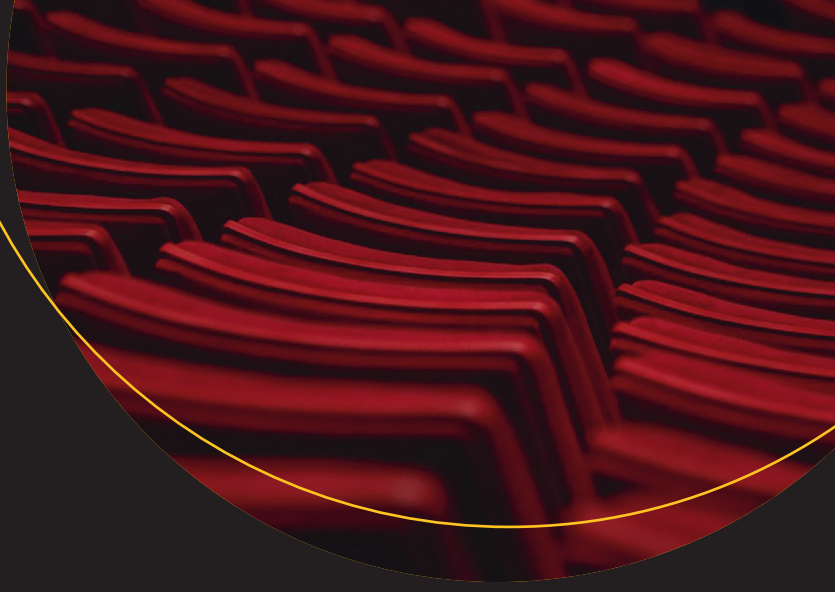

Practical Playwright Test

Next-Generation Web Testing and Automation

Jean-François Greffier

Apress®

Practical Playwright Test

Next-Generation Web Testing and Automation

Jean-François Greffier

Apress®

Practical Playwright Test: Next-Generation Web Testing and Automation

Jean-François Greffier
Rennes, France

ISBN-13 (pbk): 979-8-8688-2159-2 ISBN-13 (electronic): 979-8-8688-2160-8
https://doi.org/10.1007/979-8-8688-2160-8

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Anandadeep Roy
Project Manager: Jessica Vakili

Cover designed by eStudioCalamar

Cover image designed by Holger Langmaier on pixabay

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a Delaware LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub. For more detailed information, please visit https://www.apress.com/gp/services/source-code.

If disposing of this product, please recycle the paper

Life is a series of obstacles and challenges best engaged through assiduous study and frequent jumps into uncertainty. Anything else is boring.

—American McGee

Table of Contents

About the Author

Jean-François Greffier is an experienced front-end engineer. His main interests are software craft and testing. Recently awarded Microsoft MVP, he is a Playwright ambassador. He is not only an advocate, but also an open source contributor to Playwright on documentation, bug fixes, and features. He is also the maintainer of the Playwright Angular Schematic.

About the Technical Reviewer

Simon Knott is a Senior Software Engineer on the Playwright Core Team at Microsoft. Before joining Microsoft, he worked on the Content Delivery Networks and Functions infrastructure at Netlify and founded Quirrel, an open source job queueing solution acquired by Netlify. Simon holds a bachelor's degree in Computer Science from Hasso Plattner Institute near Berlin.

Simon is the creator of SuperJSON, a widely adopted JSON serialization library, was part of the React 18 Working group, and has contributed to projects like Chromium, VS Code, Next.js, Apache Traffic Server, and Pandas. Simon has presented at the BeJS conference, local Typescript meet-ups, and the FSJam Podcast and served as the technical reviewer for this very book!

Simon is passionate about making modern web development more productive. When he doesn't, you'll usually find him cooking, woodworking, or cycling around Berlin.

Acknowledgments

First, thanks to Anandadeep Roy for trusting me with this project and for his enthusiasm. Many thanks to everyone at Apress for helping me to focus on actually writing the book.

Thanks to my family for their patience and ongoing support.

Simon Knott helped to make the book not only technically correct but also much better. He made this the best version of the book it could be. Simon, I do hope we can grab a coffee someday.

