

# Mastering Application Testing Scaling and Continuous UI Testing with Playwright



Let's  
Connect!



-  Randy-Pagels
-  PagelsR
-  <https://xebia.com>
-  @RandyPagels

Randy Pagels  
DevOps Architect | Trainer  
[Randy.Pagels@Xebia.com](mailto:Randy.Pagels@Xebia.com)





16th Annual  
**Orlando**  
**Code Camp** | **2024**

Sponsors, Organizers, and Partners



**Xebia** **ZERØTRUSTED.AI**



**SNI TECHNOLOGY**



 **TEKsystems**  
Own change

**CODE**



Google Developer Groups  
Central Florida

 **SQLOrlando**

# Agenda

- Testing Challenges & Goals
- Playwright Overview
- Getting Started
- Authoring, Debugging, Running
- Additional Built in Features
- Time Travel Experience
- Azure Test Plans
- Continuous Integration Testing

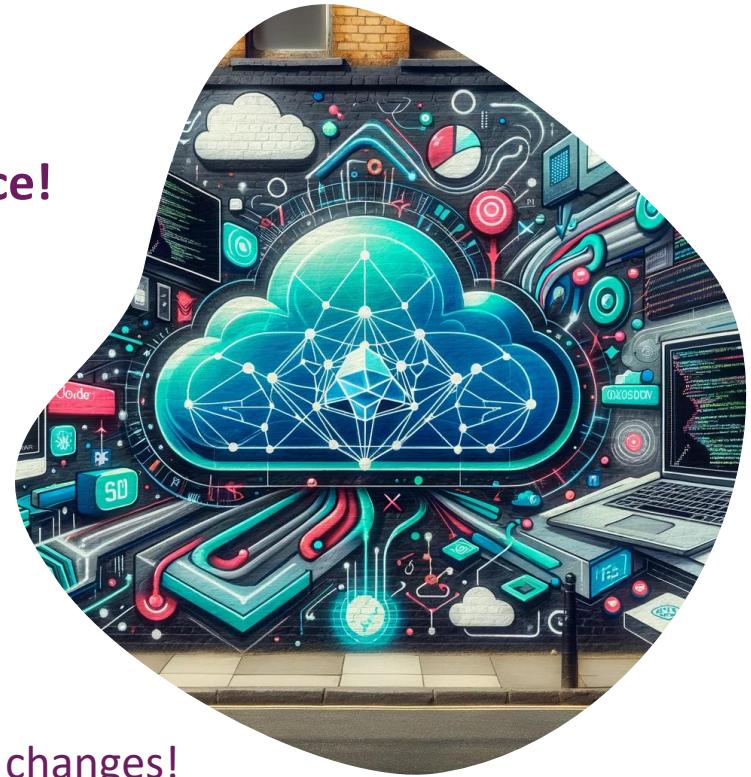


# UI Testing Challenges

**Consistent design across browsers!**

**Dynamic content and maintaining performance!**

- Testing is hard
- Testing takes time to learn
- Testing takes time to build
- Tests are slow – they take too long to run!
- Tests are flaky – they crash all the time!
- Tests are brittle – they break whenever the app changes!



# Modern Testing Goals

## Major goals for modern web testing

- Make testing easy!
- Make testing fun!
- Focus on fast feedback loops!
- Make test development as painless as possible!
- Testing tool that complements dev workflows!



# Test Automation Practices

Tests should be automated when they give positive ROI

## Advantages

- Tests can be **rerun at any time** the same way.
- Tests can run as **part of CI/CD**.
- **Setup and cleanup** can be **automated**.
- Frameworks can provide reports, logs, and screenshots.

## ProTips!

- Each **test** should **focus** on **one main behavior** or variation.
- **Tests** should **run independently** of each other and in any order.
- **Tests** should be **self-descriptive** and intuitively understandable.
- Do not automate every test – use a risk-based strategy with ROI.
- **Tests** should **run in parallel** at **scale**, especially in CI/CD.



# AN INTRODUCTION TO PLAYWRIGHT



WHAT IS  
PLAYWRIGHT?

- AN OPEN SOURCE TESTING FRAMEWORK
- ENABLES RELIABLE END-TO-END TESTING AND AUTOMATION

FOR MODERN WEB APPLICATIONS

- ★ TOOLING
- ★ AUTOMATION
- ★ API

CHECK OUT THE SOURCE

<https://github.com/microsoft/playwright>

5  
RESOURCES TO HELP YOU JUMPSTART YOUR TEST ADVENTURE

1 READ THE DOCS  
<https://playwright.dev/docs/>

2 EXPLORE THE API  
<https://playwright.dev/api/>

3 VISIT THE REPO  
<https://github.com/microsoft/playwright>

4 FOLLOW @PLAYWRIGHT  
[@playwrightweb](#)

5 CHECKOUT DEMO CODE  
<https://aka.ms/playwright-demo>

POWERFUL  
TOOLING!

1 ONE CONCERN



- CREATE A BROWSER CONTEXT PER TEST IN JUST MS.
- FULL TEST ISOLATION (ZERO OVERHEAD)
- LESS COMPLEXITY FOR TESTER

FULL ISOLATION  
FAST EXECUTION

2 INSPECTOR



- INSPECT PAGE LOGS & DEBUGGING SILENTLY
- STEP THROUGH EXEC & SEE CLICK POINTS

3 TRACE VIEWER



- CAPTURE TEST PROCESSES (STATE, FAILURES, TIMING)
- SCREENSHOTS, SNAPPLOTS, ACTION EXPLORER, TEST SOURCE (POST-MORTEM ANALYSIS)

WHY SHOULD I BE USING PLAYWRIGHT?

