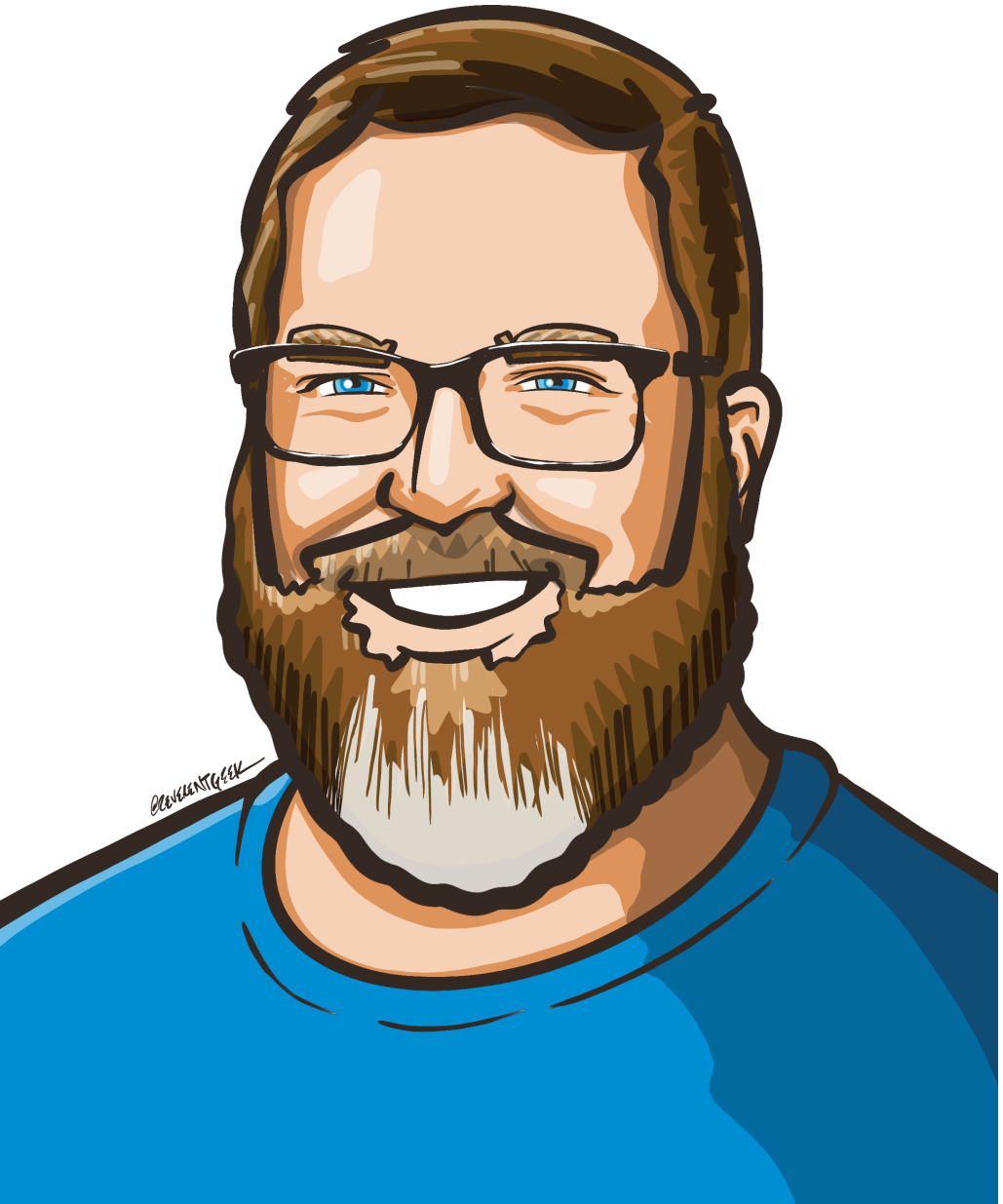
Testing with Playwright

Chris Ayers

https://chris-ayers.com



Chris Ayers

Principal Software Engineer
Azure EngOps AzRel
Microsoft

BlueSky: @chris-ayers.com

LinkedIn: - chris-l-ayers

Blog: https://chris-ayers.com/

GitHub: Codebytes

Mastodon: @Chrisayers@hachyderm.io

~~Twitter: @Chris_L_Ayers~~

Agenda

- What is Testing?
- What is Playwright?
- Benefits of Using Playwright
- Creating End-to-End Tests with Playwright
- Integrating with Continuous Integration

What is Testing?

Testing is the ***systematic process*** of **verifying** and **validating** that a software application or system meets specified requirements and functions correctly.

Types of Testing

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing
- End-to-End (E2E) Testing
- Accessibility Testing
- Visual Testing
- Component Testing

UI Testing Best Practices

- Test user-visible behavior
- Make tests as **isolated** as possible
- Avoid testing third-party dependencies
- If you test with a database, **control** the data

Testing Challenges

- Testing is **hard**
- Testing takes **time to learn**
- Testing **time to build**
- **Testing culture**

- Tests are **slow**
- Tests are **brittle**
- Tests are **flaky**

Lets try to make testing easier... with

Playwright

What is Playwright?