Thanks to the beta readers Tanguy Michel and Ronan Barbot, who gave me a welcome different perspective. Thanks to Butch Mayhew, who gave me feedback in the early process of laying down my plan. Thanks to Tanguy Michel and Nica Mellifera for agreeing to be interviewed.

Many thanks to Debbie O'Brien for welcoming me among the Playwright Ambassadors.

If it takes a village to raise a child, it certainly takes a tech community to create a book. RennesJS, Snowcamp, Slickteam, Conserto Explos, Playwright Ambassadors – these meetups, conferences, and communities helped me to grow. Thank you.

Introduction

Who Is This Book For?

For a long time, programming was a complicated craft. It had to address almost impossible technical challenges: How do I race against the beam and create a video game out of a few transistors? How can I make great software with a few bytes of RAM? It also needed to be correct and reliable, with legal requirements like calculating taxes accurately.

Software is everywhere: in our pockets, our cars, our several microcomputers. We didn't quite reach the ubiquitous computing the engineers dreamed of in the 1980s, but the closest thing we have are web apps. You can access your favorite applications, with your profile and data, from pretty much any devices that hold a web browser. Our smartphones are multiple times more powerful than the ancient machines that started personal computing.

Now as developers we have to deal with several languages, frameworks, dependencies, but we also need to care about deployment, security, accessibility, design, multi-platform… I believe that today, writing software is complex. To paraphrase the Manifesto for Agile Software Development, we are still uncovering better ways of developing software. To help us to make great software, we need great tooling. Microsoft Playwright is an awesome end-to-end testing solution that I discovered a few years ago. It's a great piece of tooling that can help you craft better software and untangle its complexity.

This book is primarily meant for front-end or full-stack developers that want to further progress by mastering end-to-end testing with Playwright Test. Quality Assurance professionals should also find value in this work, as it goes in-depth with Playwright Test techniques, automation, and testing strategy. Actually, this book is for any individual contributors that care deeply about high-quality software.

You don't have to be an expert front-end developer, but some knowledge of HTML, CSS, and TypeScript is needed. A curiosity toward testing principles is a plus. Being familiar with unit test frameworks like Jest or Vitest is helpful, but experience with end-to-end testing tools is not necessary.

Playwright officially supports TypeScript, Python, Java, and .Net. However, I chose to focus on the TypeScript flavor and its test runner: Playwright Test.

The first chapters will help you get started with Playwright Test and give you great basics. The second part details advanced concepts and will hopefully be a resource while you practice what you've learned. Finally, the last part focuses on what's beyond end-to-end testing and how to use Playwright Test in your testing strategy.

CHAPTER 1

Getting Started

In this chapter, you'll get an overview of what Playwright Test is about and its advantages and inconveniences, so you can decide when to use it. You'll learn how to set up your environment and run your first test. Look for useful online resources at the end to complement this book.

Introducing Playwright

Playwright is an open source project sponsored and primarily maintained by Microsoft. It automates the web by driving three browsers: Chromium, Firefox, and WebKit (the tech behind Safari). This means that you can automate and test your web applications on all three major browser families.

Moreover, Playwright is an automation solution geared toward end-to-end testing. It provides assertions adapted to web testing, its own Playwright Test runner, as well as a Codegen generator, and a trace recorder.

With its Developer eXperience, speed of execution, availability for different languages and test reliability, Microsoft Playwright quickly gained in popularity. In this chapter, we'll look at how to prepare your environment, install Playwright, and create your first testing project.

Why Choose Playwright

Testing is sometimes seen as too complex, difficult to implement in the customer or technical context, time-consuming, useless, and so on. Fortunately, unit and integration tests are becoming increasingly accepted practices, but end-to-end tests are not. They are rarely used, as they are considered slow, fragile, and difficult to write and maintain. End-to-end testing reputation is so bad among developers that many relegate these tests to specialized automation engineers.

