⇒ Playwright ( Java automation tool )

⇒ Characteristics (it is cross-browser, cross-platform, cross-language, test - mobile web)

- Browser context: it creates browser context for each test. It is similar to create another profile
- Log-in once: It save the authentication state and re-use it (so no need to login again and again)
- Codegen: generate tests by recording your actions.
- Playwright inspection: Inspect page, generate selectors
- Trace viewers: It capture all the values to investigate like screenshot, live DOM screenshat, action explorer, test source etc.

# Features :-

**auto-wait :** it has auto-wait feature, where it will wait for element to appear before working on it.

**web-first assertion** Playwright is created specifically for dynamic web pages Checks are automatically re-tried until all cond^n are met.

**Tracing :-** It captures screenshot, video, execution trace.

**Multiple everything :-** create scenarios with diff. context for diff. users and run them against server all in one test

**Trusted events :-** hover element, interact with dynamic control and produce trusted event. It uses a real browser input pipeline indistinguisable from a real user.

## ⇒ Some basic annotations

@Test :- This annotation tell JUnit framework that this current method can be used as a test method.

@BeforeEach :- Method attached with this annotation runs before every test method.

@AfterEach :- Method attached with this annotation runs after every test method.

@BeforeAll :- Method attached with this annotation runs ones before all test method.

@AfterAll :- Method attached with this annotation runs ones after all test method.

# ⇒ Example of a simple script

1)
```
static Playwright playwright = Playwright.create();
```
this launch a brower and return a browser object

will be used to interact and work

2)
```
static Browser browser = playwright.chromium().launch(new BrowserType.LaunchOptions().setHeadless(false));
```
this launch a chrome

3)
```
static Page page = browser.newPage();
```
→ we open a new web page

4)
```
page.navigate("http://10.82.180.36/Common/Login.aspx");
```
→ open application by sending link

Example

```
public class Demo1 {
    static Playwright playwright = Playwright.create();
    static Browser browser = playwright.chromium().launch(new BrowserType.LaunchOptions().setHeadless(false));
//  static BrowserContext bc = browser.newContext();
    static Page page = browser.newPage();

    @BeforeAll
    public static void setUpBeforeClass() throws Exception {

        page.navigate("http://10.82.180.36/Common/Login.aspx");
        System.out.println(page.url());       // To print the URL of page
        System.out.println(page.title());     // To get the title of page
        System.out.println(page.content());   // To get the Page source

    }

    @AfterAll
    public static void tearDownAfterClass() throws Exception {
        page.close();

        playwright.close();
    }

    @Test
    public void test() throws InterruptedException {


        page.goBack();        // To go a step back in Page
        page.goForward();     // To go a step forward in Page
        page.reload();        // To refresh the page

    }

}
```

⇒ Some method which might come handy later

| Methods | Description |
|---|---|
| navigate() | Accesses the browser's history and navigates to a given URL. |
| title() | Fetches the title of the currently opened web page. |
| url() | Fetches the URL of the currently opened web page. |
| content() | Fetches the page source of any webpage. |
| close() | To close the browser related instances. |
| goBack() | Allows you to navigate back to the previous page in the browser history. |
| goForward() | Allows you to navigate forward to the next page in the browser history. |
| reload() | allows you to reload the current page in the browser. |

→ To maximise the brower size, we dont have any direct method

But there are 2 ways

1) Direct method :-

```
page.setViewportSize(1366, 768);
```

(by giving the dimensions)

2) By taking help from Dimension class :-

```
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
int screenWidth = (int) screenSize.getWidth();
int screenHeight = (int) screenSize.getHeight();

// Set the viewport size of the page to the screen size
page.setViewportSize(screenWidth, screenHeight);
page.navigate("http://10.82.180.36/Common/Login.aspx");
```

# ⇒ Locators

| Methods | Description |
|---|---|
| Page.getByRole() | To locate by explicit and implicit accessibility attributes. |
| Page.getByText() | To locate by text content. |
| Page.getByLabel() | To locate a form control by associated label's text. |
| Page.getByPlaceholder() | To locate an input by placeholder. |
| Page.getByAltText() | To locate an element, usually image, by its text alternative. |
| Page.getByTitle() | To locate an element by its title attribute. |
| Page.getByTestId() | To locate an element based on its data-testid attribute (other attributes can be configured). |

Apart from these, we use id, xpath and CSSSelectors

Example:-

```
@Test
void test() throws InterruptedException {
    page.locator("xpath=//*[@id=\"body_txtUserID\"]").fill("donhere");    // Locate element by xpath
    page.locator("id=body_txtPassword").fill("don@123");                  // Locate element by id
    page.getByRole(AriaRole.BUTTON, new Page.GetByRoleOptions().setName("Login")).click();   // Locate element by role

}
```

# ⇒ Assertions

| Assertion | Description |
|---|---|
| assertThat(locator).isChecked() | Checkbox is checked |
| assertThat(locator).isDisabled() | Element is disabled |
| assertThat(locator).isEditable() | Element is enabled |
| assertThat(locator).isEmpty() | Container is empty |
| assertThat(locator).isEnabled() | Element is enabled |
| assertThat(locator).isFocused() | Element is focused |
| assertThat(locator).hasText() | Element matches text |
| assertThat(locator).hasValue() | Input has a value |
| assertThat(locator).hasValues() | Select has options selected |
| assertThat(page).hasTitle() | Page has a title |
| assertThat(locator).isHidden() | Element is not visible |
| assertThat(locator).isVisible() | Element is visible |
| assertThat(locator).containsText() | Element contains text |
| assertThat(locator).hasAttribute() | Element has a DOM attribute |

⇒ Synchronization (means waiting for an element)

↳ Here, we have auto-wait feature by default

But apart from it we can have

Implict wait :- `page.setDefaultTimeout(timeout);`

Explicit wait :- wait for a specific element

```
// Set the explicit timeout to 10 seconds and wait for element with "id=body_txtUserID"
page.waitForSelector("id=body_txtUserID", new Page.WaitForSelectorOptions().setTimeout(10000));
```