**Playwright**

**Playwright is an open-source automation framework developed by Microsoft, designed for end-to-end testing of web applications. It supports multiple programming languages and enables automation for all modern browsers. Below are detailed notes:**

**1. Key Features**

* **Cross-Browser Support: Automates Chromium (Google Chrome, Microsoft Edge), WebKit (Safari), and Firefox.**
* **Multi-Language Support: Works with JavaScript, TypeScript, Python, Java, and C#.**
* **Headless/Headful Mode: Can run tests in both headless and headful browser modes.**
* **Auto-Wait Mechanism: Automatically waits for elements to be ready before performing actions.**
* **Multi-Tab and Frame Support: Easily handles complex scenarios involving multiple tabs or frames.**
* **Native Mobile Emulation: Supports testing on mobile emulators with touch support.**
* **Parallel Testing: Runs tests concurrently to reduce execution time.**
* **Network Interception: Allows interception and modification of network requests and responses.**
* **Visual Comparisons: Supports screenshot testing and comparison for visual regressions.**

**Playwright with javaScript - Installation**

**1. Prerequisites**

* **Ensure you have Node.js (v14 or later) installed on your machine. You can download it from** [**Node.js Official Website**](https://nodejs.org/)**.**
* **VS code**
* **Create project folder – open in VS code**
* **Install playwright using terminal(cmd) = npm init playwright@latest**
* **Check the playwright version in terminal (cmd) = npm playwright -v**
* **Package.json = node project management file**
* **Playwright.config.js = playwright configuration**

**2. Install playwright using VS code extension**

* **Create project folder – open in VS code**
* **Click on extension icon in VS code and enter a playwright name in search bar and click on install**
* **Click on project that we have created and Click on view and select “Command Palette” and in search bar enter playwright and select “Test: install playwright” and select checkbox as a javaScript and click on OK**

**3. Run the playwright test**

* **npx playwright test = Runs all tests**
* **npx playwright test --headed = Run all test in headed mode (Open the browser)**
* **npx playwright test mytest.spec.js --project=chromium -- headed =Run specific test in chrome browser**

**4. Test Case/Script Format**

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

**test('Home page', async({page})=>{**

**await page.goto('https://testautomationpractice.blogspot.com/');**

**const pagetitle= await page.title();**

**console.log(pagetitle);**

**await expect(page).toHaveTitle('Automation Testing Practice');**

**})**

**5. Locating Elements in playwright**

**Locating elements in Playwright involves using selectors to identify elements on a web page. Playwright provides various ways to locate elements, allowing flexibility based on your application's structure and requirements. Below are the methods to locate elements in Playwright:**

**1. Using Selectors**

**Playwright supports multiple types of selectors for locating elements.**

**a. CSS Selectors (Most Common)**

**CSS selectors are used to locate elements based on their styles, IDs, or classes.**

**javascript**

**Copy code**

**await page.click('button#submit'); // ID selector**

**await page.click('button.submit-btn'); // Class selector**

**await page.click('div.container > button'); // Hierarchical selector**

**b. Text Selectors**

**Locate elements by visible text.**

**javascript**

**Copy code**

**await page.click('text="Submit"'); // Exact match**

**await page.click('text=Submit'); // Partial match**

**c. XPath Selectors**

**Use XPath expressions to locate elements.**

**javascript**

**Copy code**

**await page.click('//button[@id="submit"]'); // Locate button by ID**

**await page.click('//div[@class="container"]//button'); // Hierarchical XPath**

**d. Role Selectors (ARIA Roles)**

**For accessibility testing, locate elements by roles.**

**javascript**

**Copy code**

**await page.click('role=button[name="Submit"]');**

**e. Attribute Selectors**

**Use custom attributes or properties.**

**javascript**

**Copy code**

**await page.click('[data-testid="login-button"]'); // Custom attribute**

**await page.click('[placeholder="Search"]'); // Placeholder attribute**

**2. Combining Selectors**

**You can combine multiple selector types for more precise targeting.**

**javascript**

**Copy code**

**await page.click('button#submit[data-enabled="true"]');**

**3. Using Locator API**

**The Locator API provides additional methods to interact with elements.**

**javascript**

**Copy code**

**const button = page.locator('button#submit');**

**await button.click(); // Perform actions**

**await button.hover(); // Hover over the element**

**4. Handling Multiple Elements**

**a. First Matching Element**

**javascript**

**Copy code**

**await page.locator('button').first().click();**

**b. Last Matching Element**

**javascript**

**Copy code**

**await page.locator('button').last().click();**

**c. Specific Index**

**javascript**

**Copy code**

**await page.locator('button').nth(2).click(); // Third button (index starts at 0)**

**d. Iterating Through Elements**

**javascript**

**Copy code**

**const buttons = await page.locator('button').all();**

**for (const button of buttons) {**

**console.log(await button.textContent());**

**}**

**5. Using Text Content**

**a. Exact Match**

**javascript**

**Copy code**

**await page.click('text="Click Me"');**

**b. Partial Match**

**javascript**

**Copy code**

**await page.click('text=Click');**

**6. Advanced Techniques**

**a. Parent and Child Selectors**

**javascript**

**Copy code**

**await page.locator('div.container >> button').click();**

**b. Using Regular Expressions**

**javascript**

**Copy code**

**await page.click('text=/.\*Submit.\*/');**

**c. Chained Locators**

**javascript**

**Copy code**

**await page.locator('div.container').locator('button').click();**

**7. Waiting for Elements**

**To ensure the element is available before interacting with it:**

**javascript**

**Copy code**

**await page.waitForSelector('button#submit');**

**await page.click('button#submit');**

**8. Debugging Selectors**

**a. Highlight Elements**

**Use the highlight option in the locator() method:**

**javascript**

**Copy code**

**await page.locator('button#submit', { highlight: true });**

**b. Test Selectors in the Browser**

**Open the Playwright Inspector:**

**bash**

**Copy code**

**npx playwright codegen**

**This tool allows you to generate and test selectors interactively.**

**9. Common Issues and Tips**

* **Dynamic Elements: If elements take time to load, use waitForSelector or Playwright’s auto-waiting features.**
* **Frames/IFrames: Use frameLocator to locate elements within frames.**

