PLAYWRIGHT FULL DOCUMENT

ABSTRACT

The objective of this article is to track the JavaScript-based Playwright resources that are required for future study and learning.

Shaleh (Md. Abu Shaleh), IT-EA

Installation

System Requirements:

- Node.js 16+
- Windows 10+, Windows Server 2016+ or Windows Subsystem for Linux (WSL).
- MacOS 12 Monterey or MacOS 13 Ventura.
- Debian 11, Debian 12, Ubuntu 20.04 or Ubuntu 22.04

Installation:

Step 1: Create a folder on destined location.

Step 2: Ensure the NodeJS version is above 14 from the command prompt.

Step 3: Open the created folder and type **“cmd”** in the address bar.

Step 4: Type **“code”**. to launch VS Code. You can now see the created folder as a project in VS Code.

Step 5: Navigate to the **EXTENSIONS** section and type **"Playwright"**. There are multiple options available. Select the **Playwright Test for VS Code by Microsoft** option.

Step 6: Click **Install**.

Step 7: Press **CTRL + SHIFT + P** to open the command panel and type **"install Playwright"**.

Step 8: Enable the Chromium, Firefox, and WebKit checkboxes, as Playwright supports all browser engines.

Step 9: Click **Okay**.

Step 10: Expand the **Project Explorer**, and the **“example.spec.ts”** file is displayed.

Step 11: To execute the test, click the **Green arrow** icon.

Step 12: To disable the Headless mode, navigate to the **playwright.config.ts** file.

Step 13: In the **use block**, type **“headless: false,”** and execute again to view the browser.

Writing Tests

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

// test: package to test playwright || except: package for validation for the test

test('has title', async ({ page }) => {

// test (): test block || async {()}: anonymous function. This will use some fixture (page)
provided by playwright.

//Scripts should be written here

});
```

Basic Actions

Action	Description
locator.check()	Check the input checkbox
locator.click()	Click the element
locator.uncheck()	Uncheck the input checkbox
locator.hover()	Hover mouse over the element
locator.fill()	Fill the form field, input text
locator.focus()	Focus the element
locator.press()	Press single key
locator.setInputFiles()	Pick files to upload
locator.selectOption()	Select option in the drop down

Basic Assertions

Assertion	Description
expect(locator).toBeChecked()	Checkbox is checked
expect(locator).toBeEnabled()	Control is enabled
expect(locator).toBeVisible()	Element is visible
expect(locator).toContainText()	Element contains text
expect(locator).toHaveAttribute()	Element has attribute
expect(locator).toHaveCount()	List of elements has given length
expect(locator).toHaveText()	Element matches text
expect(locator).toHaveValue()	Input element has value

Assertion	**Description**
expect(page).toHaveTitle()	Page has title
expect(page).toHaveURL()	Page has URL

Example:

await except (await page.locator(“Xpath”/’Loactors’)).toBe……

<u>**Running tests**</u>

Run tests in UI Mode

npx playwright test –ui

Run tests by Terminal Command

- Running all tests

  npx playwright test
- Test by Recording

  npx playwright codegen
- Running a single test file

  npx playwright test landing-page.spec.ts
- Running tests in headed mode

  npx playwright test landing-page.spec.ts –headed
- Running tests on a specific project

  npx playwright test landing-page.ts --project=chromium
- Debugging all tests

  npx playwright test –debug
- Debugging one test file

npx playwright test example.spec.ts --debug

❖ Test Reports

npx playwright show-report

Description of Commands/Scripts

// JavaScript is asynchronous programming language. To make it synchronous we need to use async and await keyword.

// await keyword ensure to wait for the promise. Once the page is loaded then the next step will keep continue.

Common:

❖ **To redirect to an url:**

await page.goto('the url');

❖ **To verify page title:**

const pageTitle = page.title();

await except(page).toHaveTitle('Title found on the website');

❖ **To verify url:**

await except(page).toHaveURL('URL of the website');

❖ **To close the page:**

await page.close();

Locators:

❖ **To Specifiy the propery:**

await page.locator('id/name/class = defined id/name/class').click();

await page.click('id/name/class = defined id/name/class');

For Xpath

await page.locator(“Xpath”).click();

await page.click(“Xpath”);

- **Input Field:**

  await page.locator(id/name/class = defined id/name/class’).fill("Value");

  await page.fill(‘id/name/class = defined id/name/class’,’Value’);

  await page.type(‘id/name/class = defined id/name/class’,’Value’);

  For Xpath

  await page.locator(“Xpath”). fill("Value");

  await page.fill(“Xpath”,’Value’);

Locate Multiple Web Elements:

