

[Skip to main content](#)



## [Playwright Docs API](#)

### [Node.js](#)

- [Node.js](#)
- [Python](#)
- [Java](#)
- [.NET](#)

## [Community](#)

Search 

- [Getting Started](#)
  - [Installation](#)
  - [Writing tests](#)
  - [Generating tests](#)
  - [Running and debugging tests](#)
  - [Trace viewer](#)
  - [Setting up CI](#)
- [Getting started - VS Code](#)
- [Release notes](#)
- [Canary releases](#)
- [Playwright Test](#)
  - [Test configuration](#)
  - [Test use options](#)
  - [Annotations](#)
  - [Command line](#)

- [Emulation](#)
- [Fixtures](#)
- [Global setup and teardown](#)
- [Parallelism](#)
- [Parameterize tests](#)
- [Projects](#)
- [Reporters](#)
- [Retries](#)
- [Sharding](#)
- [Timeouts](#)
- [TypeScript](#)
- [UI Mode](#)
- [Web server](#)
- [Guides](#)
  - [Library](#)
  - [Accessibility testing](#)
  - [Actions](#)
  - [Assertions](#)
  - [API testing](#)
  - [Authentication](#)
  - [Auto-waiting](#)
  - [Best Practices](#)
  - [Browsers](#)
  - [Chrome extensions](#)
  - [Clock](#)
  - [Components \(experimental\)](#)
  - [Debugging Tests](#)
  - [Dialogs](#)
  - [Downloads](#)
  - [Evaluating JavaScript](#)
  - [Events](#)
  - [Extensibility](#)
  - [Frames](#)
  - [Handles](#)
  - [Isolation](#)
  - [Locators](#)
  - [Mock APIs](#)
  - [Mock browser APIs](#)
  - [Navigations](#)
  - [Network](#)
  - [Other locators](#)
  - [Page object models](#)
  - [Pages](#)
  - [Screenshots](#)
  - [Visual comparisons](#)
  - [Test generator](#)
  - [Trace viewer](#)
  - [Videos](#)
  - [WebView2](#)
- [Migration](#)

- [Integrations](#)
- [Supported languages](#)
- 
- Getting Started
- Running and debugging tests

On this page

# Running and debugging tests

## Introduction

With Playwright you can run a single test, a set of tests or all tests. Tests can be run on one browser or multiple browsers by using the `--project` flag. Tests are run in parallel by default and are run in a headless manner, meaning no browser window will be opened while running the tests and results will be seen in the terminal. However, you can run tests in headed mode by using the `--headed` CLI argument, or you can run your tests in [UI mode](#) by using the `--ui` flag. See a full trace of your tests complete with watch mode, time travel debugging and more.

### You will learn

- [How to run tests from the command line](#)
- [How to debug tests](#)
- [How to open the HTML test reporter](#)

## Running tests

### Command line

You can run your tests with the `playwright test` command. This will run your tests on all browsers as configured in the `playwright.config` file. Tests run in headless mode by default meaning no browser window will be opened while running the tests and results will be seen in the terminal.

```
npx playwright test
```



A screenshot of a terminal window titled '-zsh'. The window shows the output of a command, starting with 'Running 6 tests using 5 workers' and '6 passed (3.8s)'. Below this, instructions are given to 'To open last HTML report run:' followed by the command 'npx playwright show-report'. The terminal has a dark background with light-colored text.

```
Running 6 tests using 5 workers
6 passed (3.8s)

To open last HTML report run:
npx playwright show-report
```

## Run tests in UI mode

We highly recommend running your tests with [UI Mode](#) for a better developer experience where you can easily walk through each step of the test and visually see what was happening before, during and after each step. UI mode also comes with many other features such as the locator picker, watch mode and more.

```
npx playwright test --ui
```

 **PLAYWRIGHT** 0

> Filter (e.g. text, @tag)

Status: all Projects: chromium

1/1 passed (100%) Actions

example.spec.ts Before

has title page.s

get started link locator expected After

Actions

> Before

page.s

locator

expected

> After

Check out or [detailed guide on UI Mode](#) to learn more about it's features.