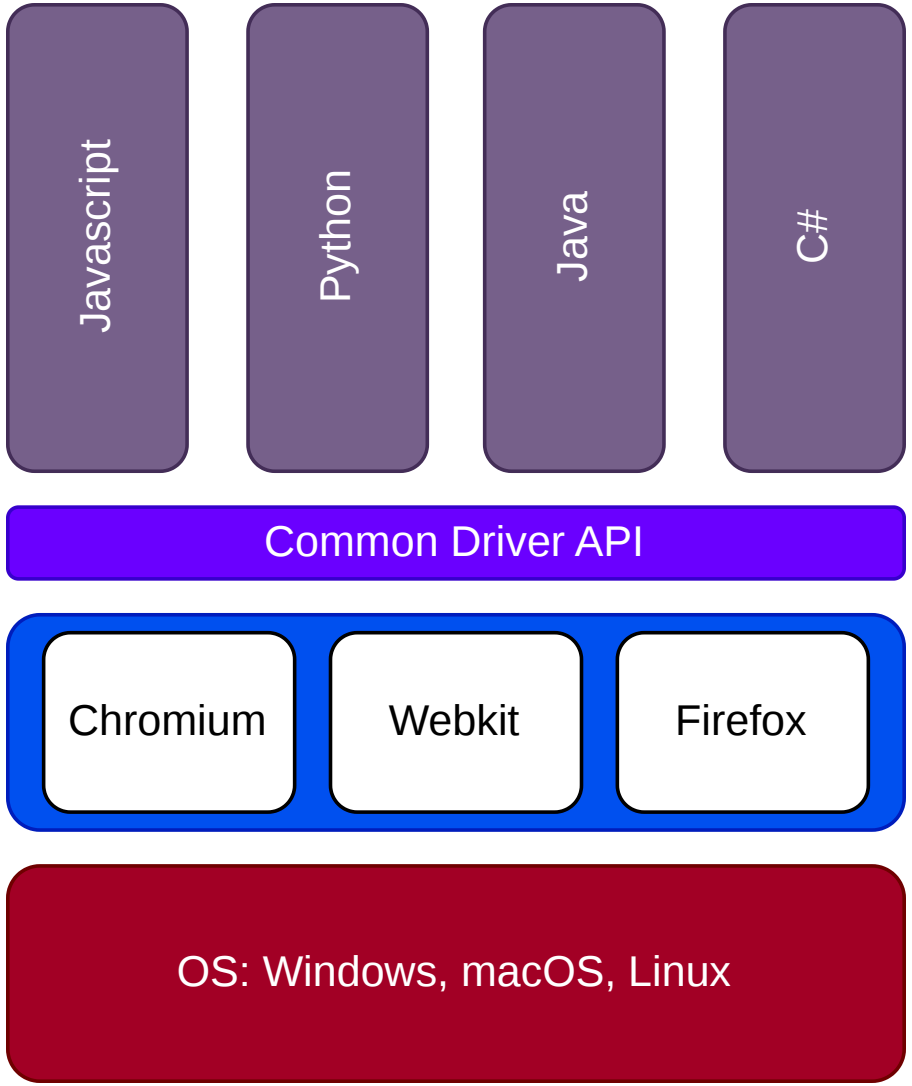
- **Open Source released by Microsoft in 2020**
- A Modern web test framework
- Works with Headless or Headed Browsers
  - Chromium - Chrome/Edge
  - Firefox
  - WebKit

Language Support

- Bindings for:
  - Python
  - Javascript/Typescript
  - Java
  - .NET

Playwright Architecture

- Language Bindings
- Single Automation Protocol
- Abstracts debugging protocols



Benefits of Using Playwright

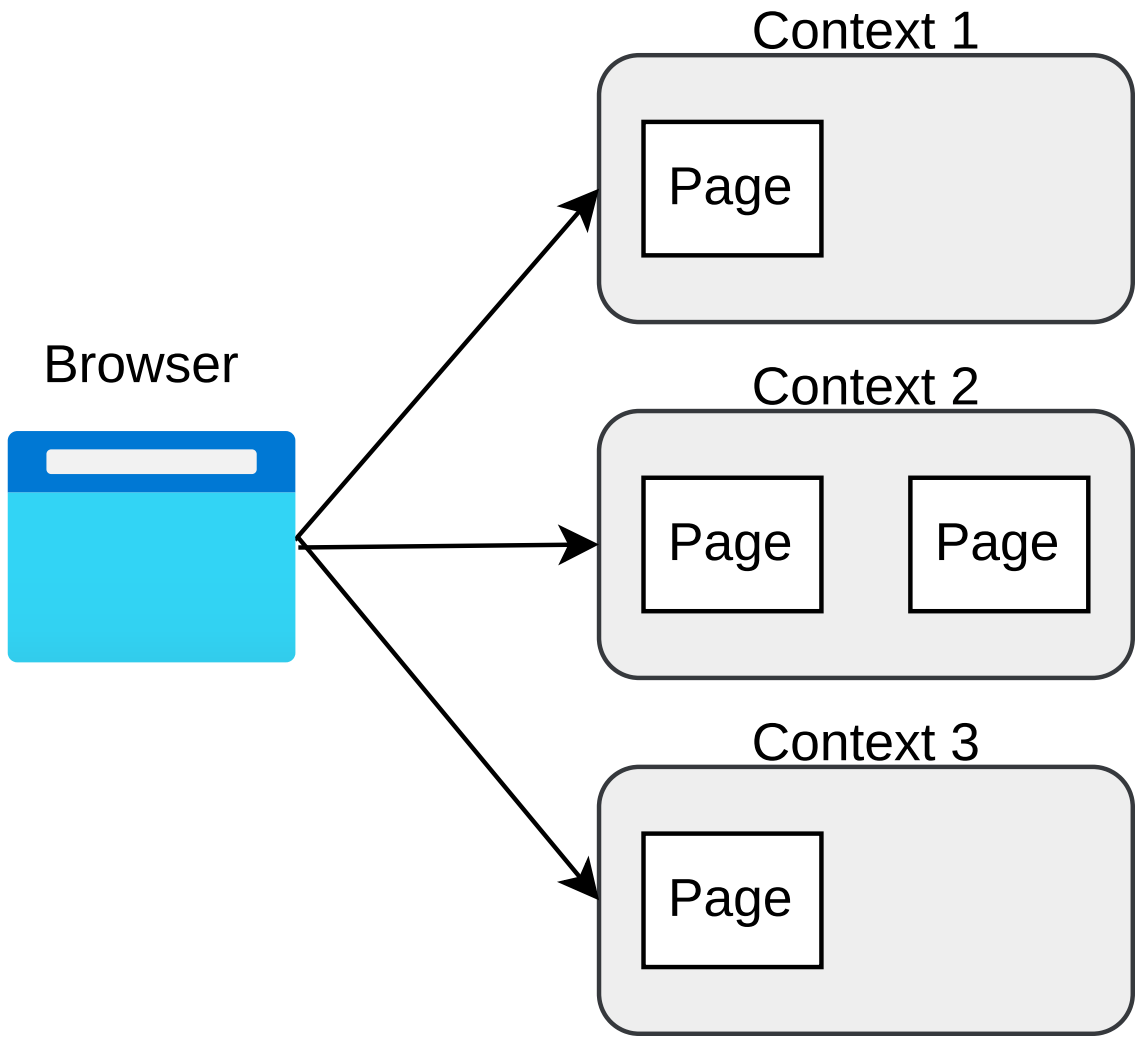
Reliable & Stable

- Automatically waits for UI to be ready.
- Handles dynamic content, animations, and AJAX requests gracefully.
- Minimizes flaky tests.
- Isolation with Browser Contexts

Browser Contexts

Browser Contexts provide a way to operate multiple independent browser sessions.

If a page opens another page, e.g. with a window.open call, the popup will belong to the parent page's browser context.