**javascript**

**Copy code**

**const frame = page.frameLocator('iframe#myFrame');**

**await frame.locator('button').click();**

* **Shadow DOM: Playwright supports Shadow DOM elements.**

**javascript**

**Copy code**

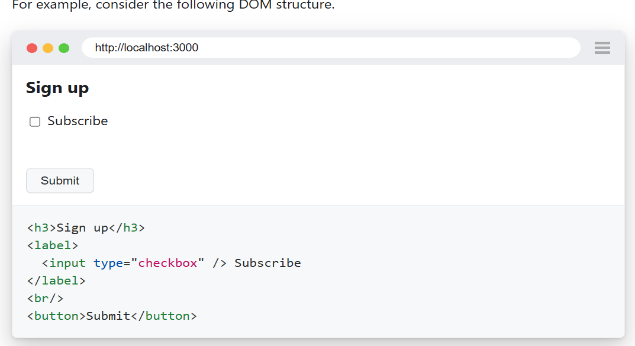
**await page.locator('css=custom-element >> text="Submit"').click();**

**6. Built-in Locators in playwright**

**These are the recommended built-in locators.**

* [**page.getByRole()**](https://playwright.dev/docs/locators#locate-by-role)**= to locate by explicit and implicit accessibility attributes.**
* [**page.getByText()**](https://playwright.dev/docs/locators#locate-by-text)**= to locate by text content.**
* [**page.getByLabel()**](https://playwright.dev/docs/locators#locate-by-label)**= to locate a form control by associated label's text.**
* [**page.getByPlaceholder()**](https://playwright.dev/docs/locators#locate-by-placeholder)**= to locate an input by placeholder.**
* [**page.getByAltText()**](https://playwright.dev/docs/locators#locate-by-alt-text)**= to locate an element, usually image, by its text alternative.**
* [**page.getByTitle()**](https://playwright.dev/docs/locators#locate-by-title)**= to locate an element by its title attribute.**
* [**page.getByTestId()**](https://playwright.dev/docs/locators#locate-by-test-id)**= to locate an element based on its data-testid attribute (other attributes can be configured).**

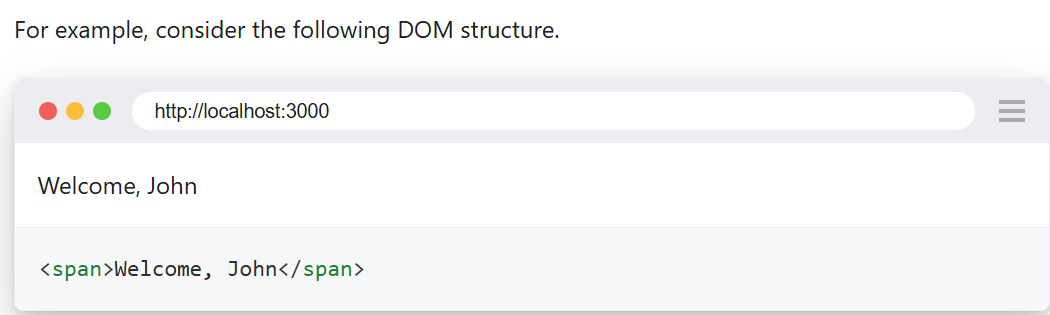
1. [**page.getByRole()**](https://playwright.dev/docs/locators#locate-by-role) **=**

****

**You can locate each element by its implicit role:**

**await expect(page.getByRole('heading', { name: 'Sign up' })).toBeVisible();  
  
await page.getByRole('checkbox', { name: 'Subscribe' }).check();  
  
await page.getByRole('button', { name: /submit/i }).click();**

1. [**page.getByText()**](https://playwright.dev/docs/locators#locate-by-text)**=**

****

**You can locate the element by the text it contains:**

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

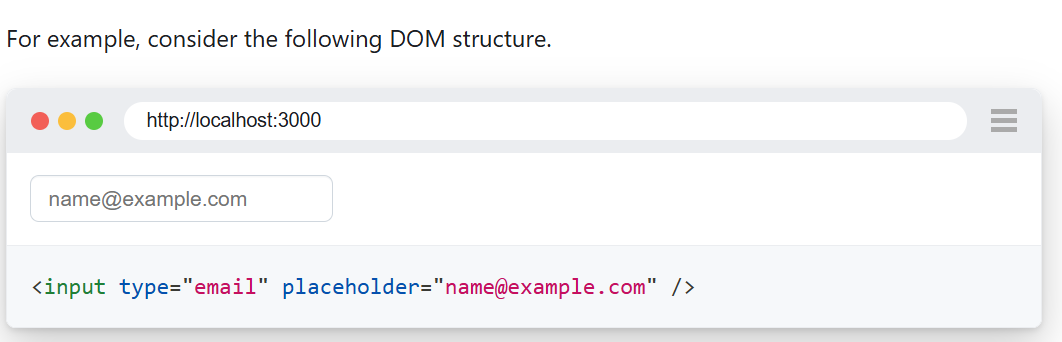
1. [**page.getByLabel()**](https://playwright.dev/docs/locators#locate-by-label)**=**

****

**You can fill the input after locating it by the label text:**

**await page.getByLabel('Password').fill('secret');**

1. [**page.getByPlaceholder()**](https://playwright.dev/docs/locators#locate-by-placeholder) **=**

****

**You can fill the input after locating it by the placeholder text:**

**await page  
 .getByPlaceholder('name@example.com')  
 .fill('playwright@microsoft.com');**

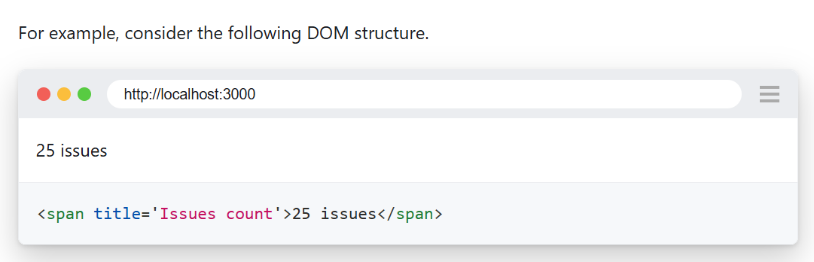
1. [**page.getByAltText()**](https://playwright.dev/docs/locators#locate-by-alt-text) **=**