## Run tests in headed mode

To run your tests in headed mode, use the `--headed` flag. This will give you the ability to visually see how Playwright interacts with the website.

```
npx playwright test --headed
```

## Run tests on different browsers

To specify which browser you would like to run your tests on, use the `--project` flag followed by the name of the browser.

```
npx playwright test --project webkit
```

To specify multiple browsers to run your tests on, use the `--project` flag multiple times followed by the name of each browser.

```
npx playwright test --project webkit --project firefox
```

## Run specific tests

To run a single test file, pass in the name of the test file that you want to run.

```
npx playwright test landing-page.spec.ts
```

To run a set of test files from different directories, pass in the names of the directories that you want to run the tests in.

```
npx playwright test tests/todo-page/ tests/landing-page/
```

To run files that have `landing` or `login` in the file name, simply pass in these keywords to the CLI.

```
npx playwright test landing login
```

To run a test with a specific title, use the `-g` flag followed by the title of the test.

```
npx playwright test -g "add a todo item"
```

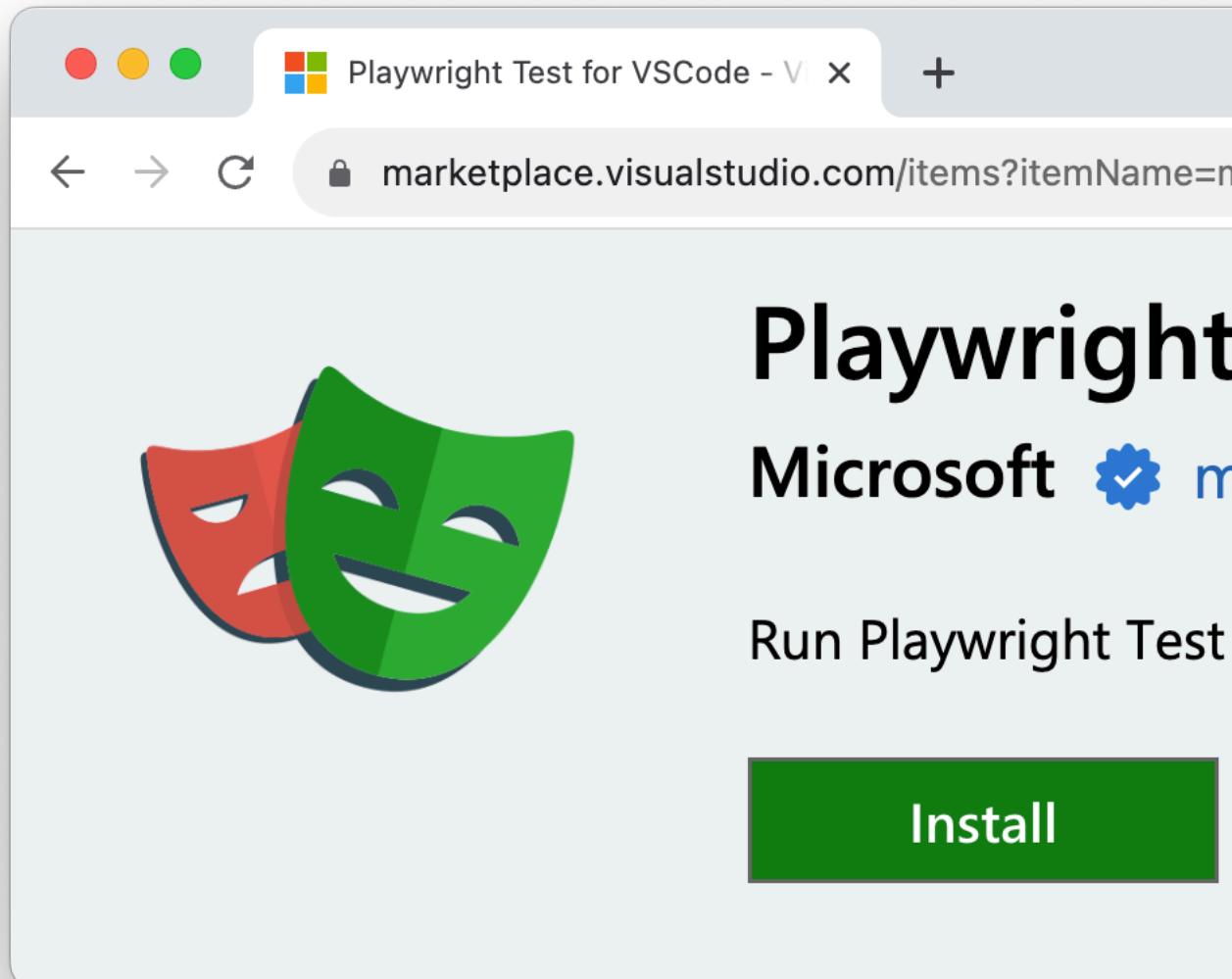
## Run last failed tests

To run only the tests that failed in the last test run, first run your tests and then run them again with the `--last-failed` flag.

```
npx playwright test --last-failed
```

## Run tests in VS Code

Tests can be run right from VS Code using the [VS Code extension](#). Once installed you can simply click the green triangle next to the test you want to run or run all tests from the testing sidebar. Check out our [Getting Started with VS Code](#) guide for more details.