```
const elements = await page.$$(‘Locator’);

for(const element of elements)
{
const elementtext = await element.textContent();
console.log(elementtext);
}
```

Example:

```
const elements = await page.$$("//*[@id='inventory_container']//div/a/div");
for(const element of elements)
{
const elementtext = await element.textContent();
console.log(elementtext);
}
```

Xpath should be selected in such a way that can get all the item in the webpage.

Then, any of these can be used to get the desired item.

- 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).

Example:

await page.getByLabel('User Name').fill('John');

await page.getByLabel('Password').fill('secret-password');

await page.getByRole('button', { name: 'Sign in' }).click();

await expect(page.getByText('Welcome, John!')).toBeVisible();

Wait Function:

await page.waitForTimeout(milliseconds); //pausing code execution

Radio Button & Checkbox:

await page.locator(“Xapth”/’Other Locators’).check();

await page.check(“Xapth”/’Other Locators’);

DropDown Menu:

await page.locator (‘Locators’).selectOptions({label:’value’});

await page.locator (‘Locators’).selectOptions(’value’); //visible Text

await page.locator (‘Locators’).selectOptions({value:’value’});

await page.locator (‘Locators’).selectOptions({index:value}); //numbers

// Check to see all the options and select desired:

```
const options=await page.$$(‘Locators’)
        let status=false;
        for(const option of options)
        {
        let value = await option.textContent();
        if(value.includes(‘Specific option’))
        {
        status=true; / await.page.selectOptions(“Locators”,’value’);
        break;
        }
        }
```

except(status).toBeTruthy(); // if await.page.selectOptions(“Locators”,’value’); is give no need to put this code. It is only applicable for status=true;

// Multi Select Drop-down:

```
const options=await page.$$(‘Locators’)
        for(const option of options)
        {
        let value = await option.textContent();
        if(value.includes(‘Specific option’) || value.includes(‘Specific option2’))
        {
        await option.click();
        }
        }
```

// Auto Suggest Drop-Down:

```
await page.locator(‘Locators’).fill(‘Value’);
        const autoSuggest = await page.$$(“Xpath”)
// Xpath should be selected in such a way that can get all the item in the drop-down.
        For (let option of autosuggest)
        {
        const value = await option.textContent()
        console.log(value); //check to see the values
        }
        await.page.waitForTimeout(XXXX);
```

// Code will be used in script for auto suggest:

```
await page.locator(‘Locators’).fill(‘Value’);
        const autoSuggest = await page.$$(“Xpath”)
        For (let option of autosuggest)
        {
        const value = await option.textContent()
        if (value.incudes(‘desired value’))
        {
        await option.click()
        break;
        }
        }
await.page.waitForTimeout(XXXX);
```

// Hidden Items in Drop-Down:

```
await page.locator(‘Locators’).fill(‘Value’);
        await.page.waitForTimeout(XXXX);
        const hiddenItems = await page.$$(“Xpath”)
// Xpath should be selected in such a way that can get all the item in the drop-down.
For (Hidden items)
        For (let option of hiddenItems)
        {
        const hdnItm = await option.textContent()
        if (hdnItm.incudes(‘desired value’))
        {
        await option.click()
        break;
        }
        }
await.page.waitForTimeout(XXXX);
```

Handle Dialogs or Alerts

In Playwright, by default, dialogs are auto dismissed. So, there’s no need to handle them.

Still there are options to handle them.

alert(); confirm(); prompt() dialogs.

// Alert handle with only Ok/Confirm button:

Page.on('dialog', async dialog=>{ //Enabling alert handling

except(dialog.type()).toContain('alert') // Assertion if needed to validate

except(dialog.message()).toContain('Text') // Assertion if needed to validate

await.dialog.accept(); // Only for pop-up ***ok/confirm*** button

)}

await page.click('Locators'); //before clicking on to any pop-up need to enable dialog handler by previous script.

await.page.waitForTimeout(XXXX);

});

// Alert handle with Ok/Confirm & Cancel button:

Page.on('dialog', async dialog=>{ //Enabling alert handling

except(dialog.type()).toContain('Confirm') // Assertion if needed to validate

except(dialog.message()).toContain('Text') // Assertion if needed to validate

await.dialog.accept(); // Only for pop-up ***ok/confirm & cancel*** button

)}

await page.click('Locators'); //before clicking on to any pop-up need to enable dialog handler by previous script. *Here Confirm/Cancel can be performed.*

await except(page.locator('Locators')).toHaveText('Text'); // Assertion if needed to validate

await.page.waitForTimeout(XXXX);

});

// Prompt Handling:

Page.on('dialog', async dialog=>{ //Enabling alert handling

except(dialog.type()).toContain('prompt') // Assertion if needed to validate

except(dialog.message()).toContain('Text') // Assertion if needed to validate

except(dialog.defaultValue()).toContain('Text') // Assertion if needed to validate

await.dialog.accept('text'); //Pass the given value

)}

await page.click(‘Locators’);

Handle Frame/iFrame

//Check total Frames

const allFrames=await page.frames()

console.log(“Number of frames: “, allFrames.length)

//Approach 1: Using name or url

const frame1 = await page.frame({url: ‘target url’)} //Using URL.

fame1.fill(“Locator”,’Text/Value’);

await.page.waitForTimeout(XXXX);

||

const frame2 = await page.frame({name: ‘target name’)} //Using Name

//Approach 2: Using Frame Locator

const inputBox = await page.frameLocator(“Locator of the Frame”).locator(“Locator of the element”)

inputBox.fill(“Value”)

await.page.waitForTimeout(XXXX);

//Nested Frame

const frame = await page.frame({url: ‘target url’)}

const childFrames=await frame.childFrames()

childFrames[index, like:0,1].locator(“Locators”).check()

await.page.waitForTimeout(XXXX);

Handle WebTable/Pagination

```
        const webTable = await page.locator(‘Locators’)
// total number of rows and columns
        const columns = await table.locator(‘locator: like tbody tr’)
        console.log(‘Number of columns:’,await columns.count())
        except(await columns.count()).toBe(Number like:4) // Assertions if validation is
needed
        const rows = await table.locator(‘locator: like thead tr th’)
        console.log(‘Number of rows:’,await rows.count())
        except(await rows.count()).toBe(Number like:5) // Assertions if validation is needed
```

Check-Box on the table:

// For single check

```
const matchedRows.filter({
                has: page.locator(‘Locators’like: ‘td’)
                hasText: ‘desired text from the table’
})
await matchedRows.locator(‘Locators’).check()
await.page.waitForTimeout(XXXX);
```

// For multiple check

```
await selectProduct(rows,page, ‘desired text from the table’ 1) // calling function
await selectProduct(rows,page, ‘desired text from the table’ 2) // calling function
and so on…..
async function selectProduct(rows, page, name)
{
const matchedRows.filter({
                has: page.locator(‘Locators’like: ‘td’)
                hasText: name
```

})

await matchedRows.locator('Locators').check()

}

// Handle Pagination:

    const pages = await page.locator('Locators'like: '.pagination li a')

    console.log('Number of pages:', await pages.count())

Same as previously used 'for loop' and 'if' statement.

<u>Handle Date Picker</u>

// Input field

await page.fill('Locators' , 'Date'like: 02/10/2023)

// Select Date

const year = "Year" // Like: "2024"

const month = "Month" // Like: "June"

const date = "Date" // Like: "21"

await page.click('Locators') //Opens Calender

    while (true)

    {

    const currYear = await page.locator('Locator of Year').textContent()

    const currMonth = await page.locator('Locator of Month').textContent()

    if(currentYear == year && currMonth == month)

    {

    Break;

    }

    await page.locator ('Next Button Locator').click() //Till we get the expected year and month. Also can be used to provide previous button to select previous date

    }

```
        const currDate await page.$$('Locator of all the date value')
        for(const dt of currDate)
        {
        if(await dt.textContent()==date)
        {
        await dt.click();
        break;
        }
        }
```

Mouse Hover Action

```
const variable1 = await page.locator('Locators')
const variable2 = await page.locator('Locators')
// Mouse Hover
        await variable1.hover()
        await variable2.hover()
```

Mouse Right Click Action

```
const button = await page.locator('Locators')
        await button.click({button: 'right'});
```

Mouse Double Click Action