****

**You can click on the image after locating it by the text alternative:**

**await page.getByAltText('playwright logo').click();**

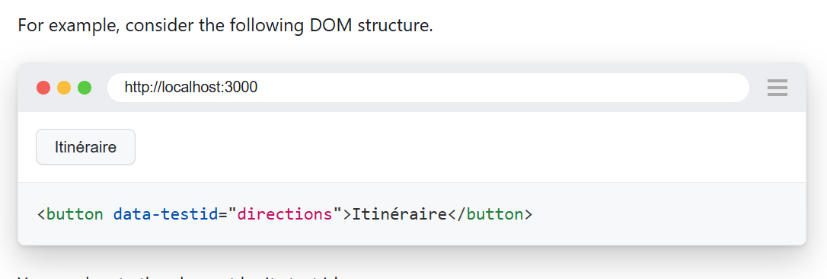
1. [**page.getByTitle()**](https://playwright.dev/docs/locators#locate-by-title) **=**

****

**You can check the issues count after locating it by the title text:**

**await expect(page.getByTitle('Issues count')).toHaveText('25 issues');**

1. [**page.getByTestId()**](https://playwright.dev/docs/locators#locate-by-test-id) **=**

****

**You can locate the element by its test id:**

**await page.getByTestId('directions').click();**

**Assertion**

**Assertions in Playwright are used to verify that a web application behaves as expected during testing. They ensure that the state of elements, the application, or outputs meet predefined conditions. Assertions are a critical part of Playwright tests to validate functionality and reliability.**

**Key Features of Assertions in Playwright:**

1. **Built-in Assertions: Playwright provides powerful and flexible assertions through the expect function.**
2. **Automatic Retries: Playwright retries assertions until they pass or the timeout is reached. This accounts for dynamic web applications where the state may take some time to stabilize.**
3. **Enhanced Debugging: When assertions fail, Playwright provides detailed error messages and snapshots to help debug the failure.**

**Common Assertion Types:**

**1. Element State Assertions:**

**These are used to validate the state of web elements, such as visibility, enabled/disabled status, or selection.**

* **Check Visibility:**

**javascript**

**Copy code**

**const element = page.locator('#button');**

**await expect(element).toBeVisible();**

**Ensures the element is visible on the page.**

* **Check Hidden State:**

**javascript**

**Copy code**

**const element = page.locator('#loading-indicator');**

**await expect(element).toBeHidden();**

* **Check Enabled State:**

**javascript**

**Copy code**

**const element = page.locator('#submit');**

**await expect(element).toBeEnabled();**

* **Check Disabled State:**

**javascript**

**Copy code**

**const element = page.locator('#cancel');**

**await expect(element).toBeDisabled();**

**2. Text Content Assertions:**

**Used to verify the text displayed by an element.**

* **Exact Match:**

**javascript**

**Copy code**

**const element = page.locator('#message');**

**await expect(element).toHaveText('Hello, World!');**

* **Partial Match:**

**javascript**

**Copy code**

**const element = page.locator('#message');**

**await expect(element).toContainText('Hello');**

**3. Attribute Assertions:**

**Validate the value of an element's attributes.**

* **Check Attribute Value:**

**javascript**

**Copy code**

**const element = page.locator('#image');**

**await expect(element).toHaveAttribute('src', '/images/logo.png');**

**4. CSS Class Assertions:**

**Verify if an element has a specific CSS class.**

**javascript**

**Copy code**

**const element = page.locator('#button');**

**await expect(element).toHaveClass(/active/);**

**5. URL Assertions:**

**Used to validate the current page's URL.**

* **Exact URL:**

**javascript**

**Copy code**

**await expect(page).toHaveURL('https://example.com/dashboard');**

* **Partial URL:**

**javascript**

**Copy code**

**await expect(page).toHaveURL(/dashboard/);**

**6. Element Count Assertions:**

**Ensures a specific number of elements are present.**

**javascript**

**Copy code**

**const items = page.locator('.list-item');**

**await expect(items).toHaveCount(5);**

**7. Snapshot Assertions:**

**Compare the current state of an element with a previously saved snapshot.**

**javascript**

**Copy code**

**await expect(page).toMatchSnapshot('screenshot.png');**

**Assertion Syntax:**

**The syntax for assertions in Playwright follows this pattern:**

**javascript**

**Copy code**

**await expect(locator).toXxx(...);**

* **locator: Specifies the element or object to be asserted.**
* **toXxx(): Represents the assertion method (e.g., toBeVisible, toHaveText).**

**Example Test with Assertions:**

**Here’s an example of a Playwright test that uses multiple assertions:**

**javascript**

**Copy code**

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

**test('Test Login Page', async ({ page }) => {**

**// Navigate to the login page**

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

**// Assert the page title**

**await expect(page).toHaveTitle('Login - Example');**

**// Assert that the username field is visible**

**const usernameField = page.locator('#username');**

**await expect(usernameField).toBeVisible();**

**// Assert the login button is disabled initially**

**const loginButton = page.locator('#login');**

**await expect(loginButton).toBeDisabled();**

**// Fill in the form**

**await usernameField.fill('testuser');**

**await page.locator('#password').fill('password123');**

**// Assert the login button is enabled after filling the form**

**await expect(loginButton).toBeEnabled();**

**// Click the login button and assert redirection**

**await loginButton.click();**

**await expect(page).toHaveURL('https://example.com/dashboard');**

**});**

**Tips for Using Assertions in Playwright:**

1. **Leverage Auto-Retry: Playwright automatically retries assertions until the timeout is reached, making tests more stable.**
2. **Use Descriptive Assertions: Write assertions that clearly describe what is being validated to improve test readability.**
3. **Combine Assertions: Validate multiple aspects of a feature within a single test to ensure comprehensive coverage.**
4. **Handle Dynamic Content: Use assertions like toBeVisible or toHaveText for content that changes dynamically.**

**SoftAssertion**

**Soft assertions allow tests to continue execution even if an assertion fails. Unlike hard assertions, where a failure stops the test immediately, soft assertions collect all assertion failures and report them at the end of the test. This approach is useful for validating multiple conditions without halting the test at the first failure.**

**How Assertions Work in Playwright**

**Playwright primarily uses expect from the @playwright/test module for assertions. By default, these are hard assertions. For soft assertions, you need to handle assertions in a way that allows the test to continue executing.**