- 1 ONE API, MULTI-EVERYTHING
- 2 RESILIENT, NO FLAKY TESTS
- 3 LIMITLESS, NO TRADEOFFS
- 4 ISOLATED, FAST EXECUTION
- 5 TOOLING, CREATE-DEBUG-ANALYZE

1 ONE API  
MULTI  
EVERYTHING

2 RESILIENT!  
NO FLAKY TESTS

3 NO TRADE-OFFS  
LIMITS

TESTING API  
TEST RUNNER API  
TEST CONFIG OPTIONS

@playwrightweb  
<https://playwright.dev>

ANY BROWSER  
ANY PLATFORM  
Chromium, Webkit, Firefox, Native, Linux, Windows

CROSS LANGUAGE  
JavaScript, TypeScript, Python, Java, .NET - Libraries

TEST MOBILE WEB  
Native emulation  
Chrome on Android  
Mobile Safari on iOS

CROSS BROWSER TESTING  
Same Rendering Engine on Cloud and Desktop

TEST AUTOMATION AT CLOUD SCALE

REDUCE TEST COMPLEXITY  
INCREASE SCALABILITY + RELIABILITY

ILLUSTRATED BY @SKETCHTHEDOCS

# Playwright Overview

Open-source web test automation library on Node.js, which makes test automation easier for browsers based on Chromium, Firefox, and Webkit through a single API.

- ✓ **Cross-Browser**
    - Chromium, Firefox, and WebKit
  - ✓ **Cross-language**
    - JavaScript, Python, and C#
  - ✓ **Cross-platform**
    - Windows, Linux, and macOS
  - ✓ **Modern Asynchronous API**
    - `async/await`
  - ✓ **Rich Element Interaction**
    - wide range of built-in functions
  - ✓ **Screenshots and Video Recording**
    - visual evidence of issues
  - ✓ **Full Isolation – Fast Execution**
    - separate browser contexts
- 
- Reliable end-to-end testing for modern web apps  
...by Microsoft  
...and the people who made Puppeteer
- <https://playwright.dev/>
- <https://github.com/microsoft/playwright>

# Playwright Architecture

- ✓ Language Bindings → 
- ✓ Single Automation Protocol → 
- ✓ (Native) browser debugging protocols, e.g. ChromeDevTools Protocol, Safari Web Inspector → 

Playwright

OS: Windows, Linux, macOS

# Getting Started

## Setting up using VS Code

### Install Extension

### Install Playwright

- npm init playwright@latest

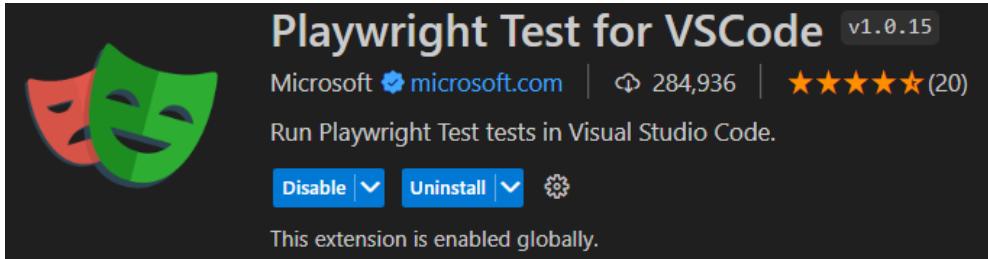
### Run Tests

- npx playwright test
- npx playwright test --ui

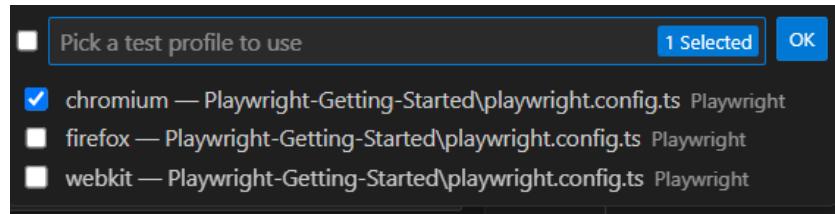
### Test Reports

- npx playwright show-report

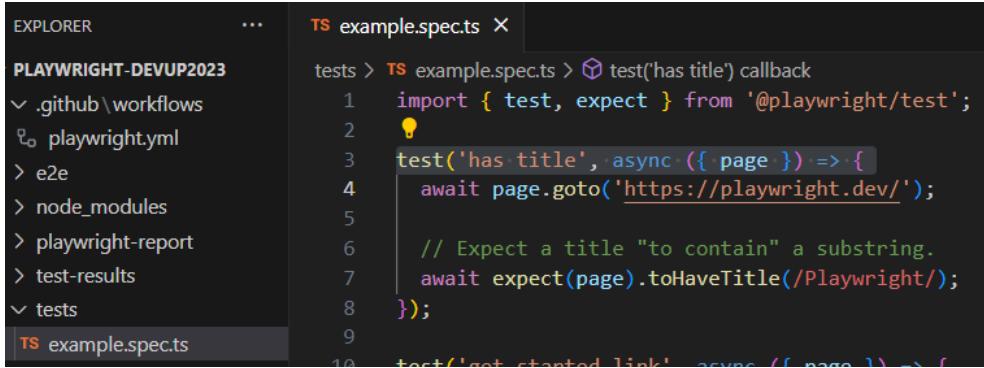
1



2



3



# Test Generator

Playwright Test Generator is a GUI tool that records actions performed in the browser and generates code.

