**Playwright enables reliable end-to-end testing for modern web apps.**

### Any browser • Any platform • One API

**Cross-browser.** Playwright supports all modern rendering engines including Chromium, WebKit, and Firefox.

**Cross-platform.** Test on Windows, Linux, and macOS, locally or on CI, headless or headed.

**Cross-language.** Use the Playwright API in [TypeScript](https://playwright.dev/docs/intro), [JavaScript](https://playwright.dev/docs/intro), [Python](https://playwright.dev/python/docs/intro), [.NET](https://playwright.dev/dotnet/docs/intro), [Java](https://playwright.dev/java/docs/intro).

**Test Mobile Web.** Native mobile emulation of Google Chrome for Android and Mobile Safari. The same rendering engine works on your Desktop and in the Cloud.

### Resilient • No flaky tests

**Auto-wait.** Playwright waits for elements to be actionable prior to performing actions. It also has a rich set of introspection events. The combination of the two eliminates the need for artificial timeouts - the primary cause of flaky tests.

**Web-first assertions.** Playwright assertions are created specifically for the dynamic web. Checks are automatically retried until the necessary conditions are met.

**Tracing.** Configure test retry strategy, capture execution trace, videos, screenshots to eliminate flakes.

### No trade-offs • No limits

Browsers run web content belonging to different origins in different processes. Playwright is aligned with the modern browsers architecture and runs tests out-of-process. This makes Playwright free of the typical in-process test runner limitations.

**Multiple everything.** Test scenarios that span multiple **tabs**, multiple **origins** and multiple **users**. Create scenarios with different contexts for different users and run them against your server, all in one test.

**Trusted events.** Hover elements, interact with dynamic controls, produce trusted events. Playwright uses real browser input pipeline indistinguishable from the real user.

**Test frames, pierce Shadow DOM.** Playwright selectors pierce shadow DOM and allow entering frames seamlessly.

### Full isolation • Fast execution

**Browser contexts.** Playwright creates a browser context for each test. Browser context is equivalent to a brand new browser profile. This delivers full test isolation with zero overhead. Creating a new browser context only takes a handful of milliseconds.

**Log in once.** Save the authentication state of the context and reuse it in all the tests. This bypasses repetitive log-in operations in each test, yet delivers full isolation of independent tests.

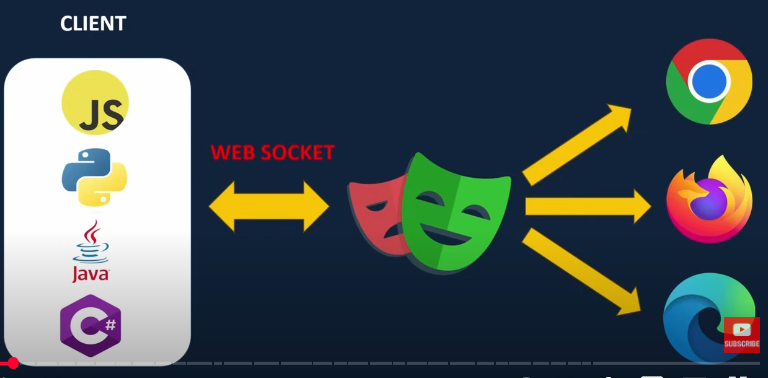
### Powerful Tooling

[**Codegen.**](https://playwright.dev/docs/codegen) Generate tests by recording your actions. Save them into any language.

[**Playwright inspector.**](https://playwright.dev/docs/debug#playwright-inspector) Inspect page, generate selectors, step through the test execution, see click points, explore execution logs.

[**Trace Viewer.**](https://playwright.dev/docs/trace-viewer-intro) Capture all the information to investigate the test failure. Playwright trace contains test execution screencast, live DOM snapshots, action explorer, test source, and many more.

Playwright Architecture



In Client side – we have different languages, such as JS, Python, Java, C# - using these languages we write Test script.

Once writing the Test Script, using WEB SOCKET we will be connected the Playwright server, using this server we connect to the Browsers like Chrome, Mozilla, Internet Edge.

Here, WEB Socket works differently than Selenium and Cypress

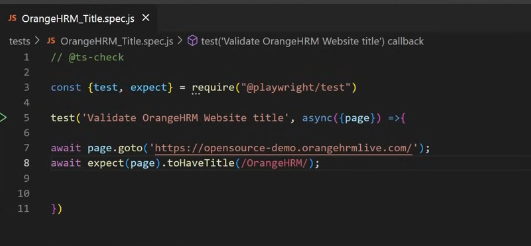
* Using Clients (languages) we will be sending the request to Playwright Server
* Playwright do a handshake establish a connection between the Client
* All communication will be done using the WebSocket, there will be no multiple connection or no discontinuation between the Client and the Server
* Then by language traffic and here is the Web Socket comes in

When the Playwright do a Handshake, it will create a permanent connection and it will be a two-way connection, likewise we don’t have develop another connection. For each test Script request, Playwight will be sending and executing according to those browser

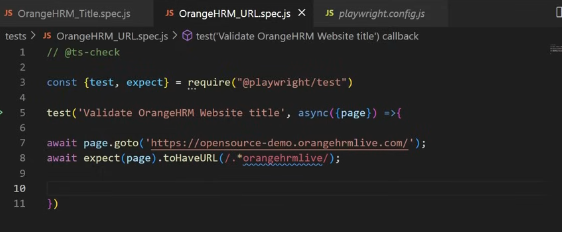
In other protocol like in HTTP, every time we send the request, it will connect with the server each time, wherein Playwright server its one time request for a continuous traffic. Hence Playwright is very fast and works efficiently.

**First Test Script**

**// To Check for the Script**



//toHaveURL()



//Running Playwright Tests



// Playwright Codegen

