YASH Technologies SM

More than what you think.

Playwright with Typescript

Agenda

- ❑ Playwright Introduction
- ❑ Playwright Installation
- ❑ What is TypeScript?
- ❑ Use chrome instead of chromium
- ❑ await / async in Java Script / Type Script
- ❑ Playwright Fixtures
- ❑ Locator Method using Selectors (Xpath, CSS Selector, Text, Id)
- ❑ Locator Method usage with Optional Argument in Playwright
- ❑ getBy methods in Playwright

Agenda

- ❑ Assertions and Soft assertions in Playwright
- ❑ Playwright configuration file playwright.config.ts
- ❑ Test Annotations
- ❑ Playwright Hooks
- ❑ fill, pressSequentially and press methods in Playwright
- ❑ Click, Double Click, Right Click and Programmatic Click in Playwright
- ❑ Some Playwright Techniques

- Playwright is an open-source test automation library initially developed by Microsoft.
- Playwright Test was created specifically to accommodate the needs of end-to-end testing.
- Playwright operates through the DevTools Protocol, which allows smooth browser control.
- Test on Windows, Linux, and macOS, locally or on CI, headless or headed with native mobile emulation of Google Chrome for Android and Mobile Safari.

Some Advantages of Playwright -

1- Easy Setup and Configuration

2- Multi-Browser Support - Chromium browsers (Chrome, Edge), Webkit (Safari), and Firefox

3- Multi-Language Support - Java Script / Type Script, Python, Java, .NET

4- Types of Testing - Functional, End to End, API Testing and Visual Regression Testing

5- Built-in Reporters - List, Dot, Line, JSON, JUnit, HTML and Allure reports

6- Parallel Browser Testing

7- CI/CD Integration Support

8- Various inbuilt tools like Codegen, Playwright Inspector, Trace Viewer

Pre-Requisite required -

Visual Studio Code IDE - https://code.visualstudio.com/download

Node JS Installation - https://nodejs.org/en/download

Playwright installation with VS Code extension

- navigate to open folder
- Install extension Playwright test for VSCode developed by microsoft
- Goto command pallate with (ctrl+shift+P)
- Select "Test Playwright Install" option.
- Select playwright components to install (Dont select use javascript)
- This will install playwright using terminal

What is TypeScript?

- TypeScript is a syntactic superset of JavaScript which adds **static typing**
- This basically means that TypeScript adds syntax on top of JavaScript, allowing developers to add **types**.
- TypeScript allows specifying the types of data being passed around within the code and has the ability to report errors when the types don't match, which means
  - TypeScript uses compile time type checking.

```
async enterDealNumber(value:string){
    await expect(this.dealNumberField).toBeEditable();
    await this.dealNumberField.fill(value);
}
```

arguments with type

```
async getTitleField() :Promise<Locator>{
    return this.titleField;
}
```

function with return type

Use chrome instead of chromium

- If you want to use Chrome browser instead of Chromium then change playwright.config.ts
- Replace the following lines:-

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

with

```
{
    name: 'chromium',
    // use: { ...devices['Desktop Chrome'] },
    use: { channel: "chrome" },
},
```

await / async in Java Script / Type Script

- **await** operator is used to wait for the **promise**, it allows us to wait until the **promise** has been completed before moving onto the next line.
- JavaScript require to use of the **await** keyword inside a function marked with an **async** keyword.
- The async/await syntax simplifies working with promises in JavaScript. It provides an easy interface to read and write promises in a way that makes them appear synchronous.
- async always goes hand in hand with await. That is, you can only await inside an async function. The async function informs the compiler that this is an asynchronous function.

Playwright **Fixtures**

- Playwright Test is based on the concept of the test fixtures.
- Test fixtures are used to establish environment for each test, giving the test everything it needs.
- The { page } argument tells Playwright Test to setup the page fixture and provide it to your test function.
- Playwright Test will setup the page fixture before running the Test and tear it down after the test has finished.
- Test fixtures are isolated between tests.