**Implementing Soft Assertions in Playwright**

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

**test('SoftAssertion', async({page})=>{**

**await page.goto('https://opensource-demo.orangehrmlive.com/web/index.php/auth/login')**

**await expect.soft(page).toHaveURL('https://opensource-demo.orangehrmlive.com/web/index.php/auth/login');**

**await expect.soft(page).toHaveTitle('OrangeH');**

**const login=await page.locator("//button[text()=' Login ']")**

**await expect.soft(login).toBeVisible();**

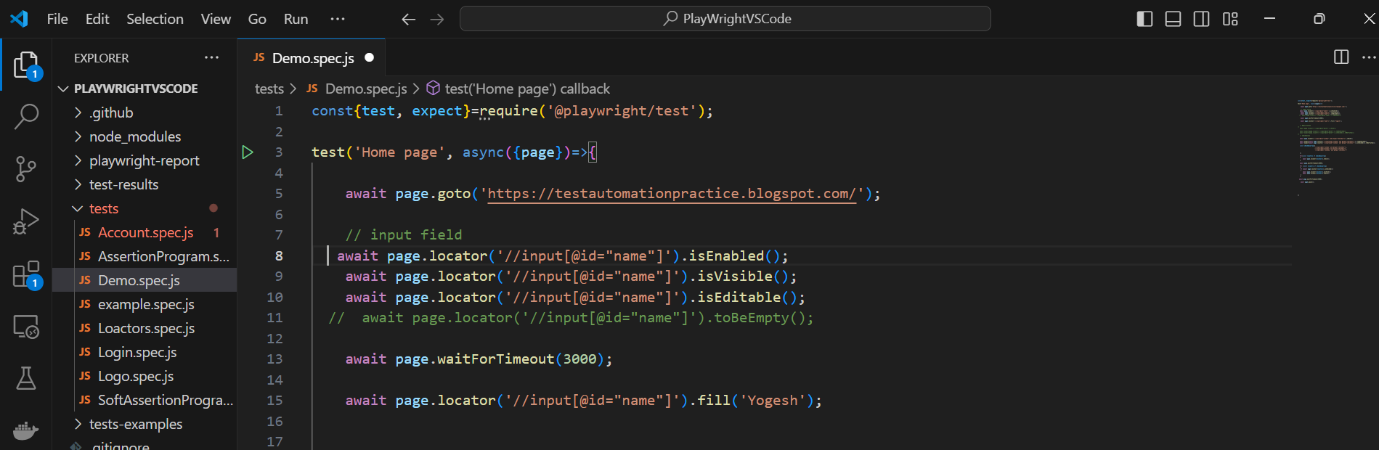
**await page.pause();**

**})**

**Handling Input Box & Radio Buttons & Checkboxses**

**1. Handling Input Boxes**

**Input boxes are handled using the fill() or type() methods to enter text.**

****

**2. Handling Radio Buttons**

**To handle radio buttons, you can use the check() method or directly click on the radio button.**

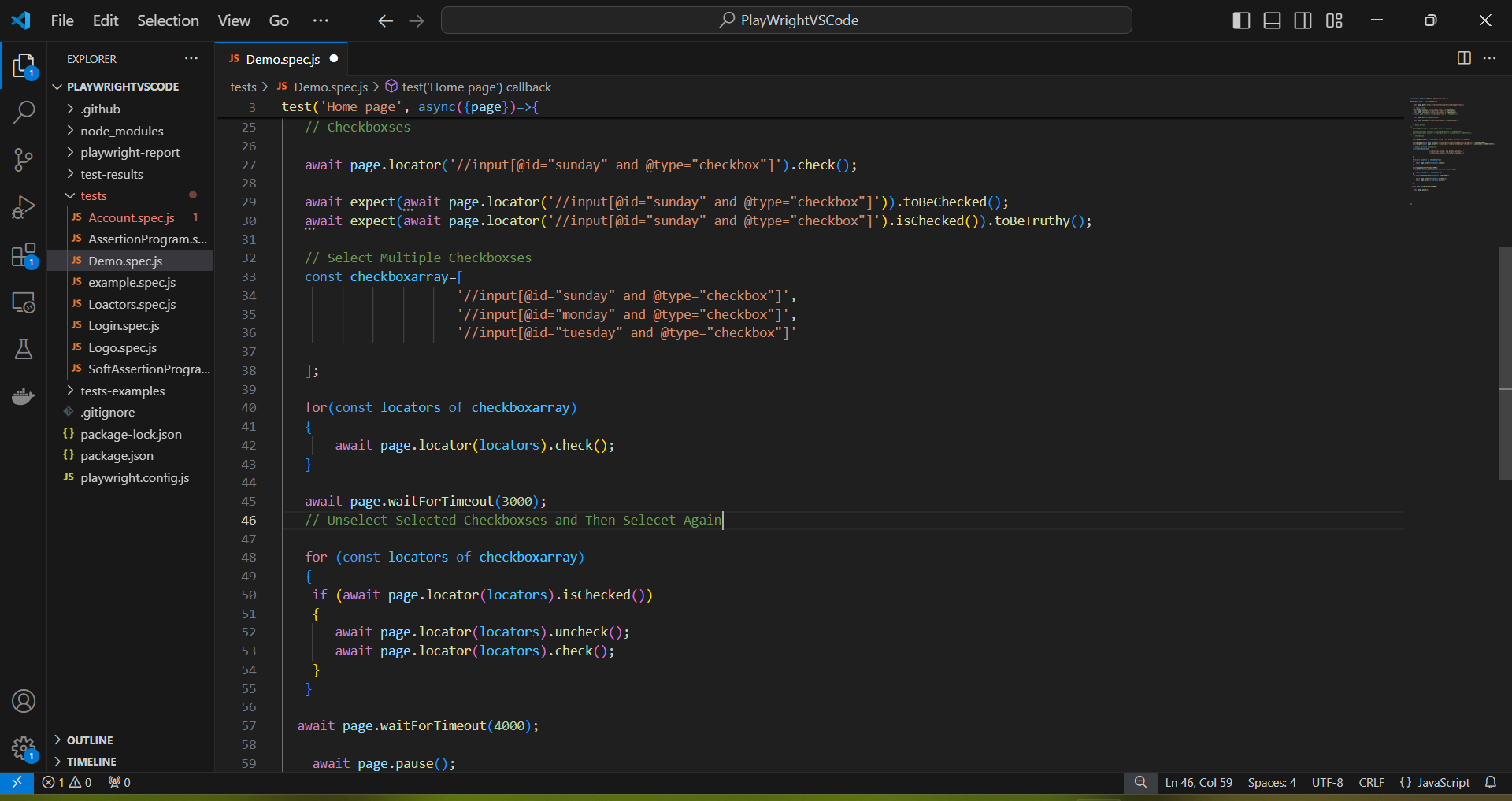
**await page.locator('//input[@id="male"]').check();**