## CLI

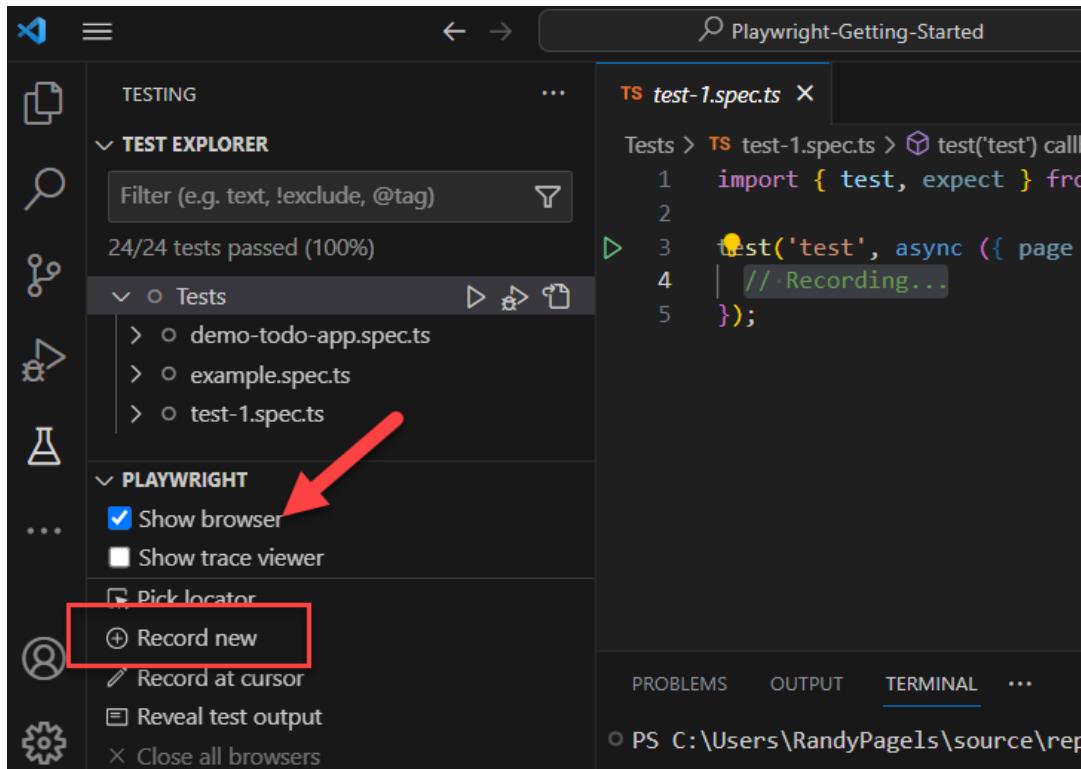
- npx playwright codegen

## VS Code

- Use Plugin, "Record Now"

With the test generator you can:

- Record Actions
- Assert Elements
- Create Locators
- Emulate Devices



# Trace Viewer

Playwright Trace Viewer is a GUI tool that helps you explore recorded Playwright traces after the script has ran.

## CLI

- `npx playwright test test.ts --trace on`

## VS Code

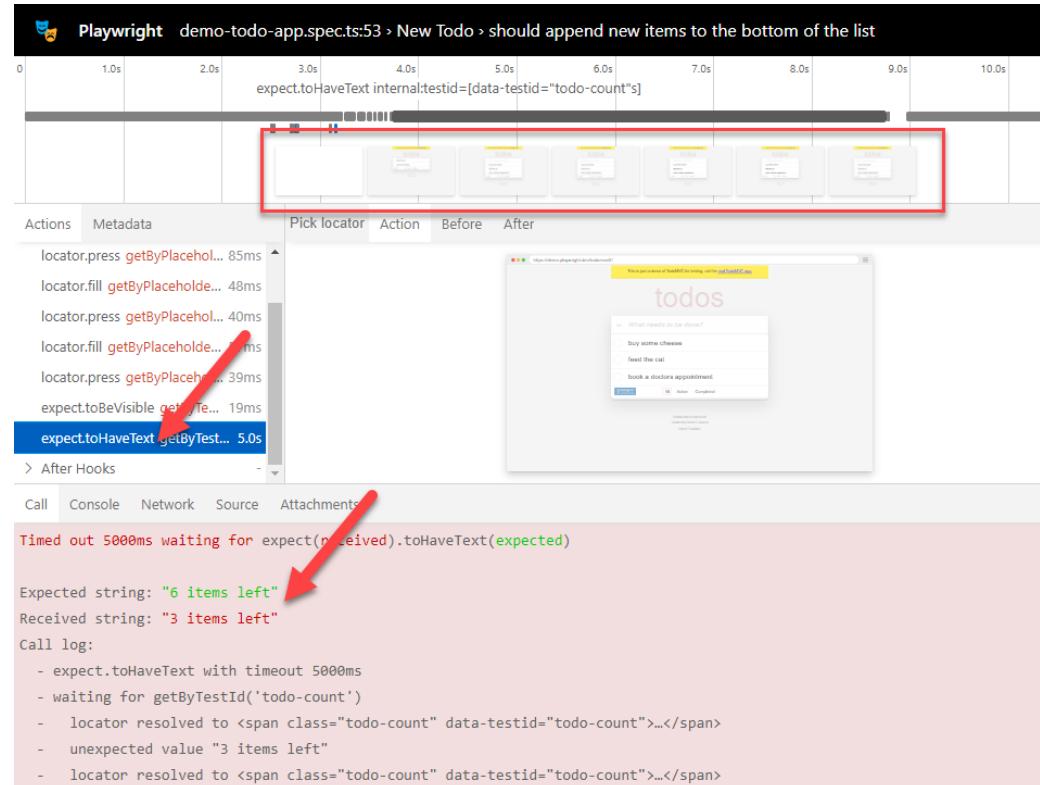
- Use Plugin

## Hosted Trace File Viewer

- <https://trace.playwright.dev>



Playwright Trace Viewer is a Progressive Web App, it does not send  
your trace anywhere, it opens it locally.



# Report List

Playwright Test comes with a few built-in reporter.

## HTML Report

- npx playwright test --reporter=html

## List Report

- npx playwright test --reporter=list

## Line Report

- npx playwright test --reporter=line

## Dot Report

- npx playwright test --reporter=dot

The screenshot shows the Playwright Test reporting interface with three numbered callouts:

- 1**: Shows the HTML Report view. At the top, there's a search bar and a summary bar with buttons for All (24), Passed (23), Failed (1), Flaky (0), and Skipped (0). A red arrow points to the Failed button. Below this, it says "Project: chromium Total time: 12.0s". The main area lists test cases under "demo-todo-app.spec.ts":
  - ✗ New Todo › should append new items to the bottom of the list (demo-todo-app.spec.ts:53) - Failed (7.2s)
  - ✓ New Todo › should allow me to add todo items (demo-todo-app.spec.ts:14) - Passed (1.3s)
  - ✓ New Todo › should clear text input field when an item is added (demo-todo-app.spec.ts:40) - Passed (1.4s)
- 2**: Shows the List Report view. It displays the command "Running 6 tests using 6 workers" followed by a list of 6 test cases, each with a green checkmark and the browser name [chromium] or [firefox]:
  - ✓ 1 [chromium] › demo-todo-app.spec.ts:14:7 > New Todo ›
  - ✓ 2 [chromium] › demo-todo-app.spec.ts:53:7 > New Todo ›
  - ✓ 3 [firefox] › demo-todo-app.spec.ts:14:7 > New Todo ›
  - ✓ 4 [firefox] › demo-todo-app.spec.ts:40:7 > New Todo ›
  - ✓ 5 [chromium] › demo-todo-app.spec.ts:40:7 > New Todo ›
  - ✓ 6 [firefox] › demo-todo-app.spec.ts:53:7 > New Todo ›At the bottom, it says "6 passed (11.1s)".
- 3**: Shows the Dot Report view. It displays the command "Running 6 tests using 6 workers" followed by a series of dots "....." and the message "6 passed (11.1s)".