## Debugging tests

Since Playwright runs in Node.js, you can debug it with your debugger of choice e.g. using `console.log` or inside your IDE or directly in VS Code with the [VS Code Extension](#). Playwright comes with [UI Mode](#), where you can easily walk through each step of the test, see logs, errors, network requests, inspect the DOM snapshot and more. You can also use the [Playwright Inspector](#), which allows you to step through Playwright API calls, see their debug logs and explore [locators](#).

## Debug tests in UI mode

We highly recommend debugging your tests with [UI Mode](#) for a better developer experience where you can easily walk through each step of the test and visually see what was happening before, during and after each step. UI mode also comes with many other features such as the locator picker, watch mode and more.

```
npx playwright test --ui
```

 PLAYWRIGHT

Filter (e.g. text, @tag)

Status: failed Projects: chromium

passed     chromium

failed     firefox

skipped     webkit

21/26 passed (80%)

Actions

> Before

locator

locator

expect

> After

demo-todo-app.spec.ts

> Item

> Editing

> Counter

> Clear completed button

should remove co...

While debugging you can use the Pick Locator button to select an element on the page and see the locator that Playwright would use to find that element. You can also edit the locator in the locator playground and see it highlighting live on the Browser window. Use the Copy Locator button to copy the locator to your clipboard and then paste it into your test.

PLAYWRIGHT

Filter (e.g. text, @tag)

Status: all Projects: chromium

26/26 passed (100%)

