7th Nov 2025

🎯 Playwright Interview Questions

Basic Level (Playwright Fundamentals)

1.What is Playwright, and how is it different from Selenium or Cypress?

Playwright is a modern end-to-end testing framework by Microsoft that supports multiple browsers (Chromium, Firefox, WebKit) and languages out of the box.

Unlike Selenium or Cypress, Playwright offers faster execution, built-in parallel testing, and powerful features like auto-waiting, network mocking, and cross-browser testing in a single API.

2. What programming languages does Playwright support?

Playwright supports JavaScript, TypeScript, Python, Java, and .NET (C#) — making it versatile across different tech stacks.

Most commonly, it's used with JavaScript/TypeScript for front-end and web automation testing.

3. How do you install and set up Playwright in a new project?

npm init playwright@latest

It automatically installs Playwright, sets up example tests, configuration files, and downloads the required browser binaries — so you're ready to run tests instantly.

4. What is the command to create a Playwright project?

npm init playwright@latest

This command initializes a new Playwright project.It sets up folders, installs dependencies, and creates example tests and configurations automatically.

5. What is the role of the playwright.config.js file?

The playwright.config.js file defines the global configuration for your Playwright tests. It controls settings like browser types, base URL, test timeouts, reporters, and parallel execution. This helps maintain consistency and reduces repetitive setup across all tests.

6. What are the main components or fixtures provided by Playwright?

Playwright provides key fixtures that help manage test setup and teardown efficiently. The main components are:

browser – Launches and manages a browser instance.

context – Creates a new browser context (isolated session).

page – Opens a new tab or page within a context.

request – Enables API testing by sending network requests without UI interaction.

7. What are browser contexts and pages in Playwright?

In Playwright, a browser context is like a separate, isolated browser session — similar to opening a new incognito window. It allows running multiple tests in parallel without interference.

A page is a single tab within that browser context where the actual test actions (like clicking, typing, or navigating) take place.

8. What is the difference between a browser instance and a browser context?

A browser instance represents the entire browser application (like launching Chrome or Firefox).

A browser context is an isolated environment within that browser instance — similar to a private/incognito window. Multiple contexts can run independently inside one browser instance, enabling parallel, faster, and isolated test execution.

9. What are Playwright fixtures?

Playwright fixtures are built-in test setup and teardown helpers that provide ready-to-use objects like browser, context, and page to your tests. They ensure each test runs in a clean, isolated environment. Fixtures make tests more reliable by automatically creating and closing resources before and after each test.

10. What is the difference between test and describe in Playwright?

In Playwright:

test is used to define an individual test case — it contains the actual steps and assertions to verify a specific functionality.

describe is used to group multiple related tests together for better organization and readability.

☞ Example:

```
test.describe('Login tests', () => {
  test('should login with valid credentials', async ({ page }) => {
    // test steps
  });
});
```

11. How do you launch a browser manually in Playwright?

You can launch a browser manually in Playwright using the chromium, firefox, or webkit objects.

Example:

```
const { chromium } = require('@playwright/test');
(async () => {
  const browser = await chromium.launch({ headless: false }); // launches browser
  const context = await browser.newContext();
  const page = await context.newPage();
  await page.goto('https://example.com');
  await browser.close();
})();
```

launch() starts the browser.

newContext() creates an isolated session.

newPage() opens a new tab for interaction.

12. What are the supported browsers in Playwright?

Playwright supports four major browsers:

Chromium → Covers Google Chrome and Microsoft Edge.

Firefox → Supports Mozilla Firefox.

WebKit → Covers Safari (Apple's browser engine).

Google Chrome (Stable Channel) → Can be run directly using the channel: 'chrome' option.

13. What is headless vs headed mode in Playwright?

In headless mode, the browser runs without a visible UI ,it executes tests in the background, making them faster and ideal for CI/CD pipelines.

In headed mode, the browser UI is visible on screen, allowing you to watch the test actions in real time , useful for debugging and development.

14. How do you record scripts in Playwright using the codegen tool?

You can record scripts in Playwright using the codegen command:

npx playwright codegen <URL>

✅ Example:

npx playwright codegen https://example.com

This opens a browser window and automatically records your actions (like clicks or typing), generating Playwright test code in real time — which you can copy, modify, and reuse in your tests.

15. What is the default timeout for actions in Playwright?

The default timeout for actions in Playwright is 30 seconds (30,000 ms).

This means Playwright will wait up to 30 seconds for actions like clicks, navigation, or element visibility before throwing a timeout error. You can override it using page.setDefaultTimeout() or in the test configuration file.

16. What are locators in Playwright?

Locators in Playwright are used to find and interact with elements on a web page. They provide a reliable way to perform actions like click, fill, or hover without worrying about timing issues.

✅ Example:

```
const button = page.locator('text=Login');
await button.click();
```

Locators automatically wait for the element to be ready before performing the action.

17. What is the difference between page.locator() and page.$()?

The main difference is in how they find and handle elements:

page.locator() → A modern, recommended API that retries automatically until the element is ready. It supports multiple actions (click, fill, etc.) and handles dynamic elements efficiently.

page.$() → Returns the first matching element immediately (like document.querySelector()), without auto-waiting or retrying — so it's less reliable for dynamic pages.

Use page.locator() for most test automation scenarios

18. What is the purpose of await and async in Playwright tests?

In Playwright, async and await are used to handle asynchronous operations smoothly.

async marks a function as asynchronous, allowing the use of await inside it.

await pauses the execution until a Promise (like a page action or navigation) is completed.

This ensures Playwright waits for each action (like click or goto) to finish before moving to the next step, preventing flaky tests.

19. What is Playwright Test Runner?

The Playwright Test Runner is Playwright's built-in framework for writing, organizing, and executing tests.

It provides features like parallel test execution, fixtures, reporters, retries, and automatic waiting. This makes it a complete end-to-end testing solution without needing external tools like Jest or Mocha.

20. How do you run Playwright tests from the command line?

You can run Playwright tests from the command line using:

npx playwright test

This command runs all tests inside the tests folder by default.

You can run a specific test file using:

npx playwright test <filename>

Use flags for customization, e.g.:

npx playwright test --headed --project=chromium

++