# Playwright UI Mode

Explore, run and debug tests with a time travel experience complete with watch mode

## CLI

```
npx playwright test testname.ts --ui
```

## DOM Inspection

- page structure, inspect elements, view attributes.

## Console Logs

- JavaScript-generated logs, errors, and warnings.

## Network Monitoring

- Requests, timings, inspect headers, payloads.

## Interactivity

- Trigger events, clicks, real-time page responses.

## Element Highlighting

- Hover over elements to highlight them.

## Source Code Access

- View HTML, CSS, JS source for debugging.

The screenshot shows the Playwright UI Mode interface. At the top, there's a timeline from 0 to 20.0s. Below it, a tree view shows a test file 'demo-todo-app.spec.ts' with a 'New Todo' section expanded, containing three green checkmarks indicating passed tests. A red box highlights this section. To the right, a preview window shows a todo list with items like 'buy some cheese', 'feed the cat', and 'book a doctors appointment'. Below the tree view, tabs for Source, Console, Network, Call, and Attachments are visible. The Source tab shows the test code:

```
0  'buy some cheese',
1  'feed the cat',
2  'book a doctors appointment'
3  ];
4  [
5  ];
6  [
7  ];
8  [
9  ];
10 [
11 ];
12 [
13 test.describe('New Todo', () => {
14   test('should allow me to add todo items', async ({ page })
15     // create a new todo locator
```

A red arrow points to the line 'test('should allow me to add todo items', async ({ page })' in the source code, highlighting the current test being run.

# Demo

- ✓ Getting Started
- ✓ Generate Tests
- ✓ Debugging
- ✓ Trace Viewer
- ✓ Test Report
- ✓ UI Mode – Visual Feedback



# Using Playwright with Azure Test Plans



# Azure Test Plans with Playwright

Publish your automated Playwright test results to Azure Test Plans

- ✓ Associate automated tests with Test Cases!
- ✓ Using Azure Test Plans with Playwright!
- **End-to-End Traceability:** Requirements (PBI's, User Stories) in Azure Boards are linked to test cases.
- **User-Friendly Testing:** Pipelines enabling the triggering of automated tests through Test Cases, rather than pipelines.
- **History Access:** Azure Test Plans offers access to historical test data, simplifying progress tracking through reports and charts.
- **Test Inventory Management:** Both automated and manual tests are managed efficiently, enabling effective tracking of their status and the progress of automation efforts.

# Azure Test Case Setup

## Publish your automated Playwright tests to the ADO service

## Manually create new test cases in Azure Test Plans

1. Create manual Test Cases
  2. Add the **Test Case ID** in brackets to the test case title
  3. Publish test results from build pipeline

The screenshot shows the Microsoft Test Plan interface. On the left, a sidebar displays the 'Test Suites' section with a dropdown for 'eShopOnWeb2024'. Below it are several collapsed test cases: 'Login and view account', '11149 : Login and new user re...', '11159 : Shop by Brand on the...', '11160 : Shop by Type on the ...', and '11151 : Change the background...'. The main area is titled '11159 : Shop by Brand on the eShopOnWeb portal (ID: 11161)'. It has tabs for 'Define', 'Execute' (which is selected), and 'Chart'. Under 'Test Points (6 items)', there is a table:

Title	Outcome	Order	Test Case Id
<input checked="" type="checkbox"/> Shop by Brand - All	Active	1	11185
<input checked="" type="checkbox"/> Shop by Brand - .Net	Active	2	11186
<input type="checkbox"/> Shop by Brand - Azure	Active	3	11187
<input type="checkbox"/> Shop by Brand - Other	Active	4	11188
<input type="checkbox"/> Shop by Brand - SQL Server	Active	5	11189
<b>Test Plan ID</b> My Brand - Visual Studio	Active	6	11190



```
7
8 test.describe('Header', () => {
9
10 > test('[11185] Shop by Brand - All', async ({ pa
28 });
29
30 test('[11186] Shop by Brand - .Net', async ({ p
31 | await expect(page).toHaveTitle('Catalog - Mic
32 |
```

3

```
1 Starting: publish test results
2 =====
3 Task       : Publish Test Results
4 Description : Publish test results to Azure Pipeli
5 Version    : 2.233.1
6 Author     : Microsoft Corporation
7 Help       : https://docs.microsoft.com/azure/dev
8 =====
9 Result Attachments will be stored in LogStore
10 Run Attachments will be stored in LogStore
11 Async Command Start: Publish test results
12 Publishing test results to test run '1016595'.
13 TestResults To Publish 47, Test run id:1016595
14 Test results publishing 47, remaining: 0. Test run
15 Published Test Run : https://dev.azure.com/xpirit/e
16 Async Command End: Publish test results
```