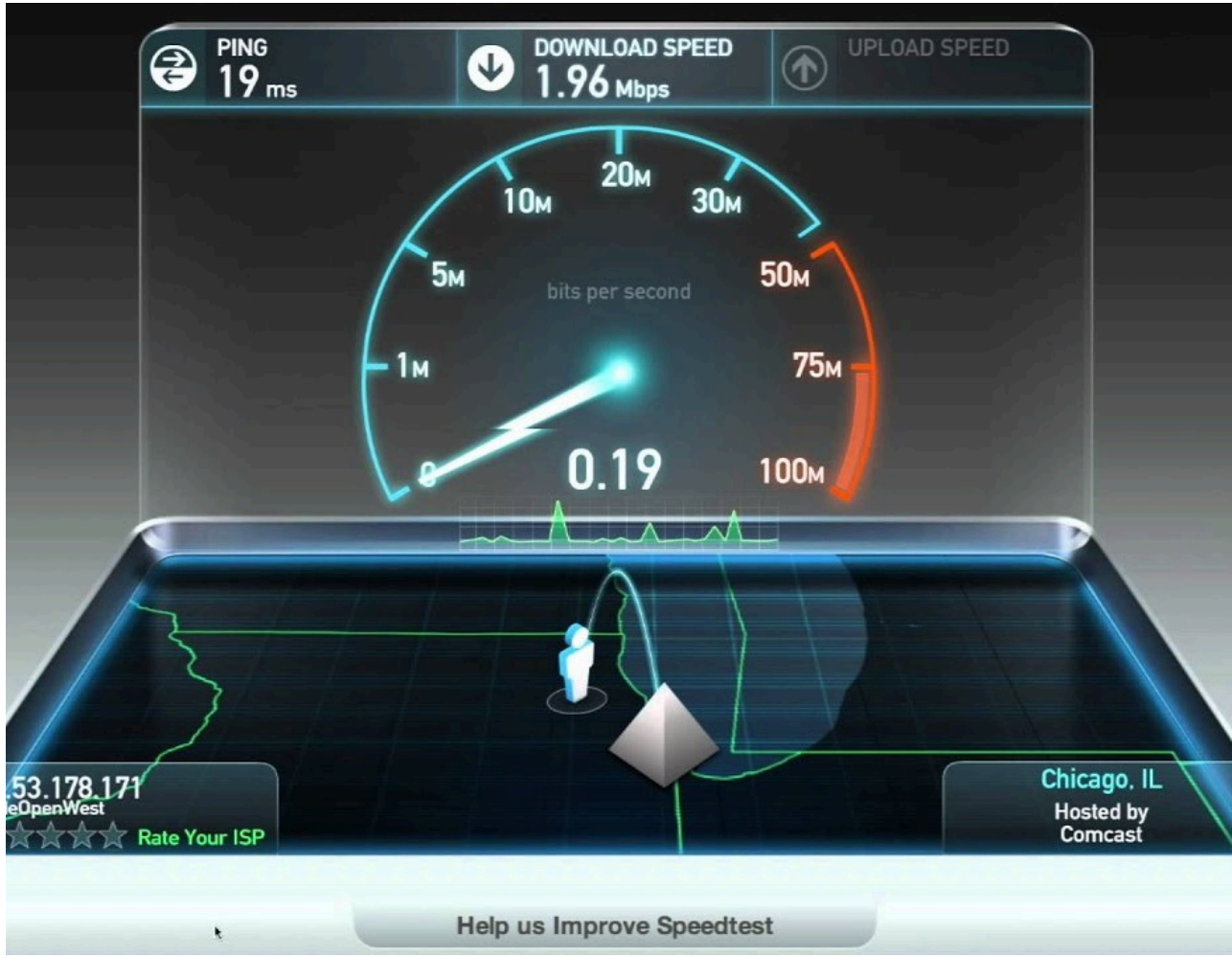


Mobile Emulation

- Easily emulate various mobile devices.
  - Mobile Chrome
  - Mobile Safari
  - Emulate Viewports and Scale
- Test responsiveness and mobile-specific features.
- Ability to emulate location via geolocation

Advanced Interactions

- Network request interception.
- Create custom scenarios (e.g., offline mode, slow network).
- API Mocking
  - HAR file support



Native Context Automation

- Automate beyond the browser:
  - Upload & download files
  - Work with iframes and shadow DOM

```
// Get frame using the frame's name attribute
const frame = page.frame('frame-login');

// Get frame using frame's URL
const frame = page.frame({ url: /.*domain.*/ });

// Interact with the frame
await frame.fill('#username-input', 'John');
```

Fast Execution

- Tests are executed swiftly, reducing waiting time.
- Parallel test execution.
- Optimal performance due to its architecture.
- Isolation with Browser Contexts

Auto-Waiting

For example, for *page.click()*, Playwright will ensure that:

- element is Attached to the DOM
- element is Visible
- element is Stable, as in not animating or completed animation
- element Receives Events, as in not obscured by other elements
- element is Enabled

See a full list of actions at: https://playwright.dev/docs/actionability

Locators

Method	Description
`page.getByRole()`	Locate by explicit and implicit accessibility attributes.
`page.getByText()`	Locate by text content.
`page.getByLabel()`	Locate a form control by associated label's text.
`page.getByPlaceholder()`	Locate an input by placeholder.
`page.getByAltText()`	Locate an element, usually image, by its text alternative.
`page.getByTitle()`	Locate an element by its title attribute.
`page.getByTestId()`	Locate an element based on its data-testid attribute.

Web-First Assertions

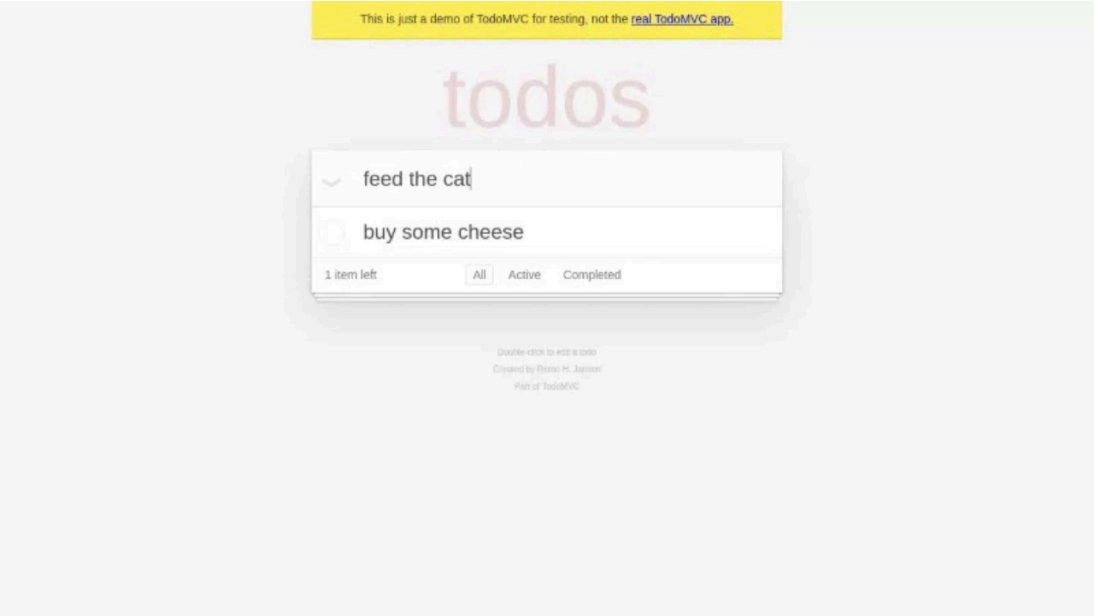
expect(locator)	expect(locator/page/response)
expect(locator).toBeAttached()	expect(locator).toHaveClass()
expect(locator).toBeChecked()	expect(locator).toHaveCount()
expect(locator).toBeDisabled()	expect(locator).toHaveCSS()
expect(locator).toBeEditable()	expect(locator).toHaveId()
expect(locator).toBeEmpty()	expect(locator).toHaveJSProperty()
expect(locator).toBeEnabled()	expect(locator).toHaveScreenshot()