**await expect(page.locator('//input[@id="male"]')).toBeChecked();**

**await expect(page.locator('//input[@id="male"]').isChecked()).toBeTruthy();**

**3. Handling Checkboxes**

**Checkboxes are handled using the check() and uncheck() methods.**

****

**Key Methods:**

* **check(selector): Checks the checkbox.**
* **uncheck(selector): Unchecks the checkbox.**
* **isChecked(selector): Checks if the checkbox is selected.**

**Handling Drop-Down**

**Handling dropdowns in Playwright involves selecting options from a dropdown element in web applications. Dropdowns are typically implemented using <select> elements or custom HTML structures styled to look like dropdowns. Playwright provides straightforward methods to handle both types of dropdowns. Here’s a detailed explanation:**

**1. Handling Standard <select> Dropdowns**

**For standard dropdowns created using the <select> element, Playwright provides the selectOption() method. This method allows you to select an option by its value, label, or index.**

**Syntax**

**javascript**

**Copy code**

**page.locator('selector').selectOption(option);**

**Parameters**

* **Value: The value attribute of the <option> element.**
* **Label: The visible text of the <option> element.**
* **Index: The zero-based index of the <option> element.**

**Examples**

**Select by Value**

**javascript**

**Copy code**

**await page.locator('select#dropdownId').selectOption('optionValue');**

**Select by Label**

**javascript**

**Copy code**

**await page.locator('select#dropdownId').selectOption({ label: 'Option Label' });**

**Select by Index**

**javascript**

**Copy code**

**await page.locator('select#dropdownId').selectOption({ index: 2 });**

**Select Multiple Options (for <select multiple> elements)**

**javascript**

**Copy code**

**await page.locator('select#multiSelect').selectOption(['value1', 'value2']);**

**Check particular value is present in drop down or not**

**const options= await page.$$('#country option')**

**let Status=false**

**for(const option of options)**

**{**

**// console.log(await option.textContent());**

**let value = await option.textContent();**

**if(value.includes('India'))**

**{**

**Status=true**

**break;**

**}**

**}**

**expect(Status).toBeTruthy();**

**Programming Example Of Multi-Select Drop Down**

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

**test('MultiSelectDropDown', async({page})=>{**

**await page.goto('https://testautomationpractice.blogspot.com/')**

**// Select multiple value from drop down**

**await page.selectOption('#colors',['Blue','Red','Green']);**

**// Assertion**

**// 1) Check number of option in dropdown**

**const count=await page.locator('#colors option');**

**await expect(count).toHaveCount(7);**

**// 2) check number of option in dropdown using JS array**

**const counts=await page.$$('#colors option');**

**// console.log(counts.length);**

**await expect(counts.length).toBe(7);**

**await page.pause();**

**})**

**Handling Auto Suggestion Box**

**Handling an auto-suggestion box in Playwright involves interacting with elements that dynamically generate suggestions based on user input, such as search bars or dropdowns. These elements are typically implemented with JavaScript, and their suggestions appear as a list of items after typing a query into an input field. Below is a detailed explanation of how to handle auto-suggestion boxes in Playwright.**

**Example of Handling Auto Suggestion Box**

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

**test('Handling Auto Suggest Drop Down', async({page})=>{**

**await page.goto('https://www.redbus.in/');**

**await page.locator('#src').fill('Delhi');**

**await page.waitForSelector('//li[@class="sc-iwsKbI jTMXri"]');**

**const value=await page.$$('//li[@class="sc-iwsKbI jTMXri"]');**

**for(const option of value)**

**{**

**const ABC=await option.textContent();**

**console.log(ABC);**

**if(ABC.includes('Dhaula Kuan Delhi'))**

**{**

**option.click();**

**}**

**}**

**await page.waitForTimeout(2000);**

**await page.locator('#dest').fill('Aurangabad');**

**await page.waitForSelector('//li[@class="sc-iwsKbI jTMXri"]');**

**const dest=await page.$$('//li[@class="sc-iwsKbI jTMXri"]');**

**for(const destination of dest)**

**{**

**const destvalue=await destination.textContent();**

**console.log(destvalue);**

**if(destvalue.includes('Central Bus Stand Aurangabad (Maharashtra)'))**

**{**

**destination.click();**

**}**

**}**

**await page.pause();**

**})**

**Handling Alerts**

**Handling alerts in Playwright involves interacting with JavaScript alerts, confirmation boxes, and prompt dialogs that pop up during test execution. Playwright provides robust methods to handle these browser dialogs. Below is a detailed explanation of how to work with alerts in Playwright.**

**Types of Alerts**

**There are three main types of dialogs in browsers that you might encounter:**

1. **Alert  
   A simple message dialog with an "OK" button.**

**javascript**

**Copy code**

**alert("This is an alert!");**

1. **Confirmation Box  
   A dialog with "OK" and "Cancel" buttons, often used for user confirmations.**