```
const btnCpy= await page.locator('Locators')
        await btnCpy.dblclick();
```

Mouse Drag-Drop Operation

```
// Approach 1
const drag = await page.locator('Locators')
const drop = await page.locator('Locators')
        await drag.hover()
        await page.mouse.down()
        await drop.hover()
        await page.mouse.up()
// Approach 2
        await drag.dragTo(drop)
        await.page.waitForTimeout(XXXX);
```

Keyboard Actions

An example of holding down Shift in order to select and delete some text:

```
await page.keyboard.type('Hello World!');
await page.keyboard.press('ArrowLeft');

await page.keyboard.down('Shift');
for (let i = 0; i < ' World'.length; i++)
  await page.keyboard.press('ArrowLeft');
await page.keyboard.up('Shift');

await page.keyboard.press('Backspace');
// Result text will end up saying 'Hello!'
```

An example of pressing uppercase

```
await page.keyboard.press('Shift+KeyA');
// or
await page.keyboard.press('Shift+A');
```

key can specify the intended keyboardEvent.key value or a single character to generate the text for. A superset of the key values can be found here. Examples of the keys are:F1 - F12, Digit0- Digit9, KeyA- KeyZ, Backquote, Minus, Equal, Backslash, Backspace, Tab, Delete, Escape, ArrowDown, End, Enter, Home, Insert, PageDown, PageUp, ArrowRight, ArrowUp, etc.

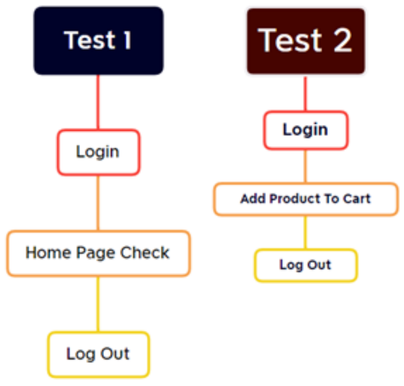
Following modification shortcuts are also supported: Shift, Control, Alt, Meta, ShiftLeft. Holding down Shift will type the text that corresponds to the key in the upper case. If key is a single character, it is case-sensitive, so the values a and A will generate different respective texts. Shortcuts such as key: "Control+o" or key: "Control+Shift+T" are supported as well. When specified with the modifier, modifier is pressed and being held while the subsequent key is being pressed.

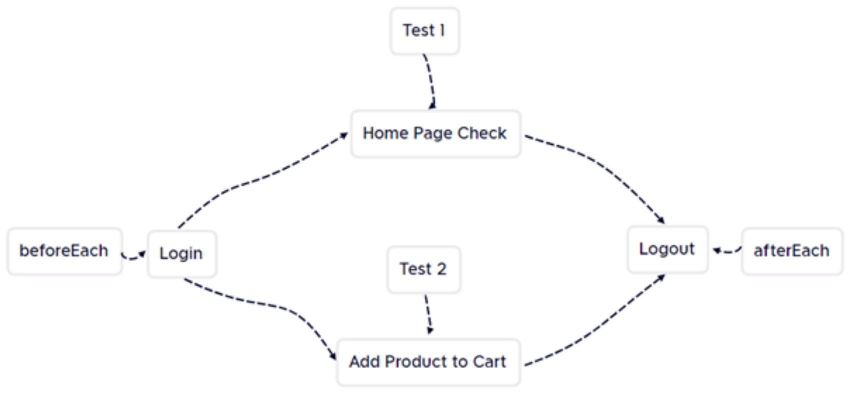
Upload Files

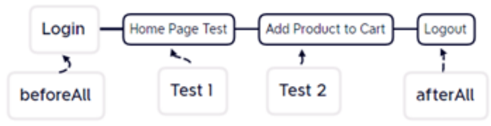
FileChooser objects are dispatched by the page in the page.on('filechooser') event.