Web-First Assertions (continued)

expect(locator)	expect(locator/page/response)
expect(locator).toBeFocused()	expect(locator).toHaveText()
expect(locator).toBeHidden()	expect(locator).toHaveValue()
expect(locator).toBeInViewport()	expect(locator).toHaveValues()
expect(locator).toBeVisible()	expect(page).toHaveTitle()
expect(locator).toContainText()	expect(page).toHaveURL()
expect(locator).toHaveAttribute()	expect(response).toBeOK()

Visual evidence

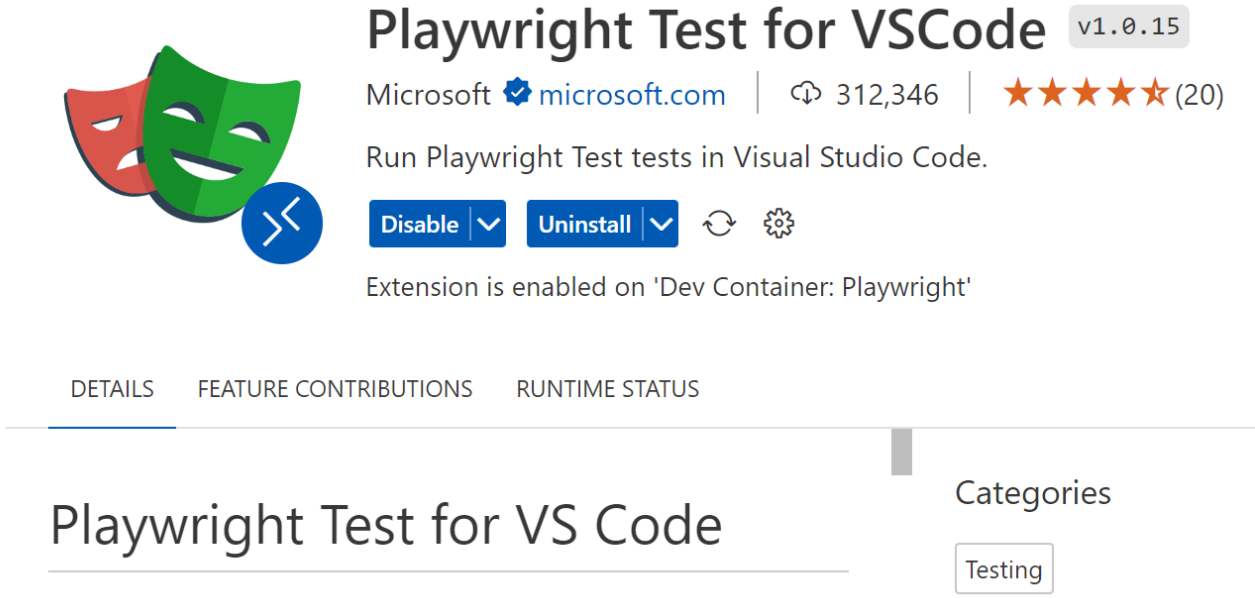
- Screenshot support
- Video Recording



Rich Tooling Ecosystem

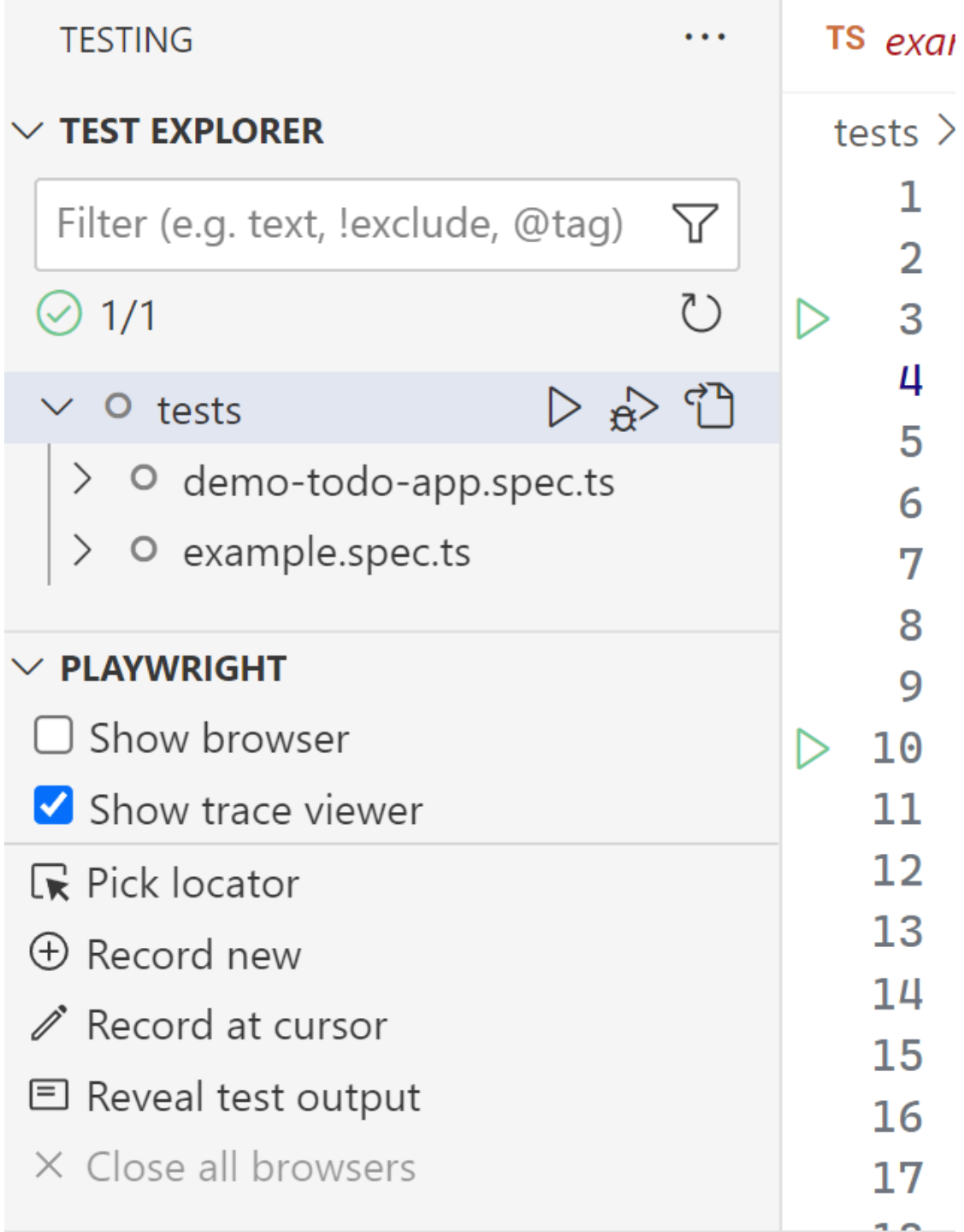
- VS Code Extension
- Integrations with popular CI/CD services.
- Compatible with multiple assertion libraries.