# Test Plans Progress Report

eShopOnWeb2024\_TestPlan1

Test Suites

Outcome

Configuration

Tester

Priority

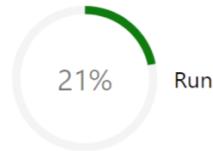


## Summary

1  
Test plans

23  
Test points

5 (5 / 23)  
Test points run



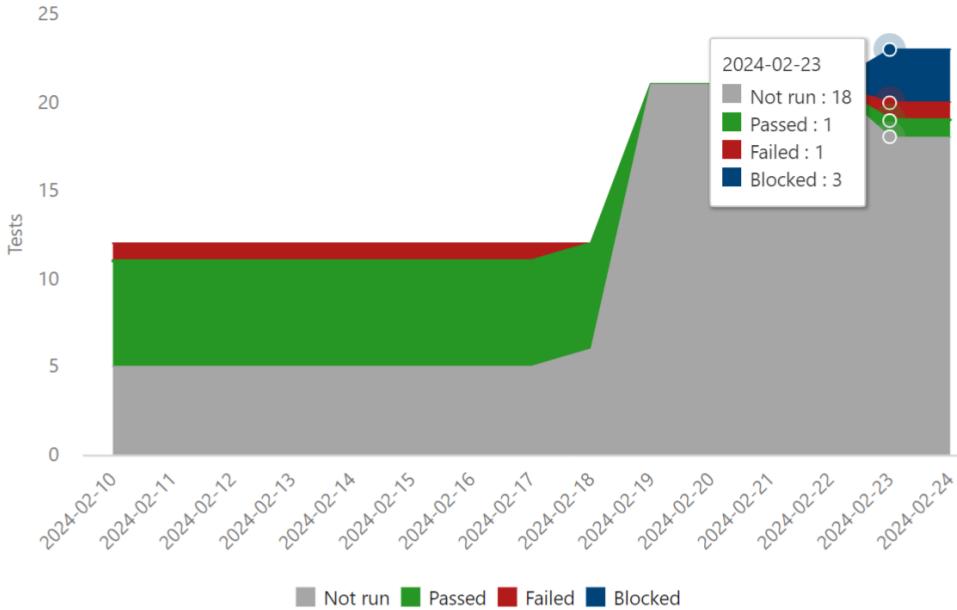
✓ 20% (1 / 5)  
Pass rate



1 Passed  
1 Failed  
3 Blocked

## Outcome trend

Last 14 Days



# Demo

- ✓ Test Cases using Playwright
- ✓ Test Runs
- ✓ Progress Report



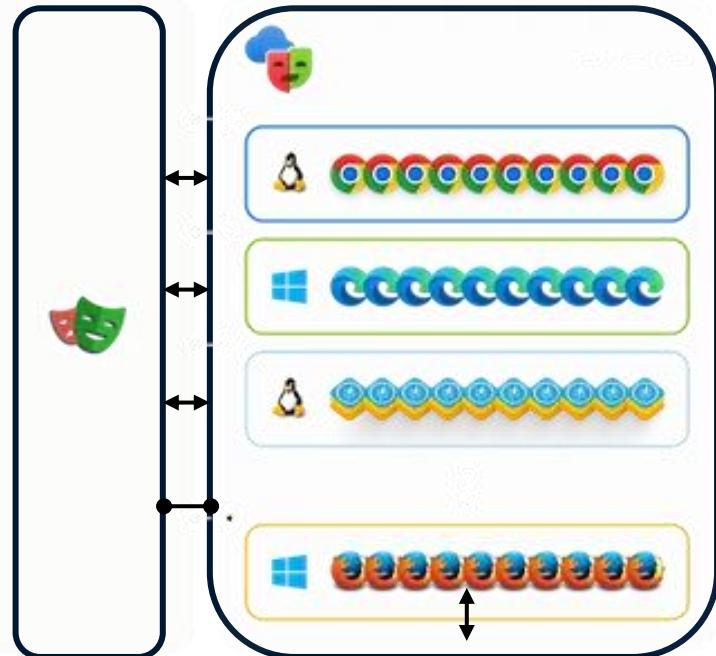
# Continuous Integration Testing

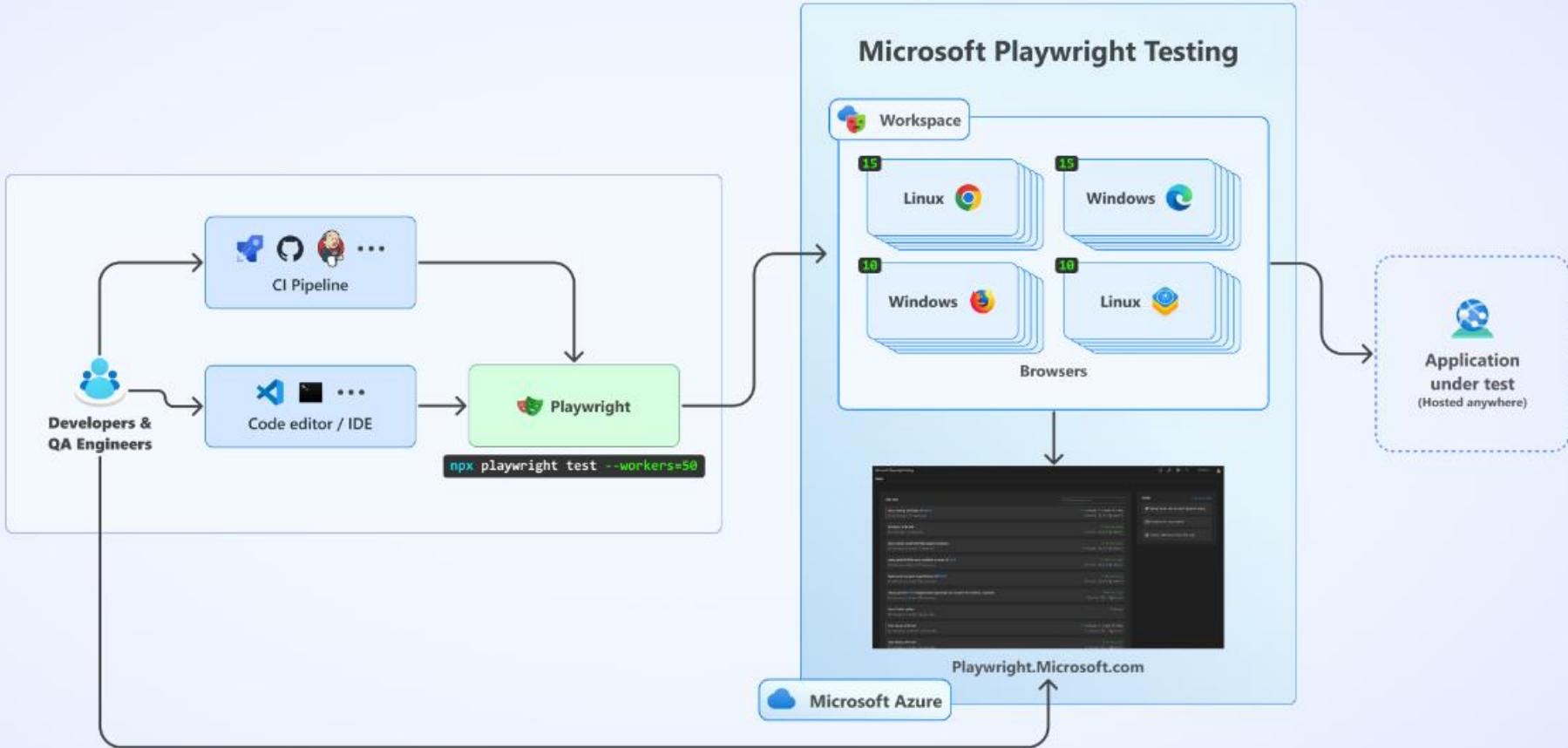


# Microsoft Playwright Testing

Scalable end-to-end modern web app testing service

- Run Playwright tests flexibly in the cloud
- Run Playwright tests with browser and OS combinations
- Cloud enabled to run Playwright tests at scale.
- High parallelization across operating system-browser combinations.
- Get tests done much faster.
- Speed up delivery of features without sacrificing quality.