Actions

> Before

locator

locator

locator

locator

locator

locator

expect

expect

locator

expect

expect

locator

expect

expect

> After

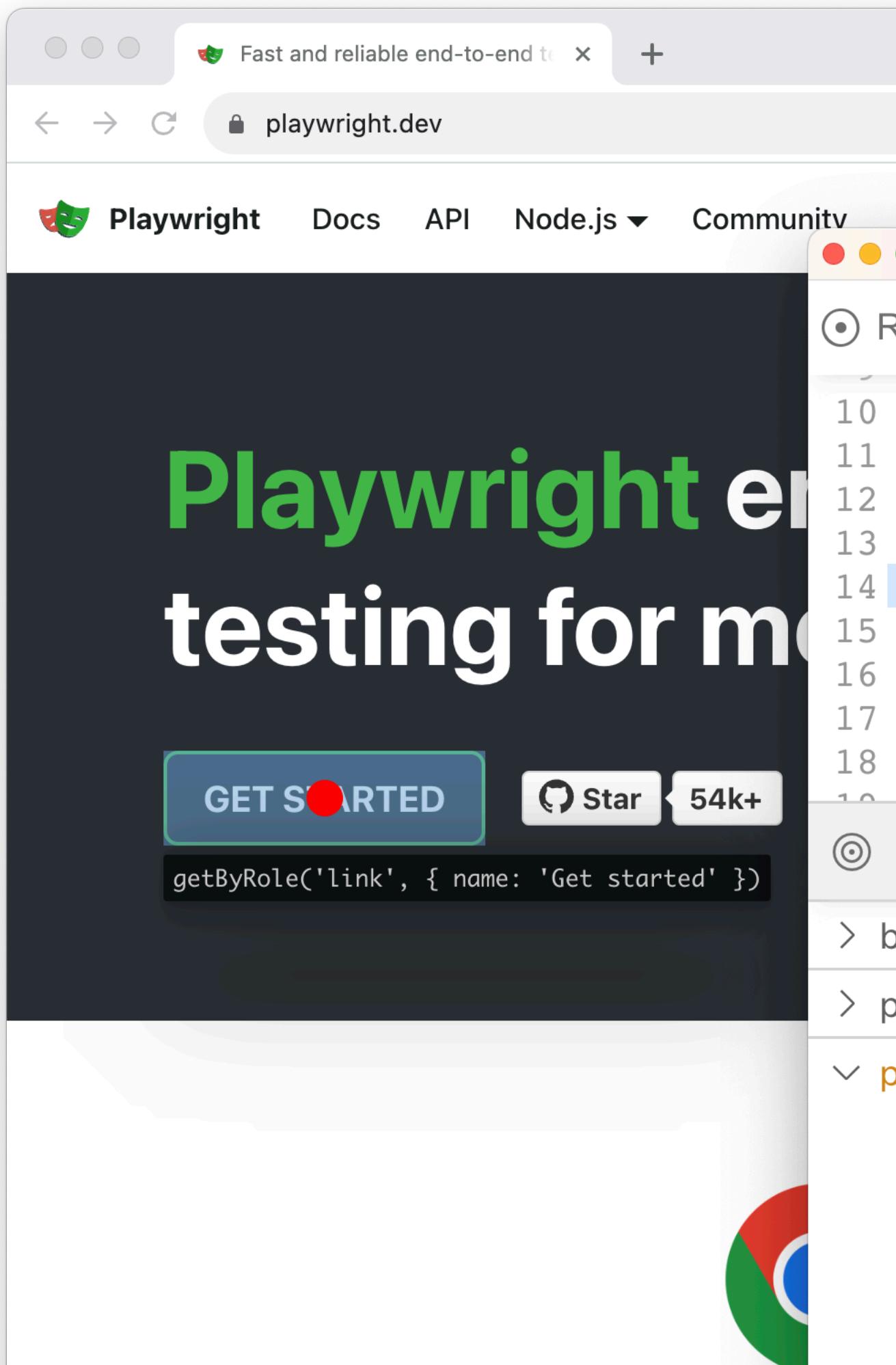
- ✓ demo-todo-app.spec.ts
  - ✓ New Todo
    - ✓ should allow me to add a new todo item 896ms
    - ✓ should clear text from the input field 839ms
    - ✓ should append new todos to the list 927ms
  - > ✓ Mark all as completed
  - ✓ Item
    - ✓ should allow me to mark a todo item as completed 740ms
    - ✓ should allow me to unmark a todo item as completed 708ms
  - > ✓ Editing
  - > ✓ Counter
  - > ✓ Clear completed button

Check out our [detailed guide on UI Mode](#) to learn more about it's features.