J.-F. Greffier, *Practical Playwright Test*, https://doi.org/10.1007/979-8-8688-2160-8_1

Playwright is significantly faster than competing end-to-end testing solutions. In fact, as demonstrated in the article "Cypress vs Selenium vs Playwright vs Puppeteer speed comparison,"[1] the tool is fast, even for large test suites. Furthermore, it's easy to parallelize test execution via vertical scaling (more CPUs) or horizontal scaling (more machines). These parallelization features are built-in and do not rely on third-party plugins.

Tests on browsers are often fragile, typically "working on my machine" but failing in continuous integration. This phenomenon, known as flakiness, is well known to end-to-end testers. Depending on network conditions and the target machine, it's sometimes tempting to wait a second or two "just in case." Or wait until something is eventually present.

Fortunately, Playwright takes care of all this automatically with **auto-wait**. Before performing an action, the test framework will ensure that the criteria are met. For example, before clicking on an element, it must be enabled, visible, not moving, and have a listener for the click.

What also happens is that a change in the application breaks the tests: the DOM structure has changed, or the CSS classes. These tests that break at the slightest change are too closely linked to implementation: use CSS or XPath selectors to find the elements to be tested.

Playwright offers numerous types of selectors: text, accessibility attributes, CSS, or even positional. Above all, you can combine selectors to find what's both relevant and robust. Playwright's generator, Codegen, generates the most robust selector possible.

Playwright tests are easier to write thanks to Codegen. It opens a browser window in which you interact with the web application you want to test, and a second window with the Playwright inspector. Your actions are recorded and create code in the language of your choice. This tool also enables you to test and design more readable and robust selectors.

Tests are also easy to debug. Step-by-step tests can be run using either the VS Code extension or Codegen. Best of all, Playwright's trace viewer provides a history of Playwright calls, browser console, and network traces and allows you to inspect the browser page at any point in time, even if the test ran on a different machine, like your Continuous Integration.

[1] https://blog.checklyhq.com/cypress-vs-selenium-vs-playwright-vs-puppeteer-speed-comparison

Architecture

I've often been asked what are Playwright, Playwright Test, and why it's only available in TypeScript when you can test in other languages.

Playwright is actually made up of several tools:

- **Playwright Library:** This is the heart of Playwright, enabling it to talk to different browsers. It can be used in several languages: TypeScript, Python, Java, .Net officially, but also Go, Ruby, and more thanks to open source language bindings. This library is used by many projects in different contexts because it only does browser automation, and does it well. No tests here.
- **Playwright Expect:** These are the famous web-first assertions that help you to assert conditions linked with the DOM. Available in all supported languages, it can be used with different test frameworks. The TypeScript version is compatible with the expect library and usable with your preferred test runner: Jest, Vitest, Mocha, …
- **Playwright Test:** This is Playwright's test runner, which fetches and executes tests, creates reports, and handles parallelization and sharding. It's a great, modern test runner that is tailored for Playwright. This component is only available for TypeScript. For other languages, it's up to you to use a suitable test runner: JUnit, Pytest, MSTest, etc.
- **Codegen:** The code generator that lets you record a user path and generate the corresponding code for several languages, whether for Playwright Library or Playwright Test.
- **Trace Viewer:** It allows you to visualize the traces of an automation or a test. The tool is also available online.

Browsers

One important point to bear in mind is the browsers supported by Playwright. In fact, Playwright's core principle is to test the web, on the major browser families.

