13.Page Objects

**Pages in Playwright**

1. **Definition**:
   * A **Page** represents a single tab or popup window within a browser context.
   * Used for navigation and interaction with page content.
2. **Creating and Navigating a Page**:

javascript

Copy code

const page = await context.newPage();

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

* + **Input Interaction**:

javascript

Copy code

await page.locator('#search').fill('query');

* + **Clicking Links**:

javascript

Copy code

await page.locator('#submit').click();

console.log(page.url()); // Outputs the new URL

**Multiple Pages**

* Each **BrowserContext** can host multiple pages (tabs).
* Pages inherit context-level configurations, like viewport size and network routes.

1. **Creating Multiple Pages**:

javascript

Copy code

const pageOne = await context.newPage();

const pageTwo = await context.newPage();

1. **Retrieve All Pages in Context**:

javascript

Copy code

const allPages = context.pages();

**Handling New Pages**

1. **Using context.on('page') for new tabs**:
   * **Example**: Handling links with target="\_blank".

javascript

Copy code

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

await page.getByText('open new tab').click();

const newPage = await pagePromise;

await newPage.getByRole('button').click();

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

1. **Listening for All New Pages**:

javascript

Copy code

context.on('page', async page => {

await page.waitForLoadState();

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

});

**Handling Popups**

1. **Listening for Popup Events**:

javascript

Copy code

const popupPromise = page.waitForEvent('popup');

await page.getByText('open the popup').click();

const popup = await popupPromise;

await popup.getByRole('button').click();

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

1. **Generic Popup Handling**:

javascript

Copy code

page.on('popup', async popup => {

await popup.waitForLoadState();

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

});

**Page Object Models (POM)**

1. **Introduction**:
   * POM is a design pattern for structuring large test suites.
   * Encapsulates page elements and functionalities into reusable classes.
2. **Advantages**:
   * Simplifies test authoring and maintenance.
   * Encapsulates selectors in one place for easier updates.

**Implementation of POM**

1. **PlaywrightDevPage Class**:

javascript

Copy code

const { expect } = require('@playwright/test');

exports.PlaywrightDevPage = class PlaywrightDevPage {

constructor(page) {

this.page = page;

this.getStartedLink = page.locator('a', { hasText: 'Get started' });

this.gettingStartedHeader = page.locator('h1', { hasText: 'Installation' });

this.pomLink = page.locator('li', { hasText: 'Guides' })

.locator('a', { hasText: 'Page Object Model' });

this.tocList = page.locator('article div.markdown ul > li > a');

}

async goto() {

await this.page.goto('https://playwright.dev');

}

async getStarted() {

await this.getStartedLink.first().click();

await expect(this.gettingStartedHeader).toBeVisible();

}

async pageObjectModel() {

await this.getStarted();

await this.pomLink.click();

}

};

**Using the PlaywrightDevPage Class**

1. **Test: Verifying Table of Contents**:

javascript

Copy code

const { test, expect } = require('@playwright/test');

const { PlaywrightDevPage } = require('./playwright-dev-page');

test('getting started should contain table of contents', async ({ page }) => {

const playwrightDev = new PlaywrightDevPage(page);

await playwrightDev.goto();

await playwrightDev.getStarted();

await expect(playwrightDev.tocList).toHaveText([

`How to install Playwright`,

`What's Installed`,

`How to run the example test`,

`How to open the HTML test report`,

`Write tests using web first assertions, page fixtures and locators`,

`Run single test, multiple tests, headed mode`,

`Generate tests with Codegen`,

`See a trace of your tests`

]);

});

1. **Test: Page Object Model Article**:

javascript

Copy code

test('should show Page Object Model article', async ({ page }) => {

const playwrightDev = new PlaywrightDevPage(page);

await playwrightDev.goto();

await playwrightDev.pageObjectModel();

await expect(page.locator('article')).toContainText('Page Object Model is a common pattern');

});

**Key Takeaways**

* Use POM for structured, reusable, and maintainable test suites.
* Pages and contexts can handle multiple tabs and popups efficiently.
* Use events like page or popup for dynamic page handling.