Playwright Extension



https://chris-ayers.com

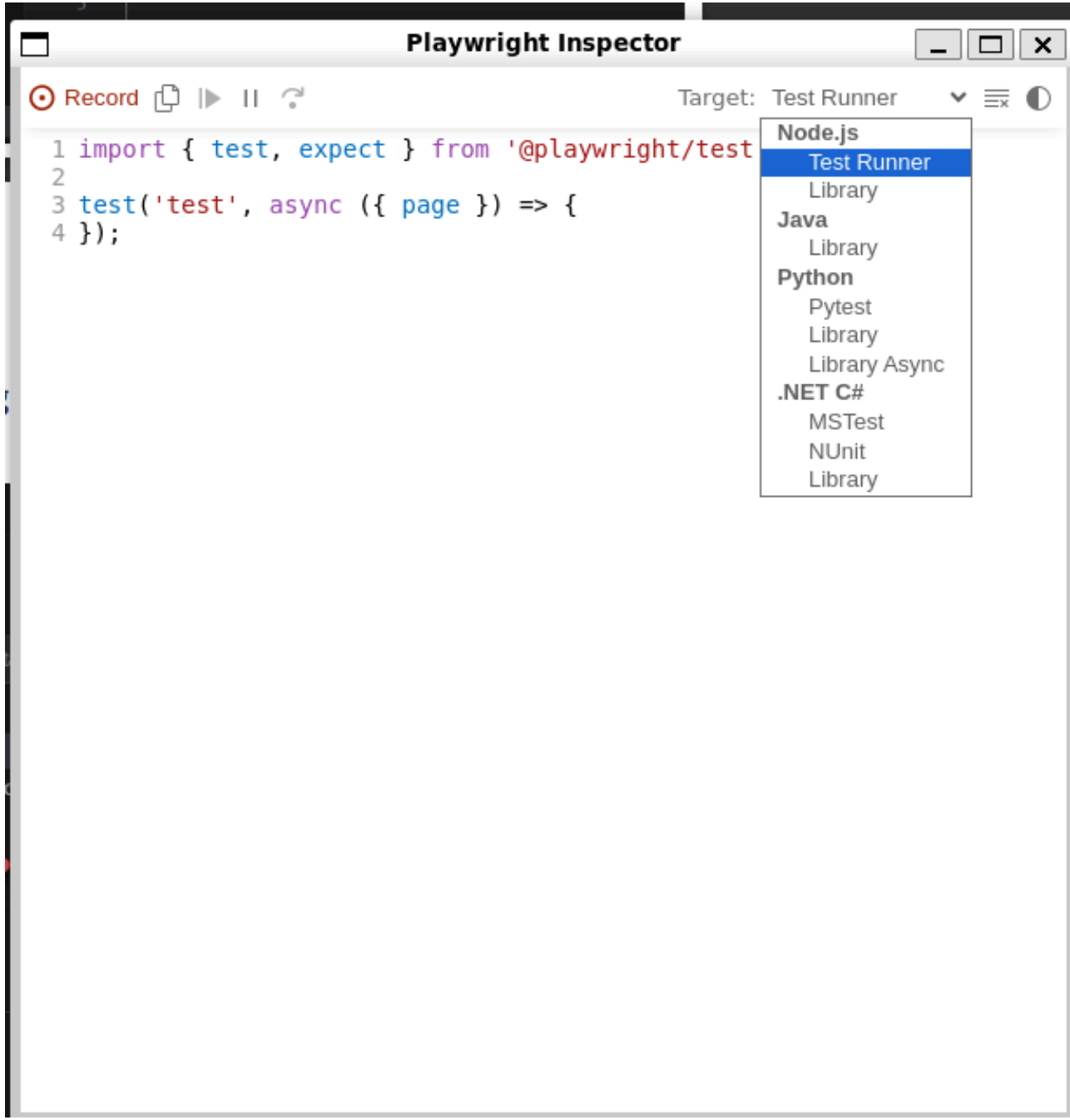
Integration with IDE Testing



https://chris-ayers.com

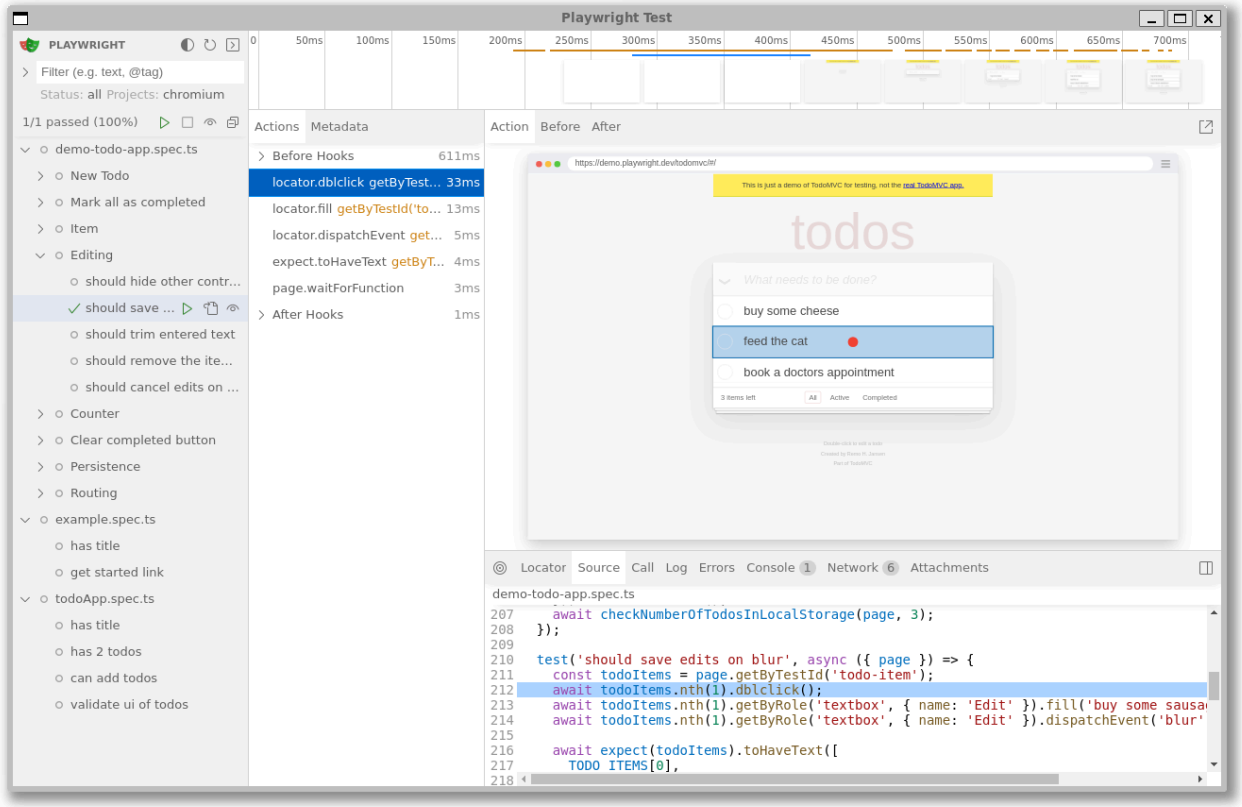
Codegen

Playwright Test Generator is a GUI tool that helps you record Playwright tests



UI Mode

Run tests and visually see how it runs



https://chris-ayers.com

Playwright Best Practices

- Use locators
  - Use chaining and filtering
  - Prefer user-facing attributes to XPath or CSS selectors
- Generate locators
  - Use codegen to generate locators
  - Use the VS Code extension to generate locators
- Use web first assertions
  - Don't use manual assertions

Playwright Best Practices

- Configure debugging
  - Local debugging
  - Debugging on CI
- Use Playwright's Tooling
- Test across all browsers
- Keep your Playwright dependency up to date
- Run tests on CI
- Lint your tests
  - Use parallelism and sharding

DEMOS

Microsoft Playwright Testing

Join the waitlist for Microsoft Playwright Testing private preview

- New Azure service for running Playwright tests.
- Cloud enabled to run Playwright tests at scale.
- High parallelization across operating system-browser combinations.
- Speed up delivery of features without sacrificing quality.

Private Preview

To learn more about Microsoft Playwright Testing, refer to:

https://aka.ms/mpt/private-preview-blog.

Questions?

https://chris-ayers.com

Resources

GitHub Repo

https://github.com/codebytes/testing-with-playwright

Playwright website

https://playwright.dev/

Contact

BlueSky: @chris-ayers.com

LinkedIn: - chris-l-ayers

Blog: https://chris-ayers.com/

GitHub: Codebytes

Mastodon: @Chrisayers@hachyderm.io

~~Twitter: @Chris_L_Ayers~~

AN INTRODUCTION TO PLAYWRIGHT

@playwrightweb
https://playwright.dev

READY TO START YOUR PLAYWRIGHT RUN ?

5 RESOURCES TO HELP YOU JUMPSTART YOUR TEST ADVENTURE

1 READ THE DOCS
- https://aka.ms/playwright
- https://aka.ms/playwright/quickstart
- Install Playwright
- Run your first Test
- Explore Tools & Libraries

2 EXPLORE THE API
https://aka.ms/playwright/api
JS, Java, Python, .NET
- Playwright Test Runner
- Test Reporter
- Testing + Experimental

3 VISIT THE REPO
https://aka.ms/playwright/github