```
//Upload Single Files
await page.waitForSlector('Locators');
await page.locators('Locators').click()
await page.locators('Locators').setInputFiles('relative path')
await.page.waitForTimeout(XXXX);

//Upload Multiple Files
await page.locators('Locators')
              .setInputFiles(['Relative path1',' Relative path2']);
await.page.waitForTimeout(XXXX);


//Removing File(s)
await page.locators('Locators')
             .setInputFiles([]);

await.page.waitForTimeout(XXXX);

// Assertion if validation needed

except await page.locators('Locators').toHaveText('Text' Like: No File Selected)
await.page.waitForTimeout(XXXX);
```

Hooks - beforeEach, afterEach, beforeAll & afterAll







Configuration:

playwright.config.js >

```
fullyParallel: false,
```

```
retries: process.env.CI ? 2 : 0,
 /* Opt out of parallel tests on CI. */
 workers: process.env.CI ? 1 : undefined,
```

Example to understand: *(This code may not work)*

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

let page; //global variable

test.beforeEach(async ({ browser }) => { //beforeAll can be used

page = await.browser.newpage(); //creating new page

 //Login

  await page.goto('https://www.saucedemo.com/v1/index.html');

```
  await page.locator('[data-test="username"]').click();
  await page.locator('[data-test="username"]').fill('standard_user');
  await page.locator('[data-test="password"]').click();
  await page.locator('[data-test="password"]').fill('secret_sauce');
  await page.getByRole('button', { name: 'LOGIN' }).click();
}
```

```
import { test, expect } from '@playwright/test';
test('Cart Test', async () => {
//Add to Cart
  await page.locator('div').filter({ hasText: /^\$29\.99ADD TO CART$/
}).getByRole('button').click();
  await page.locator('div').filter({ hasText: /^\$9\.99ADD TO CART$/
}).getByRole('button').click();
});
```

```
 import { test, expect } from '@playwright/test';
 test('Cart Test', async () => {
//Checkout
  await page.getByRole('button', { name: 'ADD TO CART' }).first().click();
  await page.getByRole('link', { name: '3' }).click();
  await page.getByRole('link', { name: 'CHECKOUT' }).click();
  await page.locator('[data-test="firstName"]').click();
  await page.locator('[data-test="firstName"]').fill('edxdds');
  await page.locator('[data-test="lastName"]').click();
  await page.locator('[data-test="lastName"]').fill('csd');
  await page.locator('[data-test="postalCode"]').click();
  await page.locator('[data-test="postalCode"]').fill('sdcsd');
  await page.getByRole('button', { name: 'CONTINUE' }).click();
  await page.getByRole('link', { name: 'FINISH' }).click();
});
```

```
  test.afterEach(async () => { //afterAll can be used
//Logout
  await page.getByRole('button', { name: 'Open Menu' }).click();
  await page.getByRole('link', { name: 'Logout' }).click();
});
```

Grouping Tests

```
test.describe('Group 1',()=>{
test 1
test 2
and so on ………..
})
test.describe('Group 2',()=>{
test 3
test 4
and so on ………..
})
```

Screenshots Capture

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

test('has title', async ({ page }) => {

  await page.screenshot({path:'path'+'filename.png'}) // It will replace the current png file.
```

So we need to save png file in such a way that all the screenshot will be saved by different name.

```
});
```

// Saving png files with different name

```
  await page.screenshot({ path:'path like: test\screenshot'+Date.now()+'Filename.png'}) //
By Date.now() function,  png file will be saved by adding current timestamp
```

// Capturing full page

await page.screenshot({ path:’Path’+Date.now()+’Filename.png’,fullPage:true}) //

// Capturing specific items

await page.locator('Locators’) .screenshot({path:’Path’+Date.now()+’Filename.png’}) //

// Without command, taking screenshots

Go to **playwright.config.js** >

```
use: {
   //headless: false,
   /* 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 */
   trace: 'on-first-retry',
   screenshot: 'on',
 },
```

Add **screenshot: ‘on’,**
This will automatically take screenshot of each command.

screenshot: ‘only-on-failure’ // It will capture if execution fails.

Video Capture/Record

Go to **playwright.config.js** >

```
use: {
   //headless: false,
   /* 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 */
   trace: 'on-first-retry',
   //screenshot: 'on',
   video: "on",
 },
```

This will automatically record on every test.

- off' - Do not record video.
- 'on' - Record video for each test.

- 'retain-on-failure' - Record video for each test, but remove all videos from successful test runs.
- 'on-first-retry' - Record video only when retrying a test for the first time.

Trace Viewer

Go to **playwright.config.js** >

- 'on-first-retry' - Record a trace only when retrying a test for the first time.
- 'on-all-retries' - Record traces for all test retries.
- 'off' - Do not record a trace.
- 'on' - Record a trace for each test. (not recommended as its performance heavy)
- 'retain-on-failure' - Record a trace for each test, but remove it from successful test runs.