**javascript**

**Copy code**

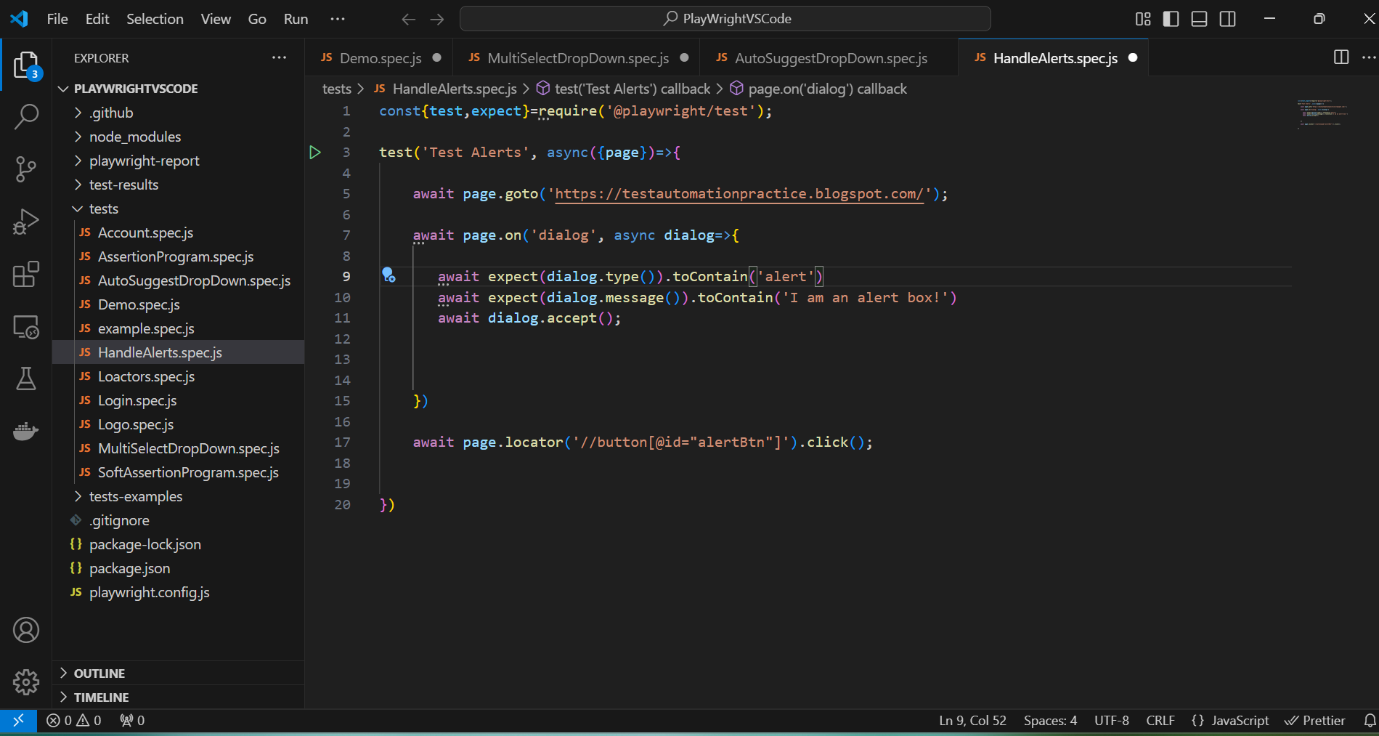
**confirm("Are you sure?");**

1. **Prompt  
   A dialog that requests input from the user, with an input field and "OK" and "Cancel" buttons.**