Fixture	Type	Description
page	Page	Isolated page for this test run.
context	BrowserContext	Isolated context for this test run. The `page` fixture belongs to this context as well. Learn how to configure context.
browser	Browser	Browsers are shared across tests to optimize resources. Learn how to configure browser.
browserName	string	The name of the browser currently running the test. Either `chromium`, `firefox` or `webkit`.
request	APIRequestContext	Isolated APIRequestContext instance for this test run.

```
import { test, expect } from '@playwright/test';

test('basic test', async ({ page }) => {
  await page.goto('/signin');
  await page.getByLabel('User Name').fill('user');
  await page.getByLabel('Password').fill('password');
  await page.getByText('Sign in').click();
  // ...
});
```

Locator Method using Selectors (Xpath, CSS Selector, Text, Id)

- Locators represent a way to find element(s) on the page at any moment.
- Locator can be created with the page.locator() method.
- Locator method returns an element locator that can be used to perform actions on page / frame.

Locator Method Usage

- page.locator(selector); or page.locator(selector, options);

Different Types of Selector

- **Xpath** - XPath is a technique to navigate through a page's HTML structure**.**

//htmltag[@attribute='attributeValue']

- **Css Selectors -** CSS Selectors are string patterns used to identify an element based on a combination of HTML tag, id, class, and attributes.

htmltag.classvalue or htmltag#idValue / #idValue or htmltag[attributeName=attributeValue]

- **Text**

text='textValue' - Cases Sensitive Text

data-testid, data-test-id, data-test, ……. and So on. id/data-test-id/data-test=value

Locator Method usage with Optional Argument in Playwright

Syntax -

- page.locator(selector, options);

Different Type of Options -

- **has Locator** - Matches elements containing an element that matches an inner locator.
- **hasNot Locator** - Matches elements that do not contain an element that matches an inner locator.
- **hasText Locator** - Matches elements containing specified text somewhere inside, possibly in a child or a descendant element.
- **hasNotText locator** - Matches elements that do not contain specified text somewhere inside, possibly in a child or a descendant element.

Note - (Regular expressions are patterns used to match character combinations in strings.)

```
await page.locator("div.TableRow-focusBorder",{has:page.locator("//a[text()='"+name+"']")}).hover();
```

```
await  page.locator("button[type='button']", {hasText:"New"}).click();
```

getBy methods in Playwright

- **getByLabel** -
  - Allows locating input elements by the text of the associated <label> or aria-labelledby element, or by the aria-label attribute.
  - Accessible Rich Internet Applications (ARIA) is a set of roles and attributes that define ways to make web content and web applications more accessible to people with disabilities.
  - Use this locator when locating form fields.
- **getByPlaceHolder** -
  - Allows locating input elements by the placeholder text.
  - Use this locator when locating form elements that do not have labels but do have placeholder texts.
- **getByText -**
  - Allows locating elements that contain given text.
  - You can match by a substring, exact string, or a regular expression when using this method.
  - It is recommended using text locators to find non-interactive elements like div, span, p, etc. For interactive elements like button, a, input, etc. use Role locator.

getBy methods in Playwright

- **getByAltText -** Allows locating elements by their alt text.
  - Use this locator when your element supports alt text such as img and area elements.
- **getByTitle -** Allows locating elements by their title attribute.
  - Use this locator when your element has the title attribute.
- **getByRole -**Allows locating elements by their ARIA role, ARIA attributes and accessible name.
  - Accessible Rich Internet Applications (ARIA) is a set of roles and attributes that define ways to make web content and web applications more accessible to people with disabilities.
  - Role locators include buttons, checkboxes, headings, links, lists, tables, and many more and follow W3C specifications for ARIA role, ARIA attributes and accessible name.
  - It is recommend prioritizing role locators to locate elements, as it is the closest way to how users and assistive technology perceive the page.
- **getByTestId -** Locates an element based on it's data-testid (Other attributes can be configured).
  - You can also use test ids when you choose to use the test id methodology or when you can't locate by role or text.

Assertions in Playwright

- We can call assertions as verifications or checks which can be done in our tests, Playwright includes test assertions in the form of expect function.