- Playwright Framework source
- Examples + Release docs
- Discussions

4 FOLLOW @playwrightweb
- Framework release updates
- Content + event announcements
- Community interactions

5 CHECKOUT DEMO CODE
https://aka.ms/playwright/demo
- Test script examples
- Github Actions integration
- Test Reporting outputs

WHAT IS PLAYWRIGHT?

0 AN OPEN SOURCE TESTING FRAMEWORK
- ENABLES RELIABLE END-TO-END TESTING AND AUTOMATION
- FOR MODERN WEB APPLICATIONS

https://github.com/microsoft/playwright

- TOOLING
- AUTOMATION
- API

CHECK OUT THE SOURCE

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

LET'S TALK ABOUT EACH OF THESE!

1 ONE API / MULTI EVERYTHING

PLAYWRIGHT – the Grey –

CHROMIUM WEBKIT FIREFOX

ONE API TO RULE THEM

- ANY BROWSER ANY PLATFORM
  Chromium, WebKit, Firefox
  MacOS, Linux, Windows
- CROSS LANGUAGE
  JavaScript, TypeScript, Python, Java, .NET – libraries.
- TEST MOBILE WEB
  Native emulation
  - Chrome on Android
  - Mobile Safari on iOS
  - Same Rendering Engine on Cloud and Desktop.

CROSS BROWSER TESTING

TEST AUTOMATION AT CLOUD SCALE

2 RESILIENT! NO FLAKY TESTS...

TESTING ENTRY

ALMOST READY FOR YOU .. HANG ON .. LET ME CHECK AGAIN ...

ARE YOU EXPECT-ing US?

TEST ACTIONS

REDUCE TEST COMPLEXITY
INCREASE READABILITY + RELIABILITY

WEB PAGE

1 AUTO-WAIT
No more artificial timeouts to manage

- waits for elements to be ACTIONABLE before execution of test steps
- + RICH introspection events available!

2 WEB-FIRST ASSERTIONS
tailored for the dynamic web!

ASYNC AWAIT EXPECT

- convenience async matchers for assertions
- separate timeout (from test) for web first assert with default = 5 secs
- checks conditions till met

3 TRACING
configurable strategy

retry? record traces? capture screenshot?

3 NO TRADE-OFFS / LIMITS

TESTING API
TEST RUNNER API
TEST CONFIG OPTIONS

1 ALIGNED WITH MODERN WEB ARCHITECTURES
- RUN TESTS OUT OF PROCESS .. FREED FROM IN-PROCESS TEST RUNNER CONSTRAINT

2 MULTIPLE EVERYTHING
MULTIPLE ORIGINS
MULTIPLE TABS
MULTIPLE USERS
ONE TEST MANY CONTEXTS

TEST ALL THE THINGS

WRITE MORE COMPREHENSIVE TEST ACTIONS & SCRIPTS!
MORE REALISTIC USER ACTION FLOWS

3 TRUSTED EVENTS
I'M IN YOUR BROWSER – ACTING LIKE ANY USER!

INDISTINGUISHABLE FROM REAL USERS!