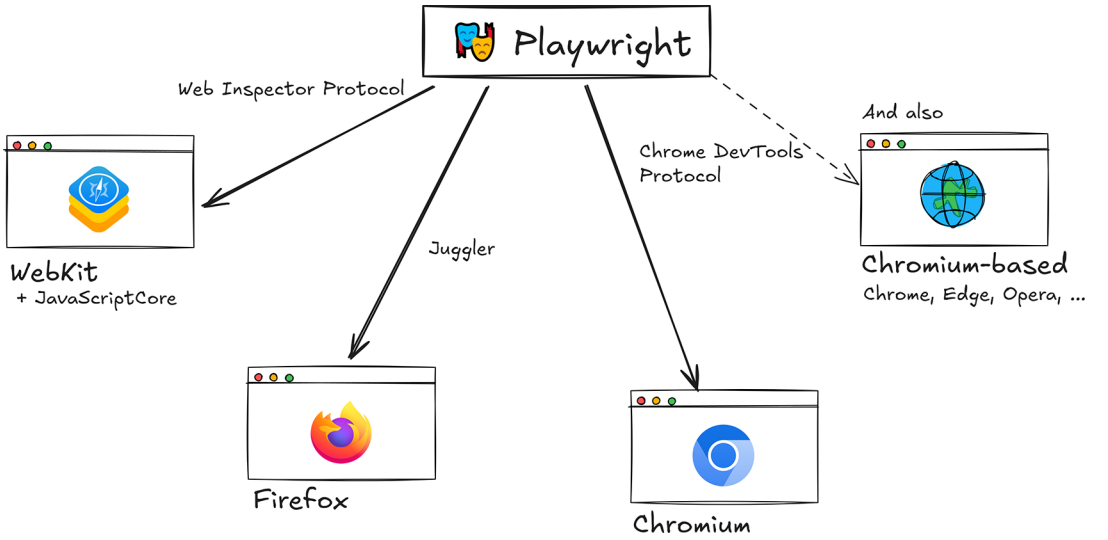

Figure 1-1. *Playwright communicates via browser debug protocol*

As shown in Figure 1-1, Playwright supports three major browsers:

- **Chromium:** Playwright supports Chromium, and browsers based on it: Chrome, Brave, Opera, Edge, ... This is made possible because Playwright uses the Chromium DevTools Protocol.[2]
- **Firefox** is supported via a patched build and the Juggler protocol.
- **WebKit** is a special case, which is here for a very specific reason. Apple Safari is only available on iPhone, iPad, and macOS. To run macOS, you need to run it on a Mac. What's more, it's not legally possible to virtualize macOS by yourself.

  So to have a browser that works on your machine (macOS, Linux, or Windows) and on your Continuous Integration, we're going to use the best alternative: a browser made with the same building bricks as Safari.

Note Playwright's build of WebKit can be used in other contexts where you need a Safari-like browser. For instance, Cypress uses it for WebKit support (experimental).

---

[2] Fun trivia: This very CDP protocol was developed by the same people as Playwright.

Thanks to its approach, Playwright can automate the three main browser engines. The main drawback is that the browsers are bound to the version of Playwright you are installing. For example, it is impossible to test an older version of Firefox with the latest Playwright.

Prepare Your Environment

Playwright is a Node.js application that comes with its own web browsers for automation. Remember, in this book, we talk about Playwright Test which means exclusively the TypeScript flavor of Playwright. It is also available for other languages: Python, Java, .Net, but this is not the subject of this book.

Prerequisites

You'll need an up-to-date version of Node.js, and the main operating systems are supported.

Here are the requirements as of today:[3]

- Node.js 20+
- Windows 10+, Windows Server 2016+, or Windows Subsystem for Linux (WSL)
- macOS 14 Ventura, or later
- Debian 12+, Ubuntu 22.04, Ubuntu 24.04, on x86-64 and arm64 architecture

I suggest you recommend the Long-Term Support version of Node.js (LTS). If you are doing frontend or backend in JavaScript, that's probably already the version you are using: the LTS version is the main version of Node.js, providing fixes to critical bugs for up to 30 months.

I highly recommend using nvm[4] to install Node.js. It will allow you to install automatically Node.js and npm, update them, and switch versions on the fly to suit your needs.

[3] See https://playwright.dev/docs/intro#system-requirements for up-to-date information

[4] nvm https://github.com/nvm-sh/nvm, or NVM for Windows https://github.com/coreybutler/nvm-windows which offers similar functionalities

Note Only Debian 12 and 13 and Ubuntu 22.04 and 24.04 are officially supported.

However, derivatives of those distros such as Raspbian, KDE Neon, Linux Mint… are fine. If you are using something else, you still can resort to Docker.