Some Common Assertions

- await expect(locator).toHaveCount() - To check number of elements are present on the page or not.
- await expect(locator).toBeEnabled() - To check if element is enabled or not.
- await expect(locator).toBeDisabled() - To check if an element is disabled or not.
- await expect(locator).toBeVisible() - To check if an element is visible or not.
- await expect(locator).toBeHidden() - To check Element is hidden or not.
- await expect(locator).toHaveText('text') - To check Element have specified text or not.
- await expect(locator).toHaveAttribute('attribute', 'AttributeValue') - To check if the element have specified value of attribute
- await expect(locator).toHaveId('IdValue') - To check if the element have specified value of id
- await expect(locator).toHaveClass('ClassValue') - To check if the element have specified value of id
- await expect(page).toHaveURL('URLValue') - To check if the page has specified URL or not
- await expect(page).toHaveTitle('TitleValue') - To check if the page has specified Title or not

Soft assertions

- Playwright also supports soft assertions.
- Failed soft assertions do not terminate test execution but mark the test as failed.

Exampless -

- expect.soft(value).not.toEqual(0);
- await expect.soft(locator).toHaveText('some text’);

- **Custom Expect Message**

You can specify a custom error message as a second argument to the expect function.

Examples -

```
await expect(locator, ‘Custom Error Message’).not.toHaveText('some text');
```

Playwright configuration file playwright.config.ts

- Playwright has many options to configure how your tests are run. You can specify these options in the configuration file.

Below are some options.

- testDir
- fullyParallel
- forbidOnly
- retries
- workers
- reporter
- Timeout
- expect with {timeout}
- use with {trace, headless}
- Projects with name and use
- headless

```
export default defineConfig({
  testDir: './tests',
  /* Run tests in files in parallel */
  fullyParallel: false,
  /* Fail the build on CI if you accidentally left test.only in the source code. */
  forbidOnly: !!process.env.CI,
  /* Retry on CI only */
  retries: process.env.CI ? 2 : 0,
  /* Opt out of parallel tests on CI. */
  workers: process.env.CI ? 1 : 3,
  /* Reporter to use. See https://playwright.dev/docs/test-reporters */
  reporter: [['html',{open:'never'}]],
   /* You can change the  default timeout duration which is 30000 ms */
  timeout:30000,
  /* Shared settings for all the projects below. See https://playwright.dev/docs/api/class-testoptions. */
  use: {
    /* Base URL to use in actions like `await page.goto('/')`. */
    // baseURL: 'http://127.0.0.1:3000',

    /* Collect trace when retrying the failed test. See https://playwright.dev/docs/trace-viewer */
    screenshot:'on',
    video:'on',
    trace: 'on-first-retry',
    /* change headless option. Default is true. */
    headless:false,
  },
```

Test Annotations

- **Playwright Test supports test annotations to deal with failures, flakiness, skip, focus etc**
- **test.describe()** - Declares a group of tests.
- **test.only()** - Declares a focused test. If there are some focused tests or suites, all of them will be run but nothing else.
- **test.skip()** - marks the test as irrelevant. Playwright Test does not run such a test. Use this annotation when the test is not applicable in some configuration.
- **test.fixme()** - Declares a test to be fixed, similarly to test(). This test will not be run.
- **test.slow()** - marks the test as slow and triples the test timeout.
- **test.setTimeOut()** - Changes the timeout for the currently running test. Zero means no timeout.
- **test.fail()** - Unconditionally marks a test as "should fail". Playwright Test runs this test and ensures that it is actually failing. This is useful for documentation purposes to acknowledge that some functionality is broken until it is fixed.

Playwright Hooks

- **beforeEach -**

Syntax -

- test.beforeEach(hookFunction) or test.beforeEach(title, hookFunction)
- beforeEach hook that is executed before each test.
- When called in the scope of a test file, runs before each test in the file.
- When called inside a test.describe() group, runs before each test in the group.
- If multiple beforeEach hooks are added, they will run in the order of their registration.

- **afterEach -**

Syntax -

- test.afterEach(hookFunction) or test.afterEach(title, hookFunction)
- afterEach hook that is executed after each test.
- When called in the scope of a test file, runs after each test in the file.
- When called inside a test.describe() group, runs after each test in the group.
- If multiple afterEach hooks are added, they will run in the order of their registration.