Tagging in Playwright

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

test('Test login page @sanity', async ({ page }) => {

  // ...

});

test('Test full report @regression', async ({ page }) => {

  // ...
});

test('Test downloading report @regression', async ({ page }) => {

  // ...
});
```

To run tagged test by terminal:

npx playwright test --grep @regression

Annotations:

- 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.skip('skip this test', async ({ page }) => {

    // This test is not run

  });
  ```

  // Conditionally

  ```
  test('skip this test', async ({ page, browserName }) => {

    test.skip(browserName === 'firefox', 'Still working on it');

  });

  ||

  test('skip this test', async ({ page, browserName }) => {

  if (bowserName == 'chromium')

  {

  test.skip()

  }

  });
  ```

- test.fail() marks the test as failing. Playwright Test will run this test and ensure it does indeed fail. If the test does not fail, Playwright Test will complain.

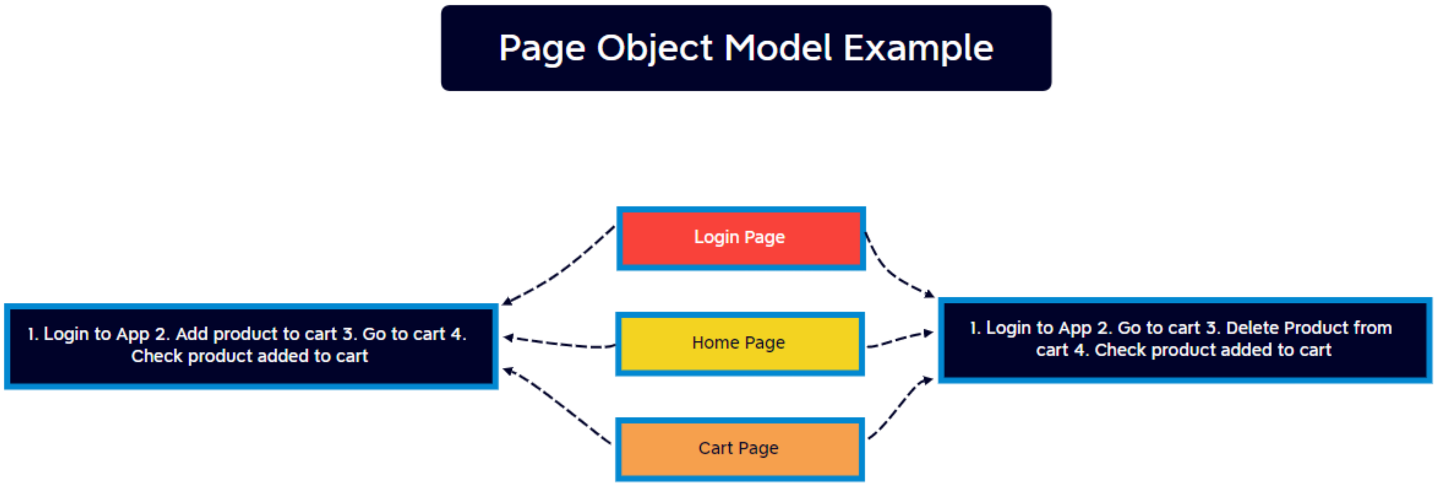
  ```
  test('skip this test', async ({ page, browserName }) => {
  if (bowserName == 'chromium')
  {

  test.fail()

  }
  except(1).toBe(2);      });
  ```

- test.fixme() marks the test as failing. Playwright Test will not run this test, as opposed to the fail annotation. Use fixme when running the test is slow or crashes.
- test.slow() marks the test as slow and triples the test timeout.