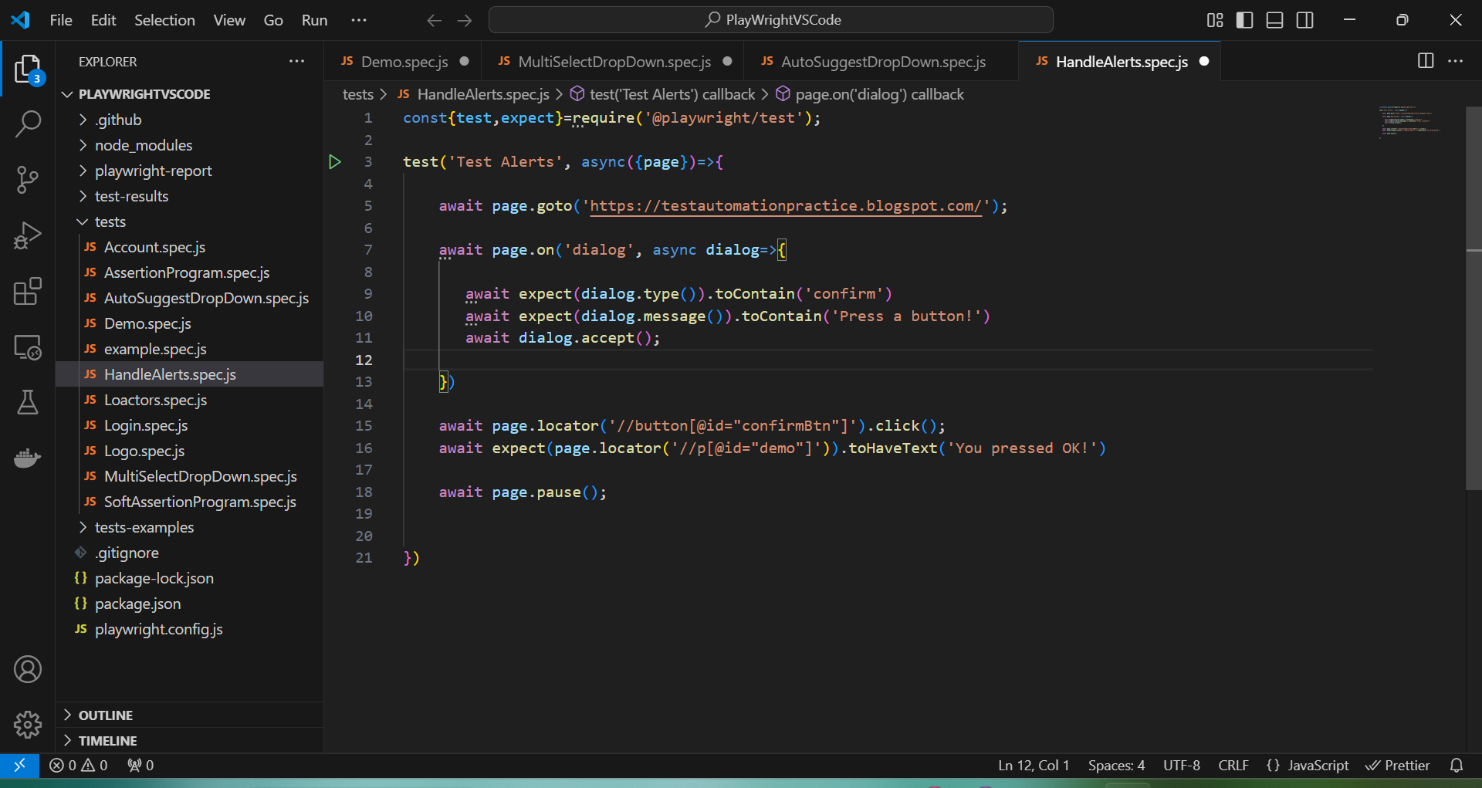
**javascript**

**Copy code**

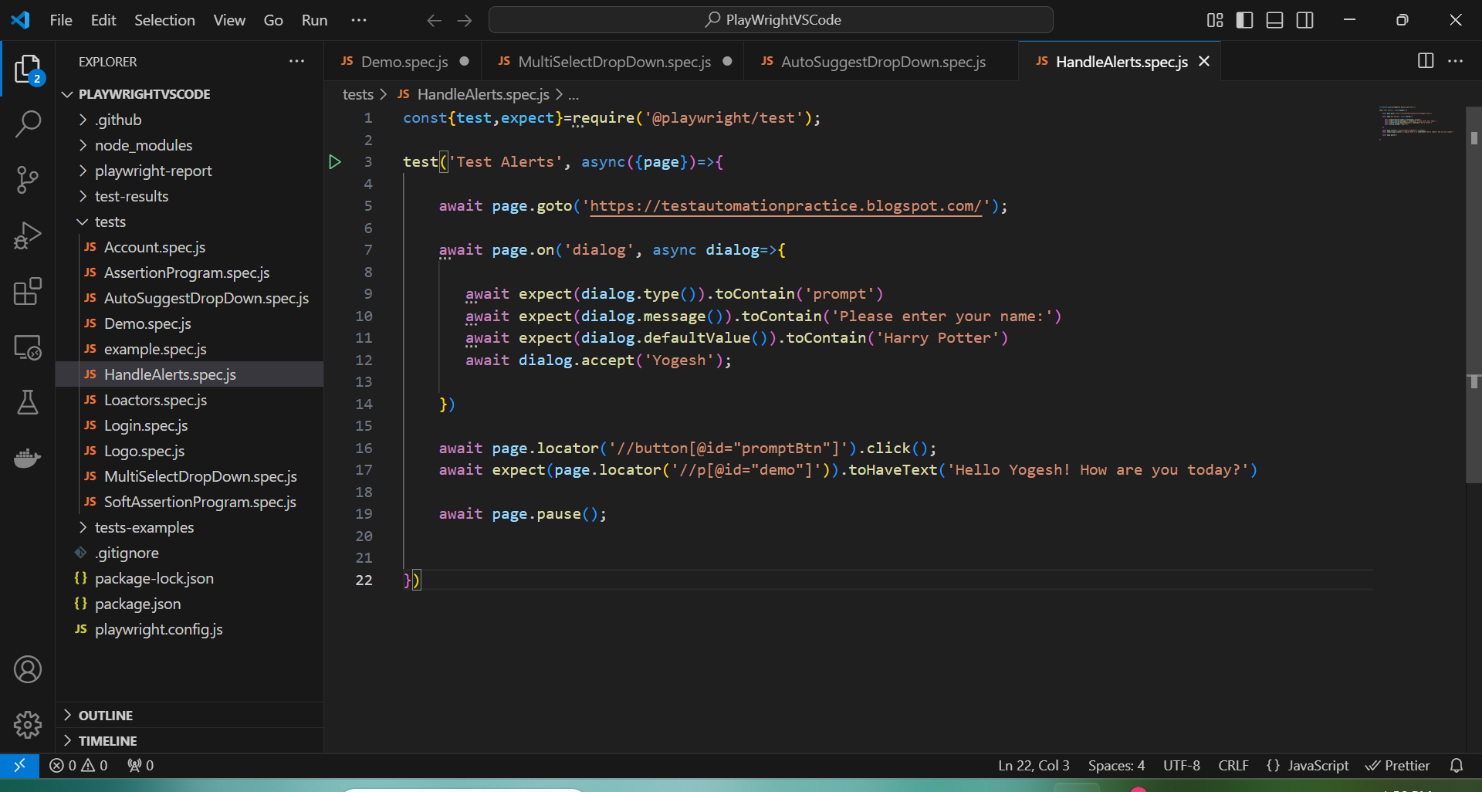
**prompt("Enter your name:", "Default Name");**

**1 Alert Example**

**2. Confirm Alert Example**

****

**3 Prompt Alert Example**

****

**Handling Frames**

**Handling frames in Playwright involves interacting with <iframe> elements embedded within a webpage. Frames are used to load separate HTML documents inside a parent page, often to display third-party content, ads, or isolated environments. Playwright provides robust support for handling frames, making it easy to interact with elements inside them.**

**Here's a detailed explanation of handling frames in Playwright:**

**1. Identifying Frames**

**Frames in Playwright can be accessed via the page.frame() or frame.childFrames() methods. Each frame has a unique name, url, or an attribute like id, which you can use to locate it.**

