Playwright Setup and Kick Start

Prerequisites for Installing Playwright

1. **Install Node.js** – Required for running JavaScript-based projects.
2. **Install Visual Studio Code (VS Code)** – A recommended code editor for development.
3. **Create a Project Folder** – Open the folder using VS Code to begin development.

Installing Playwright

Open the terminal in your project folder and run:

npm init playwright@latest

This command initializes a Playwright project and creates the following files and folders:

- **package.json** – Manages project dependencies and scripts.
- **playwright.config.js** – Contains Playwright configuration settings.
- **tests/** – Directory for organizing test files.

To check the installed version of Playwright:

npx playwright --version

Writing and Running Your First Playwright Test

Step 1: Create a New Test File

Inside the tests/ directory, create a new file, for example, FirstTest.spec.ts.

Step 2: Import Playwright Module

Add the following line at the top of your test file:

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

- test is used to define test cases.
- expect is used for assertions.

Step 3: Create a Test Block

Write your test using the test function:

```
test('Verify page title', async ({ page }) => {
  await page.goto('https://example.com');
  const title = await page.title();
  expect(title).toHaveURL('Example Domain');
});
```

Understanding async and await

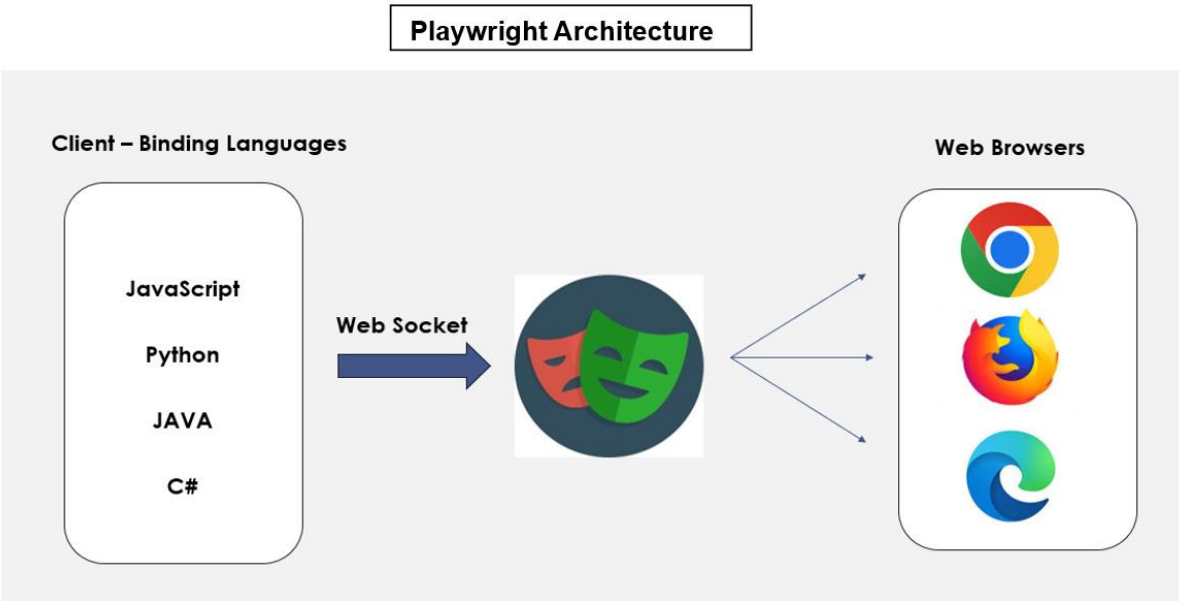
Many Playwright APIs such as page.goto(), page.title(), and page.url() return Promises because they involve asynchronous browser operations. To handle them correctly:

- Use **await** to pause execution until the Promise resolves.
- Mark the function as **async** to allow the use of await.

Keyword	Description
async	Declares a function that returns a Promise and can use await
await	Pauses the function execution until the Promise resolves

Running Tests and Debugging

Command	Description
npx playwright test	Runs all tests on all browsers in headless mode.
npx playwright test --headed	Runs all tests in headed (non-headless) mode.
npx playwright show-report	Opens the HTML test report.
npx playwright test mytest.spec.ts	Runs a specific test file.
npx playwright test --project=chromium --headed mytest.spec.ts	Runs a specific test file only on Chromium in headed mode.
npx playwright test mytest1.spec.ts mytest2.spec.ts	Runs multiple specified test files.
npx playwright test -g "test title"	Runs the test(s) that match the given title.
npx playwright test --project=chromium	Runs all tests on the Chromium browser only.
npx playwright test --debug	Runs tests in debug mode.
npx playwright test example.spec.ts --debug	Debugs a specific test file.
npx playwright test mytest.spec.ts –ui	Run the test in UI Mode



key components of Playwright Architecture

1. **Client or language binding** – Playwright supports multiple programming languages and environments like Java, JavaScript, typescript, python, etc.
2. **Web Socket** - Playwright uses a **web socket protocol** to interact with the client and server, in web socket protocol you can send back-to-back requests without terminating the connection which helps Playwright to perform test execution at a much faster pace than other automation tools.
3. **Browser Context** – It's an isolated instance of a browser that manages its storage, session IDs, cookies, caches, etc. This feature sets apart playwright from other automation tools by enabling the parallel execution of test scripts which speeds up the testing process.
4. The above image gives us a brief understanding of how Playwright operates internally, firstly client or language bindings are the programming language that we use to develop the test scripts. As we all know HTTP uses a request-response model with stateless single interactions which means that after each request & response, the connection gets terminated and with each new request a new connection is established between the client and server so it's much slower. Playwrights make use of web sockets instead of HTTP to interact with clients and servers. So once a connection is established between the client and server via web socket protocol, we can send back-to-back requests without terminating the connection which makes it much faster. Once the connection is established test execution starts, and all test cases can use the same connection for their execution, once execution is completed the connection is disabled.
5. Projects with modern applications that require multiple browser contexts, network controls, and fast and reliable cross-browser testing can opt for the Playwright tool. some of the real-time projects are Live trading applications, Gaming, Slack & GitHub.

   Reference: https://www.testingmavens.com/blogs/architecture-breakdown-selenium-cypress-and