//Execute Focused Test

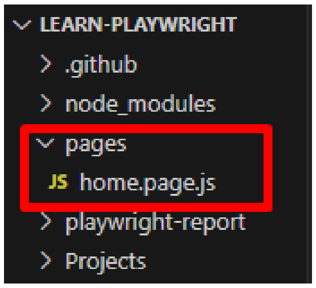
```
test.only('focus this test', async ({ page }) => {

  // Run only focused tests in the entire project.

});
```

Page Object Model in Playwright


Page Object Model Example
Login Page
Home Page
Cart Page
1. Login to App 2. Add product to cart 3. Go to cart 4. Check product added to cart
1. Login to App 2. Go to cart 3. Delete Product from cart 4. Check product added to cart

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

test('has title', async ({ page }) => {

//Login

//Home

//Cart

});
```

- ❖ Need to create a folder under playwright project
- ❖ Under the folder above page object file for each web page
  For Example:



```
LEARN-PLAYWRIGHT
  > .github
  > node_modules
  v pages
      JS home.page.js
  > playwright-report
  > Projects
```

```
//home.page.js

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

export class BrowserstackHomePage {

readonly url ="https://www.browserstack.com/";

readonly page: Page;

readonly browserstackLogo: Locator;

readonly productsMenu: Locator;

readonly productmenudropdown:Locator


constructor(page: Page) {

this.page = page;

this.browserstackLogo = page.locator('#logo');

this.productsMenu = page.locator('#product-menu-toggle');

this.productmenudropdown = page.locator('#product-menu-dropdown >div > ul >li >a
>div[class="dropdown-link-heading"]');

}


async goto(){

await this.page.goto(this.url);

}

async clickOnProducts(){

await this.productsMenu.waitFor({state:"visible"});

await this.productsMenu.click();

}
```

- ❖ Create test file under tests folder

```
// home.test.js

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

import { BrowserstackHomePage } from '../pages/home.page'; //  To create a test, we need to import the page object file.

test('Browserstack homepage verification', async ({ page }) => {

const homepage = new BrowserstackHomePage(page);

await homepage.goto();

await homepage.clickOnProducts();

await expect(homepage.productmenudropdown).toContainText(["Live", "Automate", "Percy", "App Live", "App Automate"])

});
```

Multiple Pages / Windows

```
const {test, except, chromium } = require('@playwright/test');
test ('Handle Pages/Windows', async () => {

          const browser = await chromium.launch
          const context=await browser.newContext()
          const page1 = await context.newPage()
          const page2 = await context.newPage()
          const allPages = context.pages()
          console.log("No of pages created:",allPages.length)

                    await page1.goto ('URL1');
// Two new window will be opened on browser
                    await page2.goto ('URL2');

})
```

// Nevigate one window to another

```
const {test, except, chromium } = require('@playwright/test');
test ('Handle Pages/Windows', async () => {

          const browser = await chromium.launch
          const context=await browser.newContext()
          const page1 = await context.newPage()
                    await page1.goto ('URL1')
                    const pagePromise = context.waitForEvent('page')
                    await page1.locator('Locators').click()

          const newPage = await pagePromise;
          await newPage.locator('Locators').click()
```

Rest API Testing

```
// POST:
const REPO = 'test-repo-1';
const USER = 'github-username';

test('should create a bug report', async ({ request }) => {
  const newIssue = await request.post(`/repos/${USER}/${REPO}/issues`, {
    data: {
      title: '[Bug] report 1',
      body: 'Bug description',
    }
  });});
// GET
test("Get users", async ({ request, baseURL }) => {
  const _response = await request.get(`${baseURL}public/v2/users/`);
  expect(_response.ok()).toBeTruthy();
  expect(_response.status()).toBe(200);
  console.log(await _response.json());
});
//  Passing query parameters
test("Get one user", async ({ request, baseURL }) => {
  const _response = await request.get(`${baseURL}public/v2/users/`, {
    params: {
      id: 5229,
    },
  });
  expect(_response.ok()).toBeTruthy();
  expect(_response.status()).toBe(200);
  console.log(await _response.json());
});
// Updating a resource (PUT).
test("Update a user", async ({ request, baseURL }) => {
  const response = await request.put(`${baseURL}public/v2/users/5721`, {
    data: {
      name: "Zambo",
    },
  });
  expect(response.ok()).toBeTruthy();
  expect(response.status()).toBe(200);
  console.log(await response.json());
});
//  Deleting a resource
test("Delete a user", async ({ request, baseURL }) => {
  const response = await request.delete(`${baseURL}public/v2/users/5217`);
headers: {
      'Accept': 'application/vnd.github.v3+json',
      // Add GitHub personal access token.
      'Authorization': `token ${process.env.API_TOKEN}`,
    }
  expect(response.ok()).toBeTruthy();
  expect(response.status()).toBe(204);
});
```

Thank you