‘locator’).unCheck()

**Dropdown Menu:**

* **By label:**

await page.locator('#country').selectOption({label: 'India'})

* **By Text:**

await page.locator('#country').selectOption('India')

await page.selectOption("#country", 'India')

* **By Value:**

await page.locator('#country').selectOption({value: 'uk'})

* **By Index:**

await page.locator('#country').selectOption({index: 1})

**Multi select Dropdown:**

await page.selectOption('#colors', ['Blue', 'red', 'Yellow'])

**Auto Suggest Dropdown**

*//1) Select the city*

  await *page*.locator('#src').fill('Delhi');

  await *page*.waitForSelector("//li[contains(@class,'sc-iwsKbI')]/div/text[1]");

  await *page*.$$("//li[contains(@class,'sc-iwsKbI')]/div/text[1]");

  const fromCityOptions = await *page*.$$("//li[contains(@class,'sc-iwsKbI')]/div/text[1]");

  for (const option of fromCityOptions) {

    const value = await option.textContent();

    console.log(value);

    if (value.includes('Anand Vihar')) {

      await option.click();

      break;

    }

**Alert:**

By default, dialogs are auto-dismissed by Playwright, so you don't have to handle them. However, you can register a dialog handler before the action that triggers the dialog to either **dialog**.**accept()** or **dialog.dismiss()** it.

* Alert with OK

*page*.on('dialog', async *dialog* => {

    expect(*dialog*.type()).toContain('alert');

    expect(*dialog*.message()).toContain('I am an alert box!');

    await *dialog*.accept();

  });

  await *page*.click("#alertBtn")

* Confirmation Dialog-Alert with Ok and Cancel

*page*.on('dialog', async *dialog* => {

      expect(*dialog*.type()).toContain('confirm');

      expect(*dialog*.message()).toContain('Press a button!');

      await *dialog*.accept(); *// close by using ok button*

*//await dialog.dismiss(); // close by using cancel button*

    });

    await *page*.click("//button[@id='confirmBtn']");

    await expect(*page*.locator("//p[@id='demo']")).toHaveText('You pressed OK!');

* Prompt Dialog-Alert with Ok and Cancel

*page*.on('dialog', async *dialog* => {

      expect(*dialog*.type()).toContain('prompt');

      expect(*dialog*.message()).toContain('Please enter your name:');

      expect(*dialog*.defaultValue()).toContain('Harry Potter');

      await *dialog*.accept('John'); *// close by using ok button*

*//await dialog.dismiss(); // close by using cancel button*

    });

    await *page*.click("//button[@id='promptBtn']");

    await expect(*page*.locator("//p[@id='demo']")).toHaveText('Hello John! How are you today?');

**Frames:**

* await page.frame({url: 'https://ui.vision/demo/webtest/frames/frame\_1.html'}).fill("[name='mytext1']", 'Aakash')
* await page.frameLocator("frame[src='frame\_1.html']").locator("[name='mytext1']").fill('Aakash')

**Inner Frames**

await *page*.goto('https://ui.vision/demo/webtest/frames/');

    const frame4 = await *page*.frame({url: 'https://ui.vision/demo/webtest/frames/frame\_3.html'});

    await frame4.fill("[name=mytext3]",'Hello');

*// nested frames*

    const childFrame = await frame4.childFrames();

    await childFrame[0].locator("//\*[@id='i6']/div[3]/div").check(); *// childFrame[0] is the first child frame, childFrame[1] is the second child frame*

**Table:**

* const table = await page.locator('#productTable')
* const colums = await table.locator('thead tr th')
* const rows = await table.locator('tbody tr')

**Mouse Actions:**

* await page.locator("//a[normalize-space()='Desktops']").hover()
* await page.locator("//span[@class='context-menu-one btn btn-neutral']").click({button: 'right'})
* await page.locator("//button[normalize-space()='Copy Text']").dblclick()
* await page.locator("//button[normalize-space()='Copy Text']").press('Control+Shift+KeyR')
* Approach 1: Using dragTo() method
  + await page.locator('#item-to-be-dragged').dragTo(page.locator('#item-to-drop-at'))
* Approach 2: Dragging manually
* await *page*.locator('#draggable').hover();
* await *page*.mouse.down();
* await *page*.locator('#droppable').hover();
* await *page*.mouse.up();
* *// verify the text*
* await expect(*page*.locator('#droppable')).toHaveText('Dropped!');

**Upload Files:**

* Select one file

await page.getByLabel('Upload file').setInputFiles(path.join(\_\_dirname, 'myfile.pdf'));

await *page*.locator("#singleFileInput").setInputFiles("tests/uploadFiles/testfile1.pdf");

* Select multiple files

await page.getByLabel('Upload files').setInputFiles([ path.join(\_\_dirname, 'file1.txt'), path.join(\_\_dirname, 'file2.txt'),]);

await *page*.locator("#multipleFilesInput").setInputFiles(['tests/uploadFiles/testfile1.pdf', 'tests/uploadFiles/testfile2.pdf']);

**Playwright Hooks:**

* **beforeEach:** This hook is executed before each individual test.
* **afterEach:** This hook is executed after each individual test.
* **beforeAll:** This hook is executed once before any of the tests start running.
* **afterAll:** This hook is executed once after all the tests have been run.