Engineer

Playwright

the tool that drives a browser in tests

What it is

Playwright is an open-source framework from Microsoft for automating web browsers, with one programming interface for Chromium, Firefox and WebKit. Playwright Test, published on npm as `@playwright/test`, is the test runner built on top of it, and it is what most people mean when they use the name. The same project ships the plain library for scripts that are not tests, a command-line tool and an MCP server aimed at AI agents, and an extension for Visual Studio Code, all under the Apache License 2.0.

Beyond JavaScript and TypeScript, the same browser control is available from Python, where the project recommends its pytest plugin, from Java, with any test framework such as JUnit, and from .NET, with base classes for MSTest, NUnit and xUnit.

`npx playwright install` downloads its own builds of the three engines, and because each release moves those versions forward, the command may need to be run again after an upgrade. It can also drive Google Chrome and Microsoft Edge where they are already on the machine. Tests run headless by default, without a visible window. The installation guide’s example runs tests in parallel across all three engines through configured browser projects; without those projects, the runner’s default browser is Chromium. Each test gets a fresh browser context, the equivalent of a new browser profile, so one test’s cookies and storage do not carry into the next.

A test reads like the steps a person would take:

```
import { test, expect } from "@playwright/test";

test("signing in opens the inbox", async ({ page }) => {
```

```
    await page.goto("https://example.com/sign-in");
    await page.getByLabel("Email").fill("reader@example.com");
    await page.getByRole("button", { name: "Sign in" }).click();
    await expect(page.getByRole("heading", { name: "Inbox" })).toBeVisible();
});
```

Two mechanisms carry most of the weight. Before an action, Playwright runs actionability checks and waits until all of them pass. For a click, the locator must resolve to exactly one element, and that element must be visible, stable, which means its bounding box stays the same for at least two animation frames in a row, able to receive the click rather than sitting under another element, and enabled. If the checks do not pass within the timeout, the action ends in a `TimeoutError`. Assertions written as `await expect(locator)` behave the same way: they fetch the element again and again until the condition holds or the assertion timeout, five seconds by default, runs out.

Use the concept in a decision

Put “What is Playwright?” to work in a real decision. We can help relate the concept to the system you are building or buying. One team taking an idea from a sketch to something your customers can log into, instead of handing it between three suppliers.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Led a GIS map application to 10x revenue growth — 10x revenue growth, led from the map product

Led a GIS map application to 10x revenue growth Get in touch →

Why it matters

The decision Playwright serves is which behaviour has to be checked in a real browser, and how to make those checks trustworthy enough that a red build is believed. Some faults exist only once the page is assembled: a dialog that covers the button it should reveal, a layout that hides a field at one width, a script that works in one engine and throws in another. Tests that run pieces of code in isolation cannot see them.

Browser tests turn flaky when the test and the page race each other. The page fetches data, animates and re-renders, and a test that clicks the moment its line runs sometimes clicks too early. The usual remedy was a fixed pause, too short on a slow machine and wasted time on a fast one. Auto-waiting and retrying assertions replace the pause with the condition the test actually depends on. The assertions guide puts it plainly: web pages show information asynchronously, and assertions that check only once lead to flaky tests.

The second choice is how a test finds things. The locators guide recommends user-facing attributes first, and role locators above all, because `getByRole` with an accessible name is the closest a test gets to "how users and assistive technology perceive the page". That has a side effect worth having: a test that clicks the button named "Sign in" breaks if the button loses its name, which is the same defect a screen reader user would meet. The guide also says role locators "do not replace accessibility audits and conformance tests". Test ids are the fallback. The guide calls them the most resilient option and also "not user facing", so a suite that finds everything by id proves less about what a visitor experiences. CSS and XPath selectors are not recommended, because the DOM changes often.

The cost is time. Starting a browser and loading a page makes browser tests the slowest kind a project has. When one breaks on a build server, the trace viewer is the way back in. The documented configuration example records a trace on a failed test's first retry, with retries set to two on CI and zero locally; the runner's built-in defaults are no tracing and no retries. The trace steps through each action with the page before and after it, the network requests and the console.

What it looks like in practice

One platform here keeps its business logic in the database, and the work was to build a layered test suite across four layers. Its stated task was to check each layer's behaviour where that behaviour actually lives, "rather than everything being checked from the outside through a browser." The Go API carries 981 test functions. The database carries 494 behavioural test files, so a stored function is tested in the database instead of through three layers of application above it. The frontend carries 543 Vitest and Playwright specs, including 48 end-to-end files and 30 browser specs; the page gives one count for both tools. The whole suite runs as 10 jobs in a 612-line workflow, and the page is direct about the price: the suite is slow, it taxes every change, and at this size it needs maintenance of its own.

This site drives its own browser checks with a different library, Puppeteer, and its credits page shows the same trade-off. Those checks cover what a validator cannot, such as measured contrast, focus order, target size, viewport overflow and print pagination. The gate runs all of them but viewport overflow before every push; that one, like the pixel-for-pixel visual regression check, waits for a person to run it on demand.

Where it goes wrong

Most Playwright trouble comes from tests that check less than their names claim.

A fixed pause stands in for a condition. A test waits two seconds and then reads the page, or reads a value once and compares it with a generic assertion. On a busy build server two seconds is sometimes too short. Assert on what a visitor would see with a web-first assertion such as `toHaveText`, and let it retry.

Elements are found by their markup. A selector through three nested `div` elements or a styling class breaks on the next redesign while the page still works. Find the element by role and accessible name, or by label or text, and add a test id only where nothing a user perceives identifies it.

A passing role locator is taken as an accessibility check. `getByRole` proves that one element has the role and name the test asked for. It says nothing about contrast, focus order or the rest of the page. For that, run a rule engine such as axe from the test, as Playwright's own guide does through `@axe-core/playwright`; the scan covers the page "in its current state when you call it", so open the menu or the dialog first.

A click is forced through. The `force` option turns off the non-essential actionability checks; for a click, Playwright stops checking that the target actually receives the event. A test that needs it is often working around something a visitor would also meet, such as a banner over the button. Treat the original error as a finding, and either dismiss the banner the way a visitor would or fix the overlap.

A flaky test is retried until it passes. With retries on, a test that fails on the first run and passes on a retry is marked flaky rather than failed. A test that is flaky week after week is either racing the page or catching a real intermittent fault, and the trace recorded on its first retry is where to start looking.

Everything is checked through the browser. A rule about prices or permissions, tested by clicking through three screens, is slow and breaks for reasons unrelated to the rule. Test the rule where it lives, and keep browser tests for what only a browser can show: the journey, the layout and the differences between engines.

Further reading

- Installation — installing Playwright Test, running the example tests, the HTML report and UI Mode.
- Locators — the recommended order for finding elements, from role and label to test ids.
- Auto-waiting — the actionability checks behind each action and what `force` turns off.
- Assertions — retrying and non-retrying assertions, the default timeout, soft assertions and polling.
- Trace viewer — recording a trace on retry and stepping through a failed test action by action.
- Browsers — installing the engines, running branded Chrome and Edge, and the two headless modes.

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is Playwright?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/playwright/