1. **What is Node.js?**

**Answer:** Node.js is an open-source, cross-platform, beck-end JavaScript runtime environment to execute the JavaScript code outside the Web browser.

1. **How long does Playwright wait before giving up on waiting for an element?**

**Answer:** It waits for a given timeout (can be local or global, 30 sec default)

1. **What is browser context in playwright?**

**Answer**: Browser Contexts provide a way to operate multiple independent browser sessions. Playwright allows creating a new fresh browser window with

*Browser.newContext()* method

1. **Can Playwright identify elements by ID?**

**Answer:** Yes, Playwright can identify elements by ID

e.g. await page.click("#ID")

there is more locator in playwright e.g. Text, CSS, Select by attribute, with css, XPath

1. **When a page is opened using browser context, is the page rendered in GUI or opened headlessly?**

**Answer:** While executing the test case and browser opens headless(default). However, we can make it open in headed mode while executing the test case from terminal

e.g. *npx playwright test tests/tesecasename.spec.js --headed*  
or we can use option *headless: false* in configuration file*,*

1. **Can we see what errors look like when a test fails?**

**Answer:** Yes, we can see the errors of failure I reports, even the failed step as well

1. **What are Traces in Playwright**

**Answer:** Traces is the feature of Playwright that takes screenshots of each steps for the test case automatically, it will also record video of the executed step, it will re-run the test cases which are failed due to any flakiness and also maintain the logs.

1. **Why is waitForNavigation needed if Playwright has an auto-wait mechanism?**

**Answer:** It is there for when we might need it, for example to make it clear that you are expecting a navigation. Say for example you’re filling a multi-page form and each page has a “next” button that brings you to the next page. If you want to click “next” on the first two pages, you might use waitForNavigation after the first click and before the second one to make sure you don’t click the same first button twice.

1. **Can Playwright do API testing?**

**Answer:** Yes, Playwright can be used to get access to the REST API of the application.

Sometimes you may want to send requests to the server directly from Node.js without loading a page and running js code in it

1. **Does Playwright by default work asynchronously? That is, do we not have to use async/await explicitly?**

**Answer:** That depends on the language. Our Java is sync API, our Python can be both sync and async, our JavaScript is async.

1. **How do you an attribute of an web element in playwright?**

**Answer:** We can get the attribute of any web element with .getAttribute(“Attribute”)

**e.g.,**

const Att = await page.locator(“.locator”).getAttribute(“Attribute”);

1. **How to select Drop down**

**Answer:** we can select the drop-down by value, label, index

const dropdown = page.locator("select.form-control");

//Value

await dropdown.selectOption({vaue: “value1”});

//Label

await dropdown.selectOption({label: ”label1”});

// index

await dropdown.selectOption({index: 1});

1. **Can we run test cases on browser other than chrome?**

**Answer:** Yes, we can provide the browser information in configuration file

**e.g.,**

projects: [

{

**name: 'chrome',**

use: {

headless: false,

...devices['Desktop chrome'],

},

},

{

**name: 'firefox',**

use: {

headless: false,

...devices['Desktop Chrome'],

},

},

{

**name: 'webkit',**

use: {

...devices['Desktop Safari'],

},

},

]

1. **How to Handle Child window in Playwright?**

**Answer:** we need to give wait to open the new page/tab and use below code

Const newPage= await Promise.all([

context.waitForEvent(“Page)

linkOfNewpage.click()

]);

newPage.locator(“locator of new page element”).action()

1. **What is Codegen?**

**Answer:** Playwright’s codegen tool helps us to generate the script without writing that manually. It generates the test script code based on user interactions with the page.

Command is

*npx playwright codegen* [*http://applicationurl*](http://applicationurl)

perform the actions you want, and code will be automatically generated for same.

**Purpose of Using async-await in Playwright**

Playwright is an asynchronous framework, meaning most of its functions return **Promises**. To ensure smooth execution of test scripts, Playwright leverages **async-await**, which allows handling these asynchronous operations in a synchronous-looking manner. Here’s why it's useful:

**1. Handling Asynchronous Operations**

Playwright interactions (e.g., navigating pages, clicking buttons, filling forms) take time to complete. Using await ensures that the test execution waits for the action to finish before proceeding.

Playwright is an **asynchronous framework**, meaning most of its actions (e.g., navigating, clicking, filling forms) return **Promises**. If you don’t use await, the test execution will not wait for these actions to complete, leading to failures.

* Playwright does not pause to complete each step. It simply moves to the next line **without waiting**.
* Playwright **automatically waits** for elements to be available before interacting. But if you don’t use await, the script moves on immediately without checking if the element is actually present.
* If an error happens in an asynchronous operation without await, it might not be caught properly.

**✅ Example (Without await) - May Fail**

test('example test', async ({ page }) => {

page.goto('https://example.com'); // Navigation starts, but test doesn't wait

page.click('button#submit'); // This may execute before page loads

});

💡 **Issue:** The test may attempt to click the button before the page has fully loaded.

**✅ Example (With await) - Correct Way**

test('example test', async ({ page }) => {

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

await page.click('button#submit'); // Waits for the button to be ready before clicking

});

✅ Ensures each step completes before moving to the next.

**2. Avoiding Callback Hell**

Playwright functions return Promises, so chaining them with .then() can lead to complex nested structures. Using async-await makes the code cleaner and easier to read.

**✅ Without async-await (Hard to Read)**

page.goto('https://example.com').then(() => {

page.click('button#submit').then(() => {

page.fill('#name', 'John Doe').then(() => {

console.log('Form submitted');

});

});

});

**✅ With async-await (More Readable)**

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

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

await page.fill('#name', 'John Doe');

console.log('Form submitted');

**3. Ensuring Proper Synchronization**

Playwright automatically waits for elements to be available before interacting. However, without await, execution may proceed before an element is ready.

**✅ Example (With await)**

await page.waitForSelector('#username'); // Ensures element is present

await page.fill('#username', 'testuser'); // Fills the input field safely

**4. Handling API Calls & Responses**

Playwright can test API requests too. Since API calls return Promises, async-await is needed for handling responses properly.

const response = await page.request.get('https://api.example.com/data');

const data = await response.json();

console.log(data);

**5. Exception Handling with try-catch**

Using async-await makes it easier to catch and handle errors properly.

javascript

CopyEdit

try {

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

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

} catch (error) {

console.error('Test failed:', error);

}