How to Handle Multiple Windows or Pages in Playwright

Introduction

To handle multiple windows in Playwright, we can use **browser context**.
A **browser context** is created using a browser instance and inside the browser context, we can create **multiple pages**.

Using Page Fixture

When we write a new test in Playwright, we usually pass the page fixture.
This page fixture is already provided by Playwright and represents a browser with a page.

- This default page fixture helps in writing simple test cases.
- However, if we want to handle **multiple browsers**, **multiple windows** or **multiple pages**, we need to **create our own page** using **browser context**.

How to Create Our Own Page Using Browser Context

Step-by-Step:

1. **Import chromium** from @playwright/test instead of importing page, because we are going to create our own page using chromium.
2. **Launch the browser** using chromium:

   ```
   const browser = await chromium.launch();
   ```
3. **Create a new context** from the browser:

   ```
   const context = await browser.newContext();
   ```
4. **Create a page** inside the context:

   ```
   const page = await context.newPage();
   ```

So here:

- **Browser** contains the **context**
- **Context** contains **multiple pages**

Note

If you want to use the existing page, you can simply use the page fixture in your test and write your code with it.

Example Code with Page Fixture

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

test("Page with existing page fixture", async ({ page }) => {

  await page.goto("https://...");

});
```

Example Code Without Page Fixture (Creating Our Own Page)

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

test("Creating our own page", async () => {

  const browser = await chromium.launch();

  const context = await browser.newContext();

  const page = await context.newPage();

  await page.goto("https://...");

});
```

Creating Multiple Pages in Same Context

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

test("Creating multiple pages", async () => {

  const browser = await chromium.launch();

  const context = await browser.newContext();

  const pageOne = await context.newPage();

  const pageTwo = await context.newPage();

});
```

Check How Many Pages Are Created

You can use the pages() function to get the list of all created pages:

```
const allPages = context.pages();

console.log("No of pages created:", allPages.length);
```

Note

Even though both pages are created under the same context, they are **independent**.

You can open **two different applications** on each page:

```
await pageOne.goto("https://instagram.com");

expect(pageOne).toHaveTitle("Instagram");

await pageTwo.goto("https://whatsapp.com");

expect(pageTwo).toHaveTitle("Windows");
```

- These are two **independent pages**.
- We are **not navigating** from one page to another.

How to Navigate from One Page to Another Page in Playwright

Understanding the Difference

There's a difference between:

- **Opening multiple independent pages**, and
- **Navigating from one page to another** because of a user action (like clicking a link or button).

Opening Multiple Pages

- In the previous example, we saw how to create multiple pages using browserContext.
- Each page was created manually and works **independently**.
- We can perform different actions on each of them.

Navigating to Another Page

- Sometimes, an **action on one page (like a click)** will **open another page**.
- In this case, there's a **link or relation** between the first and the second page.
- This is considered **navigating** from one page to another.

Handling Linked Pages (Triggering a New Page)

When we click a button or link that opens a new page, it **triggers an event**.

To handle this scenario in Playwright:

Step-by-Step:

1. Use the waitForEvent function from the **context** to wait for the new page event.

```
const pagePromise = context.waitForEvent('page');
```

This creates an empty page and waits for the 'page' event to be triggered.

2. Then, perform the action that **triggers the new page**, like clicking a link:

```
await page.locator("//a[@id='newPageWillOpen']").click();
```

3. After the click, the new page will be captured in pagePromise. So, **assign it to a new variable**:

```
const newPage = await pagePromise;
```

4. You can now perform actions on newPage, such as verifying the title:

```
await expect(newPage).toHaveTitle("New page title");
```

Full Example

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

test("Handling multiple pages", async () => {

 const browser  = await chromium.launch();

 const context = await browser.newContext();

 const page = await context.newPage();

 await page.goto("https://LukesCafeMainMenuPage");

 await expect(page).toHaveTitle("Luke’s Cafe Main Menu Page");
```

```
  // Before clicking the link, wait for the new page event
  const pagePromise = context.waitForEvent('page');
  // Click the link that opens the coffee list page
  await page.locator("//a[@id='newPageCoffeeListed']").click();
  // pagePromise now holds the new page
  const newPage = await pagePromise;
  // Validate the title of the new page
  await expect(newPage).toHaveTitle("Coffee list in Luke’s cafe");
  // Close the new page only (not the main menu page)
  await newPage.close();
});
```

Important Note

Unlike Selenium, where we manually **switch to a new window or tab**, in Playwright:

- We **create multiple pages using context**.
- We use waitForEvent('page') to **wait for the new page** to be created after a triggering action.
- This gives us control to decide **which page to use or close**, without needing to switch back and forth.

Summary

- Use browser.newContext() and context.newPage() to create **multiple independent pages**.
- Use context.waitForEvent('page') to handle **new pages opened by user actions** like clicking a link.
- **Independent pages** work separately.
- **Triggered pages** are connected to the main page and require event handling.