- TEST HOVER ELEMENTS
- INTERACT WITH DYNAMIC CONTROLS
- USE REAL BROWSER INPUT PIPELINES

4 TEST FRAMES
PIERCE SHADOW DOM WITH PLAYWRIGHT SELECTORS!

4 FULL ISOLATION / FAST EXECUTION

1 BROWSER CONTEXT
TEST SCRIPT
LIKE HAVING A BRAND NEW BROWSER PROFILE EACH TIME

- CREATES A BROWSER CONTEXT PER TEST IN JUST ms.
  - FULL TEST ISOLATION
  - ZERO OVERHEADS
  - LESS COMPLEXITY FOR TESTER

2 LOG IN ONCE!
TEST SCRIPT
STORE AUTH STATE
PATH TO FILE
LOAD AUTH STATE (from stored location)
TEST NEW TIMING

- SAVE AUTHENTICATION STATE (tokens, cookies) FROM FIRST RUN – & REUSE IN ALL SUBSEQUENT TESTS
- BYPASS REPETITIVE LOGIN OPERATIONS (many third party sites don't want automated logins)
- STILL FULL ISOLATION FOR INDEPENDENT TESTS

5 POWERFUL TOOLING!

1 CODEGEN
INTERACTIVE SESSION
AUTO CAPTURED

- GENERATE TESTS BY RECORDING USER ACTION (FLOW)
- SAVE IN ANY SUPPORTED LANG. (Java, JS/TS, Python etc.)

2 INSPECTOR
TEST – DEBUG
INSPECTOR

- INSPECT PAGE, LOCS
- GENERATE SELECTOR
- STEP THROUGH EXEC
- SEE CLICK POINTS

3 TRACE VIEWER
- CAPTURE TEST PROGRESS (STATE, FAILURES, TIMING)
- SCREENCASTS, SNAPSHOTS, ACTION EXPLORER, TEST SOURCE (POST-MORTEM ANALYSIS)

ILLUSTRATED BY @SKETCHTHEDOCS

WANT TO LEARN ABOUT THE PLAYWRIGHT CLI?

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

```
(base) nitya@nityas-mbp ~ % npx playwright show-trace --help
Usage: npx playwright show-trace [options] [trace]

Show trace viewer

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

Examples:

  $ show-trace https://example.com/trace.zip
```

npx playwright show-trace --help

THE "playwright test" command will be the one you use most often!

playwright test --debug will launch the INSPECTOR tool!

SCREENSHOTS TAKEN FOR PLAYWRIGHT v1.17.1 — try them out!

OPEN PAGE WITH DESIRED BROWSER OR EMULATED DEVICE WITH DESIRED CONTEXT LOAD/SAVE STATE!

1

```
(base) nitya@nityas-mbp ~ % npx playwright open --help
Usage: npx playwright open [options] [url]

open page in browser specified via -b, --browser

Options:
  -b, --browser <browserType>  browser to use, one of cr, chromium, ff, firefox, wk, webkit (default: "chromium")
  --channel <channel>          Chromium distribution channel, "chrome", "chrome-beta", "msedge-dev", etc
  --color-scheme <scheme>      emulate preferred color scheme, "light" or "dark"
  --device <deviceName>        emulate device, for example  "iPhone 11"
  --geolocation <coordinates>  specify geolocation coordinates, for example "37.819722,-122.478611"
  --ignore-https-errors        ignore https errors
  --load-storage <filename>    load context storage state from the file, previously saved with --save-storage
  --lang <language>            specify language / locale, for example "en-GB"
  --proxy-server <proxy>       specify proxy server, for example "http://myproxy:3128" or "socks5://myproxy:8080"
  --save-storage <filename>    save context storage state at the end, for later use with --load-storage
  --save-trace <filename>      record a trace for the session and save it to a file
  --timezone <time zone>       time zone to emulate, for example "Europe/Rome"
  --timeout <timeout>          timeout for Playwright actions in milliseconds (default: "10000")
  --user-agent <ua string>     specify user agent string
  --viewport-size <size>       specify browser viewport size in pixels, for example "1280, 720"
  -h, --help                   display help for command
```

npx playwright open --help

6

8 launch HTML reporter
7 launch Test runner
6 launch trace viewer
5 generate PDFs of page
1 open pages and emulate mobile contexts
2 generate test scripts from your interactions!
3 install browsers and dependencies
4 generate screenshots of page

USE PLAYWRIGHT CLI TO..

START HERE

THE PLAYWRIGHT CLI

```
(base) nitya@nityas-mbp ~ % npx playwright --help
Usage: npx playwright [options] [command]

Options:
  -V, --version                       output the version number
  -h, --help                          display help for command

Commands:
  open [options] [url]                open page in browser specified via -b, --browser
  codegen [options] [url]             open page and generate code for user actions
  install [options] [browser...]      ensure browsers necessary for this version of Playwright are installed
  install-deps [browser...]           install dependencies necessary to run browsers (will ask for sudo permissions)
  cr [options] [url]                  open page in Chromium  ] same as open -b {cr|ff|wk}
  ff [options] [url]                  open page in Firefox   ]
  wk [options] [url]                  open page in WebKit    ]
  screenshot [options] <url> <filename>  capture a page screenshot
  pdf [options] <url> <filename>      save page as pdf
  show-trace [options] [trace]        Show trace viewer
  test                                Run tests with Playwright Test. Available in @playwright/test package.
  show-report                         Show Playwright Test HTML report. Available in @playwright/test package.
  help [command]                      display help for command
```

npx playwright --help

READ THE DOCS!

playwright dev/docs/test-cli

npx playwright test

Complete set of Playwright Test options is available in the configuration file. Following options can be passed to a command line and take a priority over the configuration file:

- `--headed`: Run tests in headed browsers. Useful for debugging.
- `--browser`: Run test in a specific browser. Available options are `"chromium"`, `"firefox"`, `"webkit"` or `"all"` to run tests in all three browsers at the same time.
- `--debug`: Run tests with Playwright Inspector. Shortcut for `PWDEBUG=1` environment variable and `--timeout=0 --maxFailures=1 --headed --workers=1` options
- `-c <file>` or `--config <file>`: Configuration file. If not passed, defaults to `playwright.config.ts` or `playwright.config.js` in the current directory.
- `-C <dir>` or `--config <dir>`: Directory with the tests to run without configuration file.
- `--forbid-only`: Whether to disallow `test.only`. Useful on CI.
- `-g <grep>` or `--grep <grep>`: Only run tests matching this regular expression. For example, this will run `'should add to cart'` when passed `-g="add to cart"`.
- `--grep-invert <grep>`: Only run tests **not** matching this regular expression. The opposite of `--grep`.
- `--global-timeout <number>`: Total timeout for the whole test run in milliseconds. By default, there is no global timeout.
- `--list`: List all the tests, but do not run them.
- `--max-failures <N>` or `-x`: Stop after the first N test failures. Passing `-x` stops after the first failure.
- `--output <dir>`: Directory for artifacts produced by tests, defaults to `test-results`.
- `--project <name>`: Only run tests from one of the specified projects. Defaults to running all projects defined in the configuration file.
- `--quiet`: Whether to suppress stdout and stderr from the tests.
- `--repeat-each <N>`: Run each test N times, defaults to one.
- `--reporter <reporter>`: Choose a reporter: minimalist `dot`, concise `line` or detailed `list`. See reporters for more information.
- `--retries <number>`: The maximum number of retries for flaky tests, defaults to zero (no retries).
- `--shard <shard>`: Shard tests and execute only selected shard, specified in the form `current/all`, 1-based, for example `3/5`.
- `--timeout <number>`: Maximum timeout in milliseconds for each test, defaults to 30 seconds.
- `--update-snapshots` or `-u`: Whether to update snapshots with actual results instead of comparing them. Use this when snapshot expectations have changed.
- `--workers <number>` or `-j <number>`: The maximum number of concurrent worker processes that run in parallel.