## Debug tests with the Playwright Inspector

To debug all tests, run the Playwright test command followed by the `--debug` flag.

```
npx playwright test --debug
```



This command will open up a Browser window as well as the Playwright Inspector. You can use the step over button at the top of the inspector to step through your test. Or, press the play button to run your test from start to finish. Once the test has finished, the browser window will close.

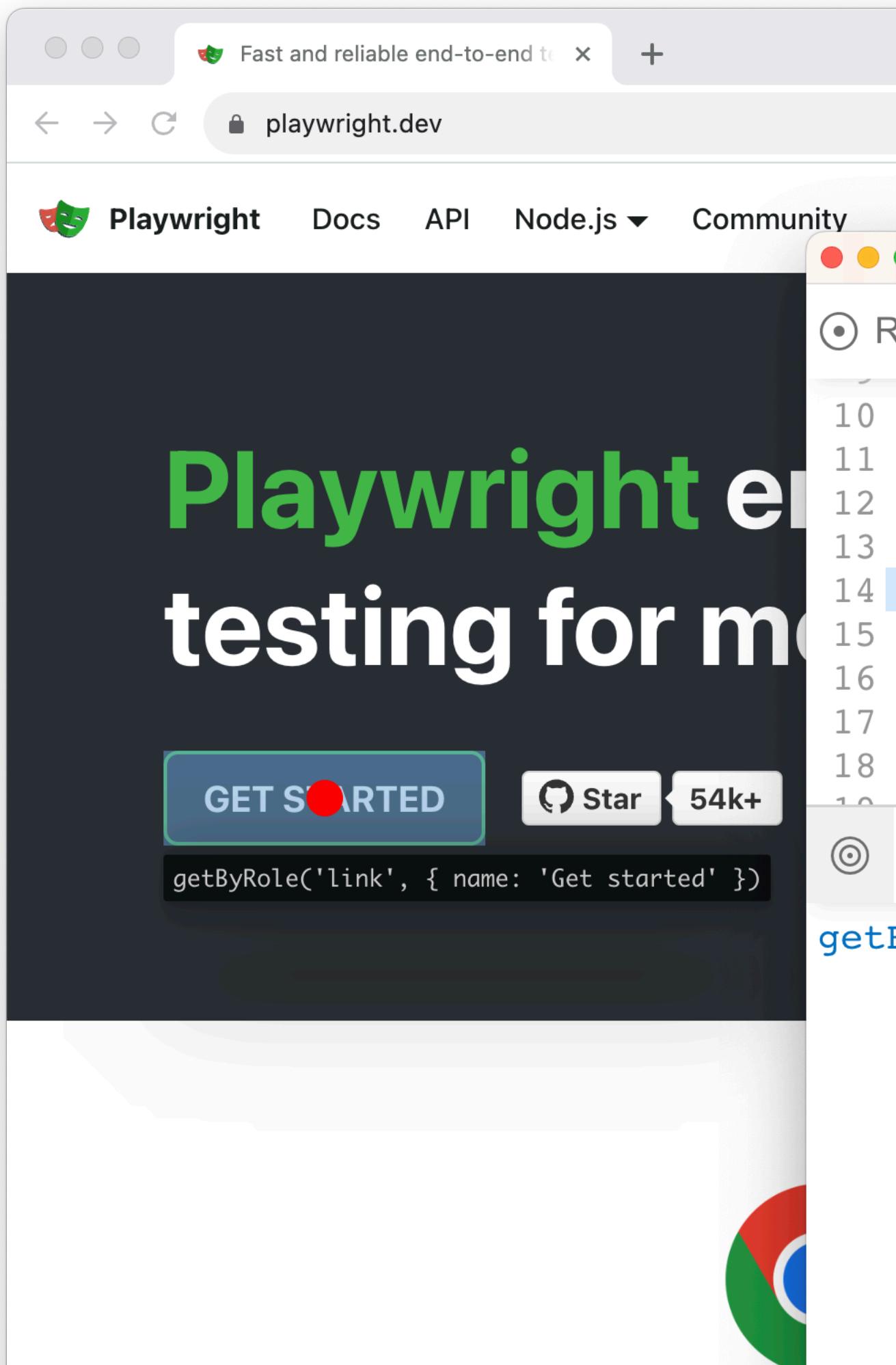
To debug one test file, run the Playwright test command with the name of the test file that you want to debug followed by the `--debug` flag.

```
npx playwright test example.spec.ts --debug
```