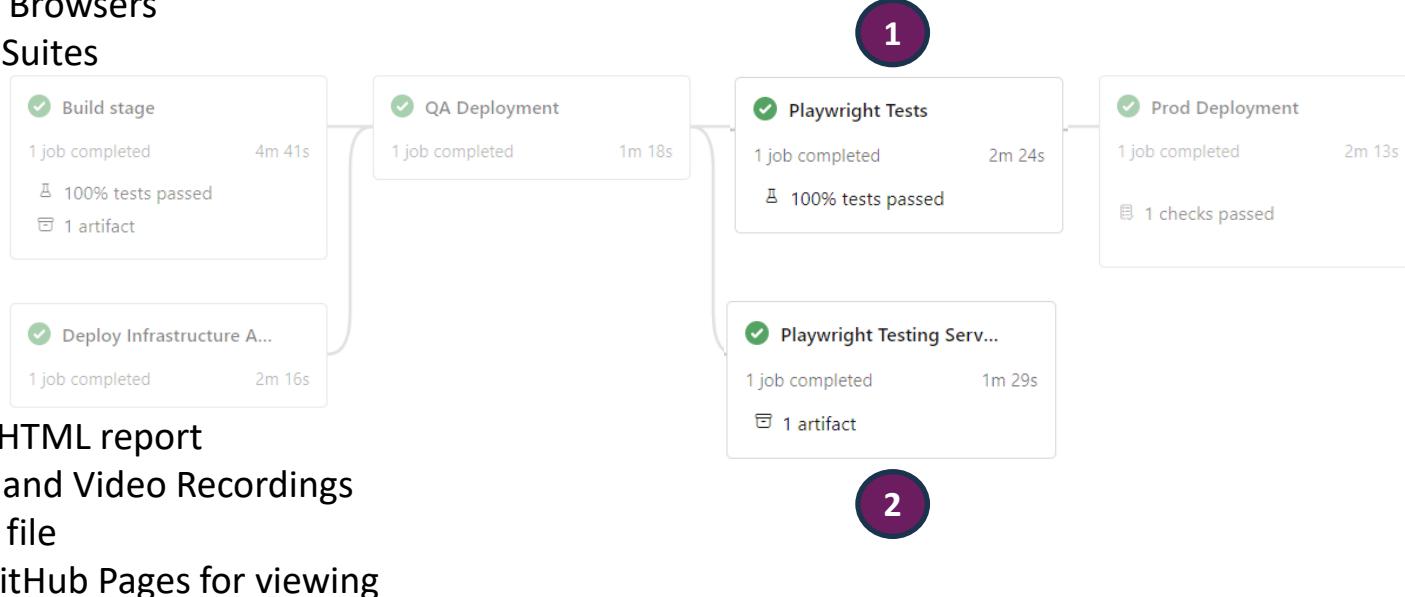




# Continuous Testing

Tests run on each push and pull request into the main/master branch

- Installs all dependencies
- Installs Playwright and Browsers
- Parallelization for Test Suites
- Run all the tests



# Demo

- ✓ Build and Deploy Pipeline
- ✓ Pipeline Results
- ✓ Testing as Scale – Playwright Testing Service
- ✓ HTML Report on GitHub Pages for viewing



# Pipeline Results

- Per build
1. Passed!
  2. Failed!
  3. Percentage!

Summary   Tests   Environments   Advanced Security Alerts   Scans   Snyk Report   Associated pipelines   Code Coverage   Mend   WhiteSource Bolt Build Report

Summary

5 Run(s) Completed ( 4 Passed, 1 Failed ) [4 unique failing tests in the last 14 days](#)

**121** Total tests +121



120 Passed  
1 Failed  
0 Others

**99.17%** Pass percentage ↑ 99.1%



1 Failed tests (+1)  
1 New  
0 Existing

**3m 51s** Run duration ⓘ ↑ +3m 51s

**0** Tests not reported

Bug ▾ Link

Test run ⓘ

Filter by test or run name

Test	Duration	Failing since	Failing build
Playwright Tests from Azure Deploy Pipeline (47/47)	0:02:59.317		
Header > [11190] Shop by Brand but Flaky	0:02:42.897		
Header > [11179] Shop by Type but Flaky	0:01:23.136		
Header > [11441] Login - Admin but flaky <b>New</b>	0:01:20.357	Monday	Current build
Header > [11084] Login - General	0:00:16.354		

# Azure AD Authentication

- Store the credentials of test accounts using environment variables.

```
test(`example test`, async ({ page }) => {
  // ...
  await page.getLabel('User Name').fill(process.env.USERNAME);
  await page.getLabel('Password').fill(process.env.PASSWORD);
});
```

- Local Development? Use .env files and add to .gitignore.

```
# .env file
STAGING=0
USERNAME=me
PASSWORD=secret
```

- Set the values of these variables to use your CI system's secret management or Azure Key Vault.

```
testE2E:
  name: Run Playwright E2E tests
  runs-on: ubuntu-latest
  container: mcr.microsoft.com/playwright:v1.27.0-jammy
  env:
    USERNAME: ${{secrets.USERNAME}}
    PASSWORD: ${{secrets.PASSWORD}}
```

Note: MFA cannot be fully automated and requires manual intervention.