Depending on your operating system, you'll probably need a number of dependencies in order to run the browsers for our tests. Playwright downloads and installs for you its own browsers: Chromium, Firefox, and WebKit (like Safari).

On a fresh Debian image (Bookworm Slim), here's what you get if you don't install these dependencies but try to install the Playwright browsers.

Playwright Host validation warning:

```
╔═══════════════════════════════════════════════════════════════════╗
║ Host system is missing dependencies to run browsers.              ║
║ Missing libraries:                                                ║
║     libglib-2.0.so.0                                              ║
║     libgobject-2.0.so.0                                           ║
║     libnss3.so                                                    ║
║     libnssutil3.so                                                ║
║     libsmime3.so                                                  ║
║     libnspr4.so                                                   ║
║     libdbus-1.so.3                                                ║
║     libatk-1.0.so.0                                               ║
║     libatk-bridge-2.0.so.0                                        ║
║     libcups.so.2                                                  ║
║     libgio-2.0.so.0                                               ║
║     libdrm.so.2                                                   ║
║     libexpat.so.1                                                 ║
║     libxcb.so.1                                                   ║
║     libxkbcommon.so.0                                             ║
║     libatspi.so.0                                                 ║
║     libX11.so.6                                                   ║
║     libXcomposite.so.1                                            ║
║     libXdamage.so.1                                               ║
║     libXext.so.6                                                  ║
```

```
╔═══════════════════════════════════════════════════════════════════════╗
║     libXfixes.so.3                                                    ║
║     libXrandr.so.2                                                    ║
║     libgbm.so.1                                                       ║
║     libpango-1.0.so.0                                                 ║
║     libcairo.so.2                                                     ║
║     libasound.so.2                                                    ║
╚═══════════════════════════════════════════════════════════════════════╝
```

Fortunately, there's a command that will check these dependencies and install them if necessary. Note that this will ask for root access if needed.

```
npx playwright install-deps
```

If you don't want Playwright to install the dependencies automatically for you, you can make a dry run. In this way, you can inspect what will be installed, copy-paste, modify, and adapt the command to manage it yourself.

```
$ npx playwright install-deps --dry-run
sh -c "apt-get update&& apt-get install -y --no-install-recommends
libasound2 libatk-bridge2.0-0 libatk1.0-0 libatspi2.0-0 libcairo2 libcups2
libdbus-1-3 libdrm2 libgbm1 libglib2.0-0 libnspr4 libnss3 libpango-1.0-0
libx11-6 libxcb1 libxcomposite1 libxdamage1 libxext6 libxfixes3
libxkbcommon0 libxrandr2 xvfb fonts-noto-color-emoji fonts-unifont
libfontconfig1 libfreetype6 xfonts-scalable fonts-liberation fonts-ipafont-
gothic fonts-wqy-zenhei fonts-tlwg-loma-otf fonts-freefont-ttf libcairo-
gobject2 libdbus-glib-1-2 libgdk-pixbuf-2.0-0 libgtk-3-0 libharfbuzz0b
libpangocairo-1.0-0 libx11-xcb1 libxcb-shm0 libxcursor1 libxi6 libxrender1
libxtst6 libsoup-3.0-0 gstreamer1.0-libav gstreamer1.0-plugins-bad
gstreamer1.0-plugins-base gstreamer1.0-plugins-good libegl1 libenchant-2-2
libepoxy0 libevdev2 libgles2 libglx0 libgstreamer-gl1.0-0 libgstreamer-
plugins-base1.0-0 libgstreamer1.0-0 libgudev-1.0-0 libharfbuzz-icu0
libhyphen0 libicu72 libjpeg62-turbo liblcms2-2 libmanette-0.2-0 libnotify4
libopengl0 libopenjp2-7 libopus0 libpng16-16 libproxy1v5 libsecret-1-0
libwayland-client0 libwayland-egl1 libwayland-server0 libwebp7
libwebpdemux2 libwoff1 libxml2 libxslt1.1 libatomic1 libevent-2.1-7"
```

TL;DR

Prepare your Linux environment with this:

```
sudo apt update
sudo apt install curl
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.3/install.
sh | bash
nvm install --lts
nvm use --lts
npx playwright install-deps
```

npm, yarn, pnpm...