To debug a specific test from the line number where the `test(..)` is defined, add a colon followed by the line number at the end of the test file name, followed by the `--debug` flag.

```
npx playwright test example.spec.ts:10 --debug
```

While debugging you can use the Pick Locator button to select an element on the page and see the locator that Playwright would use to find that element. You can also edit the locator and see it highlighting live on the Browser window. Use the Copy Locator button to copy the locator to your clipboard and then paste it into your test.



Check out our [debugging guide](#) to learn more about debugging with the [VS Code debugger](#), UI Mode and the [Playwright Inspector](#) as well as debugging with [Browser Developer tools](#).

## Test reports

The [HTML Reporter](#) shows you a full report of your tests allowing you to filter the report by browsers, passed tests, failed tests, skipped tests and flaky tests. By default, the HTML report is opened automatically if some of the tests failed, otherwise you can open it with the following command.

```
npx playwright show-report
```

The screenshot shows a web-based test report for a file named `example.spec.ts`. The interface has a header with window controls (red, yellow, green circles), a title "Playwright Test Report", and a URL "localhost:9323". Below the header is a search bar with a magnifying glass icon. The main content area displays a list of test cases, each with a status indicator (green checkmark for pass, red X for fail), the test name, a browser-specific button, and the file path.

- ✓ **has title** chromium example.spec.ts:3
- ✓ **has title** firefox example.spec.ts:3
- ✓ **has title** webkit example.spec.ts:3
- ✗ **get started link** chromium example.spec.ts:10
- ✗ **get started link** firefox example.spec.ts:10
- ✗ **get started link** webkit example.spec.ts:10

You can filter and search for tests as well as click on each test to see the tests errors and explore each step of the test.

Playwright Test Report

localhost:9323/#?testId=a30a6eba6312f6b87ea5

get started link

example.spec.ts:10

chromium

Run

Errors

Test timeout of 30000ms exceeded.

```
Error: locator.click: Page closed
=====
waiting for getByRole('link', { name: 'Intro' })
=====

12 |
13 |     // Click the get started link.
> 14 |     await page.getByRole('link', { name: 'I
15 |
16 |     // Expects the URL to contain intro.
17 |     await expect(page.getByRole('heading', { name: 'Intro' }))
```

Pending operations:

# What's next

- [Generate tests with Codegen](#)
- [See a trace of your tests](#)
- [Explore all the features of UI Mode](#)
- [Run your tests on CI with GitHub Actions](#)

[Previous](#)

[Generating tests](#)

[Next](#)

[Trace viewer](#)

- [Introduction](#)
- [Running tests](#)
  - [Command line](#)
  - [Run tests in UI mode](#)
  - [Run tests in headed mode](#)
  - [Run tests on different browsers](#)
  - [Run specific tests](#)
  - [Run last failed tests](#)
  - [Run tests in VS Code](#)
- [Debugging tests](#)
  - [Debug tests in UI mode](#)
  - [Debug tests with the Playwright Inspector](#)
- [Test reports](#)
- [What's next](#)

Learn

- [Getting started](#)
- [Playwright Training](#)
- [Learn Videos](#)
- [Feature Videos](#)

Community

- [Stack Overflow](#)
- [Discord](#)
- [Twitter](#)
- [LinkedIn](#)

More

- [GitHub](#)
- [YouTube](#)
- [Blog](#)

- [Ambassadors](#)

Copyright © 2024 Microsoft