**Slide 1: Session Title**

* **Title:** Web Automation using Playwright
* **Subtitle:** Assertions, Window Handling, Playwright Inspector
* **Time:** 45 Minutes
* **Goals:** Learn advanced automation techniques using Playwright.

**Slide 2: Agenda**

1. Assertions in Playwright
2. Window Handling
3. Playwright Inspector
4. Demo for each topic

**Slide 3: Assertions in Playwright**

* **Definition:** Assertions validate the expected behaviour of a web application during a test.
* **Types of Assertions in Playwright:**
  + expect from Playwright Test Library
  + Common Assertions: toBe, toContain, toEqual
  + Element Assertions: toBeVisible, toBeEnabled, toHaveText

**Slide 4: Assertions - Examples**

**Example Code:**

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

test('Validate element visibility', async ({ page }) => {

await page.goto('https://example.com');

const element = page.locator('#elementId');

await expect(element).toBeVisible();

});

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

await page.goto('https://example.com');

await expect(page).toHaveTitle('Example Domain');

});

* **Tips:**
  + Use specific locators for stable assertions.
  + Validate both content and element states.

**Slide 5: Talking Points - Assertions**

* Discuss the importance of assertions for test validation.
* Highlight differences between soft and hard assertions.
* Emphasise error messages generated during assertion failures.

**Slide 6: Demo - Assertions**

* **Demo Plan:**
  + Test if the login button is visible.
  + Check if the page title updates after navigation.
* **Execution:**
  + Run tests with both passing and failing assertions.
  + Show error messages and debugging techniques.

**Slide 7: Window Handling**

* **Definition:** Manage multiple browser windows or tabs during automation.
* **Key Functions:**
  + context.newPage()
  + page.waitForEvent('popup')
  + page.close()
* **Common Scenarios:**
  + Handling authentication popups
  + Navigating through child windows

**Slide 8: Window Handling - Examples**

**Example Code:**

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

test('Handle multiple tabs', async ({ browser }) => {

const context = await browser.newContext();

const page = await context.newPage();

await page.goto('https://example.com');

// Trigger a new tab

const [newPage] = await Promise.all([

context.waitForEvent('page'),

page.click('a[target=\_blank]'),

]);

await newPage.waitForLoadState();

console.log(await newPage.title());

// Close the new tab

await newPage.close();

});

**Slide 9: Talking Points - Window Handling**

* Explain the concept of browser contexts and isolation.
* Describe how to synchronise actions with event listeners.
* Discuss use cases such as OAuth popups and tab navigation.

**Slide 10: Demo - Window Handling**

* **Demo Plan:**
  + Automate a link that opens a new tab.
  + Perform actions on the new tab.
  + Return to the original tab and verify state.
* **Execution:**
  + Run code step by step.
  + Debug synchronisation issues and discuss best practices.

**Slide 11: Playwright Inspector**

* **Definition:** A debugging tool to pause and inspect tests interactively.
* **Features:**
  + Step-by-step execution
  + Locators highlighting
  + Code generation
* **Usage:** Run tests with the PWDEBUG=1 environment variable.

**Slide 12: Playwright Inspector - Example**

**Example Command:**

PWDEBUG=1 npx playwright test

**Steps:**

1. Launch the inspector.
2. Pause the test at any step.
3. Inspect DOM elements and test behaviour.

**Slide 13: Talking Points - Playwright Inspector**

* Highlight the benefits of interactive debugging.
* Explain how it helps in identifying flaky tests.
* Discuss its role in generating robust selectors.

**Slide 14: Demo - Playwright Inspector**

* **Demo Plan:**
  + Debug a failing test.
  + Use the inspector to locate issues.
  + Modify the test in real time.
* **Execution:**
  + Showcase step execution.
  + Demonstrate locator generation and verification.

**Slide 15: Summary**

* **Key Learnings:**
  + Assertions ensure test reliability.
  + Window handling manages complex workflows.
  + Playwright Inspector simplifies debugging.
* **Next Steps:** Practise the examples and explore advanced topics.

**Slide 16: Q&A**

* Open the floor for questions.
* Provide clarifications and additional examples if needed.