Playwright Hooks

❑ **beforeAll -**

Syntax -

- test.beforeAll(hookFunction) or test.beforeAll(title, hookFunction)
- beforeAll hook that is executed once per worker process before all tests.
- When called in the scope of a test file, runs before all tests in the file.
- When called inside a test.describe() group, runs before all tests in the group.
- If multiple beforeAll hooks are added, they will run in the order of their registration.

❑ **afterAll -**

Syntax -

test.afterAll(hookFunction) or test.afterAll(title, hookFunction)

- afterAll hook is executed once per worker after all tests.
- When called in the scope of a test file, runs after all tests in the file.
- When called inside a test.describe() group, runs after all tests in the group.
- If multiple afterAll hooks are added, they will run in the order of their registration.

fill, pressSequentially and press methods in Playwright

- **locator.fill() -** Using locator.fill() is the easiest way to fill out the form fields. It focuses the element and triggers an input event with the entered text. It works for <input>, <textarea> and [contenteditable] elements.
- **locator.pressSequentially()** - The locator.pressSequentially() Type into the field character by character, as if it was a user with a real keyboard.
- **locator.press()** -The locator.press()The method focuses the selected element and produces a single keystroke. It accepts the logical key names.

```
// Hit Enter
await page.getByText('Submit').press('Enter');

// Dispatch Control+Right
await page.getByRole('textbox').press('Control+ArrowRight');

// Press $ sign on keyboard
await page.getByRole('textbox').press('$');
```

```
Backquote, Minus, Equal, Backslash, Backspace, Tab, Delete, Escape,
ArrowDown, End, Enter, Home, Insert, PageDown, PageUp, ArrowRight,
ArrowUp, F1 - F12, Digit0 - Digit9, KeyA - KeyZ, etc.
```

Click, Double Click, Right Click, and Programmatic Click in Playwright

- **Click** - click() method performs a simple human click on an element. -

Syntax - locator.click()

- **Double Click** - dblclick() method performs a simple double click on an element. -

Syntax - locator.dblclick()

- **Right Click / Context click** - click({button:'right'}) method performs a simple right click.

Syntax - locator.click({button:'right'})

- **Programmatic click -** If you are not interested in testing your app under the real conditions and want to simulate the click by any means possible, you can trigger the HTMLElement.click() behaviour via simply dispatching a click event on the element.

Syntax - locator.dispatchEvent('click')

```
// Generic click
await page.getByRole('button').click();

// Double click
await page.getByText('Item').dblclick();

// Right click
await page.getByText('Item').click({ button: 'right' });

// Shift + click
await page.getByText('Item').click({ modifiers: ['Shift'] });

// Ctrl + click or Windows and Linux
// Meta + click on macOS
await page.getByText('Item').click({ modifiers: ['ControlOrMeta'] });

// Hover over element
await page.getByText('Item').hover();

// Click the top left corner
await page.getByText('Item').click({ position: { x: 0, y: 0 } });
```

Some Playwright Techniques

- TestInfo object - TestInfo contains information about currently running test. It is is available in every Playwright test case as the second argument.
  - It is available to all test functions and hooks.
  - TestInfo provides utilities to control test execution: attach files, update test timeout, determine which test is currently running and whether it was retried, etc.

```
import { test, expect } from '@playwright/test';

test('basic test', async ({ page }, testInfo) => {
  await page.goto('https://playwright.dev');
  const screenshot = await page.screenshot();
  await testInfo.attach('screenshot', { body: screenshot, contentType: 'image/png
});
```

- slowMo - literally slows down browser interactions.
  - It is invaluable for debugging, allowing developers to observe the browser's actions step-by-step.
  - The delay is applied to actions like clicks, typing, and navigation.
  - Apply slowMo when you need to observe the browser's actions in real-time for debugging

```
/* Configure projects for major browsers */
projects: [
  {
    name: 'chromium',
    // use: { ...devices['Desktop Chrome'] },
    use: { channel: "chrome",
      launchOptions:{
        slowMo:800,
      },
    },

  },
```

YASH Technologies

More than what you think.

Thank you!