# Tool Comparison

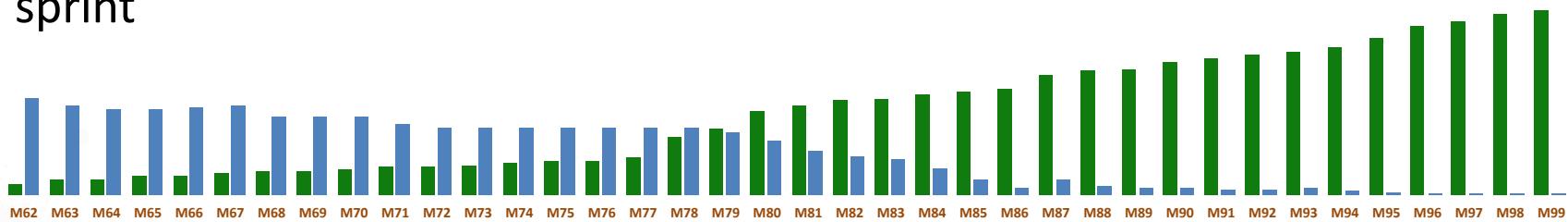
Playwright is not the only browser automation tool out there

Selenium	Cypress	Playwright
Popular among testers	Popular among developers	So hot right now
C#, Java, JavaScript, Ruby, Python	JavaScript only	C#, Java, JavaScript, Python
All major full browsers	Chrome, Edge, Firefox, Electron	Chromium+, Firefox, WebKit
WebDriver protocol	In-browser JavaScript	Debug protocols
Just a tool – requires a framework	Full, modern framework	Full, modern framework
Can be slow/flaky if waiting is poor	Visual execution but can be slow	Typically the fastest execution
Open source, standards, & gov	Open source from Cypress	Open source from Microsoft

# Microsoft Azure DevOps team

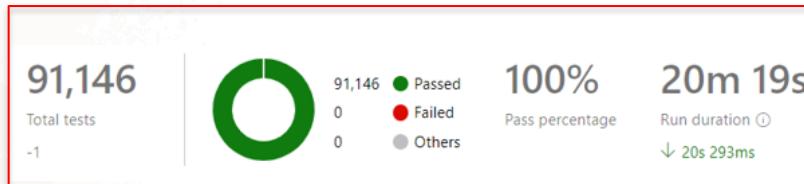
Balancing UI vs Unit Testing in your strategy to maximizes test coverage and effectiveness.

Long running UI functional tests at end of sprint



Shifted to unit tests from automated UI functional tests

ADO PRs require 100% pass rate before merging



ADO Unit Tests in build pipeline for past 14 days



# Playwright CLI

**WANT TO LEARN ABOUT THE PLAYWRIGHT CLI?**

The “-help” option on a command will give you usage documentation

**SCREENSHOTS TAKEN FOR PLAYWRIGHT v1.7! — try them out!**

**OPEN PAGE WITH DESIRED CONTEXT** (e.g., emulated mobile context, with desired state!)

**npx playwright codegen --help**

open page and generate code for user actions

**Options:**

- c output-dir (optional) (string)
- t target-clang (optional) (string)
- d channel (optional) (string)
- r device-width (optional) (number)
- l ignore-https-errors (optional) (boolean)
- lang (optional) (string)
- s save-storage (optional) (boolean)
- tmaximize (optional) (boolean)
- user-agent (optional) (string)
- w user-width (optional) (number)
- z user-height (optional) (number)
- h --help

saves the generated script to a file  
Language to generate, one of `javascript`, `ts`, `python`, `python-native`, `cssless` (default: `text`)  
Chromium distribution channel, “`stable`”, “`beta`”, “`canary`”, “`headless`”, etc  
Desired device width, for example “`1024x768`”  
Ignore https errors, for example “`--ignore-https-errors`”  
Specify language / locale, for example “`--lang de`”  
Save storage state from the file, previously saved with `--storage`  
Specify port number, for example “`--port 3128`” or “`sockets://:rayport:8000`”  
Save contact storage state at the end, for later use with `--load-storage`  
Time to maximize, for example “`--tmaximize 1000`”  
Time to come to a stable state, for example “`--tmaximize 10000`”  
Specify user agent string, for example “`--user-agent “Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4369.90 Safari/537.36”`”  
Specify user width in pixels, for example “`1024, 720`”  
Display help for command

**npx playwright codegen --help**

1. **EFFORTLESS TEST SCRIPT FILE**  
2. **SAVE SCRIPT TO FILE**  
3. **SET TARGET LANGUAGE**  
4. **CONFIGURE CONTEXT**

**npx playwright install [options] [browser...]**

ensure browsers are installed  
Options:  
-m (optional) (string)  
-l (optional) (string)  
-r (optional) (string)  
-c (optional) (string)  
-s (optional) (string)  
-n (optional) (string)  
-e (optional) (string)  
-k (optional) (string)  
-f (optional) (string)  
-t (optional) (string)  
-u (optional) (string)  
-o (optional) (string)  
-d (optional) (string)  
-b (optional) (string)  
-p (optional) (string)  
-g (optional) (string)  
-j (optional) (string)  
-a (optional) (string)  
-w (optional) (string)  
-z (optional) (string)  
-h --help

Install system dependencies for browser  
Install custom browser  
Install custom browser, supports chromium, chrome, chrome-dev, nodejs, nodejs-beta, nodejs-edge, Firefox, webkit.

**npx playwright install --help**

**INSTALL BROWSERS (+SPECIFIC CHANNEL LIST) ON DEMAND**

**LEARN MORE ABOUT PLAYWRIGHT COMMAND LINE TOOLS & OPTIONS** [playwright.dev/docs/cli](#)

**USEFUL FOR TESTING AGAINST CUTTING EDGE BROWSER RELEASERS!**

**npx playwright show-trace --help**

Show trace viewer

Options:  
-b, --browser browserType browser to use, one of cr, chromium, ff, firefox, wk, webkit (default: “chromium”)  
--help display help for command