**2. Key Methods for Frames**

**Accessing a Specific Frame**

**You can locate a specific frame using:**

* **Name or ID:**

**javascript**

**Copy code**

**const frame = page.frame({ name: 'frameName' });**

* **URL Matching:**

**javascript**

**Copy code**

**const frame = page.frame({ url: /partial-url-pattern/ });**

**Getting All Frames**

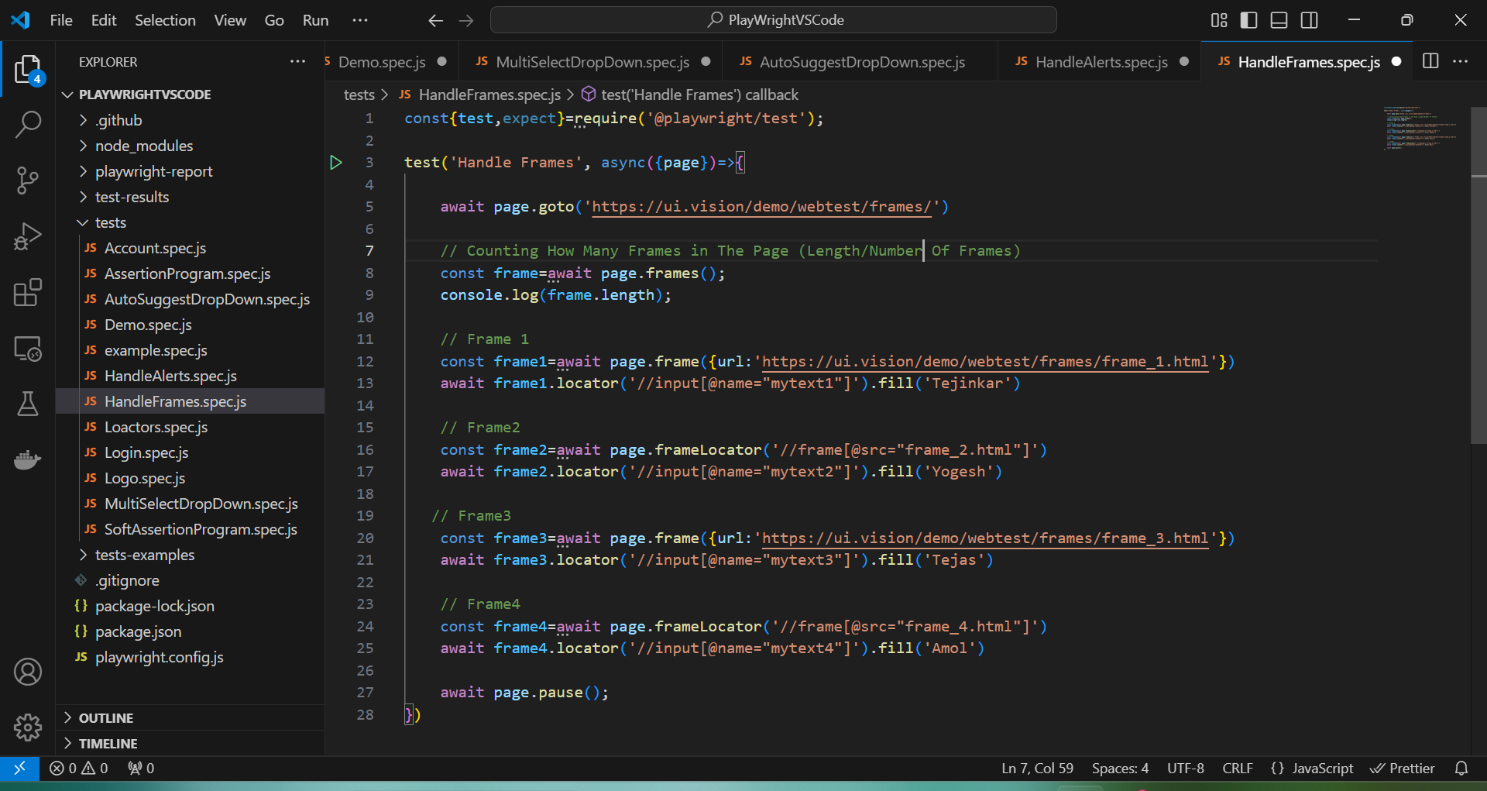
**javascript**

**Copy code**

**const frames = page.frames();**

**frames.forEach(frame => console.log(frame.url()));**

**Example :**

****

**Handling Child Frames :**

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

**test('Handle Child Frames', async({page})=>{**

**await page.goto('https://ui.vision/demo/webtest/frames/')**

**// Switch to Parent Frames**

**const ParentFrame=await page.frame({url:'https://ui.vision/demo/webtest/frames/frame\_3.html'})**

**// Switch to Child Frames**

**const Childframe=await ParentFrame.childFrames();**

**await Childframe[0].locator("//span[text()='I am a human']/parent::div/parent::div/parent::div/child::div[1]").click();**

**await page.pause();**

**})**

**Handle Web Table**

**Handling web tables in Playwright involves interacting with the rows, columns, and cells of an HTML table on a webpage. Playwright provides various methods to locate and manipulate table elements efficiently. Here's a detailed explanation:**

**Key Concepts for Handling Web Tables in Playwright**

**1. Understanding the Structure of a Web Table**

**An HTML table typically consists of:**

* **<table>: The main table container.**
* **<thead>: The table header containing <tr> (rows) and <th> (header cells).**
* **<tbody>: The body of the table containing <tr> (rows) and <td> (cells).**
* **<tfoot> (optional): The table footer.**

**Example Handling Web Table :**

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

**test('Handle Tables', async({page})=>{**

**await page.goto('https://testautomationpractice.blogspot.com/')**

**const table=await page.locator('//table[@id="productTable"]')**

**// Counts The Coloumn**

**const cloumn=await table.locator('thead tr th')**

**console.log('Number of coloumn', await cloumn.count())**

**// Counts The Row**

**const row=await table.locator('tbody tr')**

**console.log('Number of rows', await row.count())**

**/\***

**const matchrow=await row.filter({**

**has: page.locator('td'),**

**hasText: 'Laptop'**

**})**

**await matchrow.locator('input').check();**

**\*/**

**await Selectoption(row,page,'Laptop')**

**await Selectoption(row,page,'Tablet')**

**await page.waitForTimeout(3000);**

**// Print All Product Details**

**/\***

**for ( let i=0; i< await row.count();i++)**

**{**

**const Rows=row.nth(i)**

**const tds=Rows.locator('td')**

**for (let j=0; j< await tds.count()-1;j++)**

**{**

**console.log(await tds.nth(j).textContent())**

**}**

**}**

**\*/**

**// Print Number Of Pages in Table**

**const pages=await page.locator('//ul[@id="pagination"]/child::li')**

**console.log('Number of pages in the table',await pages.count());**

**// Print all pages data**

**for ( let p=0;p< await pages.count();p++)**

**{**

**if(p>0)**

**{**

**await pages.nth(p).click();**

**}**

**for ( let i=0; i< await row.count();i++)**

**{**

**const Rows=row.nth(i)**

**const tds=Rows.locator('td')**

**for (let j=0; j< await tds.count()-1;j++)**

**{**

**console.log(await tds.nth(j).textContent())**

**}**

**}**

**await page.waitForTimeout(3000);**

**}**

**})**

**// below function is used for selecting partiular product checkbox from table**

**async function Selectoption(row,page,name){**

**const matchrow=await row.filter({**

**has: page.locator('td'),**

**hasText: name**

**})**

**await matchrow.locator('input').check();**

**}**