Quick reminder that the example scripts use `npm`, the node package manager that comes with Node.js. You can use others; the examples are all written with `npm` so as not to clutter the text.

A few equivalents of common package manager usage:

```
# init
npm init playwright
yarn create playwright
pnpm create playwright

# run a script that is defined by package.json
npm run hello
yarn hello
pnpm run hello

# run a binary from a package
npx playwright test
yarn playwright test
pnpm exec playwright test
```

Bun is an alternative runtime that is compatible with Node.js and does many more things. Playwright doesn't officially support bun, but the growing popularity of both solutions means that bun has made quite an effort to be able to run Playwright Test. Check their website to know more: https://bun.sh.

Here are a few hints for you to try bun for yourself:

```
# init
# npm init something == npx create-something
bunx create-playwright
# run a script that is defined by package.json
bun run hello
# run a binary from a package
bunx playwright test

# not working at the time of writing
bunx --bun playwright test
```

Check the Prerequisites

Now we can verify that we have everything set up: Node.js must return the current version, dependencies are installed.

```
node -v
npx playwright install-deps
```

We are now ready to initiate our first Playwright Test project.

Your First Test

Let's start by creating a new project:

```
mkdir playwright-example
cd playwright-example
npm init -y
```

We've just initialized an empty Node.js project; now let's install Playwright Test.

```
npm init playwright
```

`npm init playwright` will use another package behind the scenes, `create-playwright`, which is a wizard that will ask you a few questions to help you set up your test project. It's pretty straightforward, but two questions might puzzle you: which language to use and whether to install browsers.

At the question on using TypeScript or JavaScript, I advise selecting TypeScript. First, your tests don't have to be in the same language as the app being under tests. So, if your app is not using TypeScript, it's not a problem. Second, TypeScript is pretty awesome at catching issues while you write tests. If you are not familiar with it, I promise it's not too scary.

The wizard asks you if it should download browsers for you; you should always say yes as it will download browsers only if needed. Even better, Playwright will keep track of the browsers usage and will clean up browsers that are unused. For example, this happens when you update a Playwright Test installation and the browser version is not used by another project.

OK, let's take a look at what files the initialization script has added (Table 1-1).

Table 1-1. *Files added by initialization script*

File	
tests/example.spec.ts	Test file, which will be run with through Playwright Test
tests-examples/demo-todo-app.spec.ts	Example of a much more complete test on a TODO list application. Move this file to the tests folder or change the config to run it.
playwright.config.ts	Playwright Test's configuration file
package.json	
.gitignore	

Also note that the `@playwright/test` package is installed as a dev dependency and that the `.gitignore` file is updated.

Note If you're working with Angular, there is a schematic for Playwright.

This community Playwright Test integration with Angular is the one featured when you run `ng e2e` and select Playwright.

In addition to initializing Playwright Test, it will configure your project to run an Angular development server. Note that you can also use `npm init playwright` without any drawbacks.[5]

[5] Trust me, I'm the author of Playwright Angular Schematic.

We'll now check the installation and run the tests:

```
npx playwright test
```

Let's take a look at the test file we just ran.

Listing 1-1. example.spec.ts

```
import { test, expect } from "@playwright/test"; //1

test("has title", async ({ page }) => { //2
  await page.goto("https://playwright.dev/");

  // Expect a title "to contain" a substring.
  await expect(page).toHaveTitle(/Playwright/);
});

test("get started link", async ({ page }) => {
  await page.goto("https://playwright.dev/");

  // Click the get started link.
  await page.getByRole("link", { name: "Get started" }).click(); // 3

  // Expects page to have a heading with the name of Installation.
  await expect(
    page.getByRole("heading", { name: "Installation" })
  ).toBeVisible(); // 4
});
```

Some important points (see inline comments in Listing 1-1):

1. The test functions must be explicitly imported. `test` or `expect` are not global.
2. The test syntax is fairly close to other popular test frameworks, such as Jest.
3. The API allows you to perform actions.
4. And assertions.