Examples:  
\$ show-trace https://example.com/trace.zip

**npx playwright open --help**

Open page in browser specified via `--url`, browser to use, one of cr, chromium, ff, firefox, wk, webkit (default: “chromium”)  
Desired device width, for example “`1024x768`”  
Desired device height, for example “`720`”  
Desired screen resolution, for example “`1024x768`”  
Desired color scheme, for example “`light`” or “`dark`”  
Desired geolocation coordinates, for example “`37.417722,-122.476911`”  
Desired orientation, for example “`portrait`” or “`landscape`”  
Desired window size, for example “`1024, 720`”  
Desired window position, for example “`37.417722,-122.476911`”  
Specify proxy server, for example “`http://proxypool:12345`” or “`sockets://:rayport:8000`”  
Specify proxy port, for example “`--proxy-port 12345`”  
Specify user agent string, for example “`--user-agent “Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4369.90 Safari/537.36”`”  
Specify user width in pixels, for example “`1024, 720`”  
Display help for command

**npx playwright open --help**

1. **START HERE**

2. **US PLAYWRIGHT**  
3. **GENERATE TEST REPORTS** (FOR YOUR INTERACTIONS!)  
4. **INSTALL BROWSERS AND DEPENDENCIES**  
5. **GENERATE SCREENSHOTS OF PAGE**

**npx playwright --help**

output the version number  
display help for command

**npx playwright --help**

1. **LAUNCH HTML REPORTS**  
2. **LAUNCH TEST WORKER**  
3. **LAUNCH TRACE VIEWER**  
4. **GENERATE PDFS OF PAGE**

**npx playwright test --help**

Run tests with playwright. Available in `playwright/test` package.  
Run tests with `test` command. See `playwright/test` package.  
Display help for command

**npx playwright test --help**

1. **LAUNCH TEST WORKER**  
2. **LAUNCH TRACE VIEWER**  
3. **LAUNCH HTML REPORTS**  
4. **LAUNCH TEST WORKER**  
5. **LAUNCH TRACE VIEWER**  
6. **LAUNCH HTML REPORTS**  
7. **LAUNCH PARALLELIZE**  
8. **SHARING PARALLELISM**

**npx playwright show-report --help**

1. **LAUNCH TEST WORKER**  
2. **LAUNCH TRACE VIEWER**  
3. **LAUNCH HTML REPORTS**  
4. **LAUNCH TEST WORKER**  
5. **LAUNCH TRACE VIEWER**  
6. **LAUNCH HTML REPORTS**  
7. **LAUNCH PARALLELIZE**  
8. **SHARING PARALLELISM**

**npx playwright pdf --help**

Save page as pdf

Options:  
-w wait-for selector (optional) (string)  
-r wait-for selector (optional) (string)  
-b --browser browserType browser to use, one of cr, chromium, ff, firefox, wk, webkit (default: “chromium”)  
--color-scheme colorScheme desired color scheme, “light” or “dark”  
--device-width deviceWidth desired device width, for example “`1024, 720`”  
--geolocation-coordinates geolocationCoordinates desired geolocation coordinates, for example “`37.417722,-122.476911`”  
--headless headless whether to run in headless mode, “true” or “false”  
--load-storage load-storage contact storage state from the file, previously saved with `--storage`  
--page-size page-size desired page size, for example “`1024x768`” or “`screensize`”  
--proxy-server proxy-server specify proxy server, for example “`http://proxypool:12345`” or “`sockets://:rayport:8000`”  
--proxy-port proxy-port specify proxy port, for example “`--proxy-port 12345`”  
--user-agent user-agent specify user agent string, for example “`--user-agent “Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4369.90 Safari/537.36”`”  
--width width desired width in pixels, for example “`1024, 720`”  
Display help for command

**npx playwright pdf --help**

**BUILT IN ABILITY TO WAIT FOR SELECTION (RELIABILITY) OR TIMEOUT (STABILITY) TO SAVE!**

**DID YOU KNOW “GREP” HELPS!**  
ONLY RUN TESTS IN SCRIPT FILE IF THEY MATCH THE CLI EXPRESSION!

**THE PLAYWRIGHT TEST RUNNER WORKS WITH A STATIC TEST CONFIGURATION FILE THAT PROVIDES TEST RUN CONTEXT**

**THE PLAYWRIGHT TEST RUNNER WORKS WITH A STATIC TEST CONFIGURATION FILE THAT PROVIDES TEST RUN CONTEXT**

**PROJECTS**  
**FIXTURES (use)**  
**TIMEOUT**  
**DIRECTORIES (runners)**  
**METADATA**

**ON YOU CAN CALL IT COMING UP LATER**

[HTTPS://BIT.LY/LEARN-PLAYWRIGHT](https://bit.ly/learn-playwright)

<https://www.azurestaticwebapps.dev/blog/devtools-playwright>

# Playwright Resources

Documentation: <https://playwright.dev>

Source / Issues: <https://github.com/microsoft/playwright>

## Social Media

- <https://aka.ms/playwright/discord>
- <https://aka.ms/playwright/youtube>
- <https://aka.ms/playwright/x>

## Microsoft Playwright Testing

<https://aka.ms/mpt/about>

<https://playwright.microsoft.com/workspaces>

## Test Automation with Playwright with Visual Guide Cheatsheet

<https://www.azurestaticwebapps.dev/blog/devtools-playwright>

eShopOnWeb: <https://github.com/dotnet-architecture/eShopOnWeb>

Slides: <https://github.com/PagelsR/Talks> (Mastering Application Testing with Playwright-v2.pdf)

# Playwright Customer References

- **Mozilla:** Testing many projects, such as [Glean dictionary](#), [Firefox monitor](#), and [Firefox accounts](#).
- **Tesla:** starting using Playwright for e2e testing [September 2021](#).
- **WordPress:** migrated e2e and performance tests to Playwright March 2022. Blog post with details [here](#).
- **Adobe:** E2E for visual regression testing, and accessibility testing for [spectrum web components](#).
- **ING Bank:** uses Playwright to test their [web components](#), since [July 2020](#).
- **Facebook:** uses Playwright in their [React JS library](#).
- **NASA:** uses Playwright for end-to-end testing in [Open MCT \(Open Mission Control Technologies\)](#)
- **Johnson and Johnson:** [migrated to Playwright May 2022](#) to test their [Bodiless-JS Framework](#) for building editable websites on the JAMStack.
- **VMWare:** uses Playwright for integration testing [Tanzau Kubeapps](#).
- **Elastic:** [Test across their SaaS offerings](#) for search, logging, security, observability, and analytics.
- **Target:** uses Playwright to test [GoAlert](#) for on-call scheduling, automated escalations, and notifications.

# Thank you!