DEBUG

REPORTS

7

SHARDS!!

2

```
(base) nitya@nityas-mbp ~ % npx playwright codegen --help
Usage: npx playwright codegen [options] [url]

open page and generate code for user actions

Options:
  -o, --output <file name>         saves the generated script to a file
  --target <language>              language to generate, one of javascript, test, python, python-async, csharp (default: "test")
  -b, --browser <browserType>      browser to use, one of cr, chromium, ff, firefox, wk, webkit (default: "chromium")
  --channel <channel>              Chromium distribution channel, "chrome", "chrome-beta", "msedge-dev", etc
  --color-scheme <scheme>          emulate preferred color scheme, "light" or "dark"
  --device <deviceName>            emulate device, for example  "iPhone 11"
  --geolocation <coordinates>      specify geolocation coordinates, for example "37.819722,-122.478611"
  --ignore-https-errors            ignore https errors
  --load-storage <filename>        load context storage state from the file, previously saved with --save-storage
  --lang <language>                specify language / locale, for example "en-GB"
  --proxy-server <proxy>           specify proxy server, for example "http://myproxy:3128" or "socks5://myproxy:8080"
  --save-storage <filename>        save context storage state at the end, for later use with --load-storage
  --save-trace <filename>          record a trace for the session and save it to a file
  --timezone <time zone>           time zone to emulate, for example "Europe/Rome"
  --timeout <timeout>              timeout for Playwright actions in milliseconds (default: "10000")
  --user-agent <ua string>         specify user agent string
  --viewport-size <size>           specify browser viewport size in pixels, for example "1280, 720"
  -h, --help                       display help for command
```

npx playwright codegen --help

EFFORTLESS TEST SCRIPT CREATION FTW!

- SAVE SCRIPT TO FILE
- PICK TARGET LANGUAGE
- CONFIGURE CONTEXT

npx playwright show-report

8

THINK PARALLELIZATION

DID YOU KNOW "GREP" HELPS!

ONLY RUN TESTS IN SCRIPT FILE IF THEY MATCH THE CLI-given EXPRESSION!!

3

```
Usage: npx playwright install [options] [browser...]

ensure browsers necessary for this version of Playwright are installed

Options:
  --with-deps  install system dependencies for browsers
  -h, --help   display help for command

Examples:
  - $ install
    Install default browsers.

  - $ install chrome firefox
    Install custom browsers, supports chromium, chrome, chrome-beta, msedge, msedge-beta, msedge-dev, firefox, webkit.
```

npx playwright install --help

INSTALL BROWSERS (+ SPECIFIC CHANNEL DIST.) ON DEMAND

USEFUL FOR TESTING AGAINST CUTTING EDGE BROWSER RELEASES!

LEARN MORE ABOUT PLAYWRIGHT COMMAND LINE TOOLS & OPTIONS
playwright.dev/docs/cli

4

```
(base) nitya@nityas-mbp ~ % npx playwright screenshot --help
Usage: npx playwright screenshot [options] <url> <filename>

capture a page screenshot

Options:
  --wait-for-selector <selector>  wait for selector before taking a screenshot
  --wait-for-timeout <timeout>    wait for timeout in milliseconds before taking a screenshot
  --full-page                     whether to take a full page screenshot (entire scrollable area)
  -b, --browser <browserType>     browser to use, one of cr, chromium, ff, firefox, wk, webkit (default: "chromium")
  --channel <channel>             Chromium distribution channel, "chrome", "chrome-beta", "msedge-dev", etc
  --color-scheme <scheme>         emulate preferred color scheme, "light" or "dark"
  --device <deviceName>           emulate device, for example  "iPhone 11"
  --geolocation <coordinates>     specify geolocation coordinates, for example "37.819722,-122.478611"
  --ignore-https-errors           ignore https errors
  --load-storage <filename>       load context storage state from the file, previously saved with --save-storage
  --lang <language>               specify language / locale, for example "en-GB"
  --proxy-server <proxy>          specify proxy server, for example "http://myproxy:3128" or "socks5://myproxy:8080"
  --save-storage <filename>       save context storage state at the end, for later use with --load-storage
  --save-trace <filename>         record a trace for the session and save it to a file
  --timezone <time zone>          time zone to emulate, for example "Europe/Rome"
  --timeout <timeout>             timeout for Playwright actions in milliseconds (default: "10000")
  --user-agent <ua string>        specify user agent string
  --viewport-size <size>          specify browser viewport size in pixels, for example "1280, 720"
  -h, --help                      display help for command
```

npx playwright screenshot --help

CAPTURE page screenshot with emulated device or specific contexts!!

- DEVICE TYPE
- COLOR SCHEME etc.
- GEO LOCATION

5

```
(base) nitya@nityas-mbp ~ % npx playwright pdf --help
Usage: npx playwright pdf [options] <url> <filename>

save page as pdf

Options:
  --wait-for-selector <selector>  wait for given selector before saving as pdf
  --wait-for-timeout <timeout>    wait for given timeout in milliseconds before saving as pdf
  -b, --browser <browserType>     browser to use, one of cr, chromium, ff, firefox, wk, webkit (default: "chromium")
  --channel <channel>             Chromium distribution channel, "chrome", "chrome-beta", "msedge-dev", etc
  --color-scheme <scheme>         emulate preferred color scheme, "light" or "dark"
  --device <deviceName>           emulate device, for example  "iPhone 11"
  --geolocation <coordinates>     specify geolocation coordinates, for example "37.819722,-122.478611"
  --ignore-https-errors           ignore https errors
  --load-storage <filename>       load context storage state from the file, previously saved with --save-storage
  --lang <language>               specify language / locale, for example "en-GB"
  --proxy-server <proxy>          specify proxy server, for example "http://myproxy:3128" or "socks5://myproxy:8080"
  --save-storage <filename>       save context storage state at the end, for later use with --load-storage
  --save-trace <filename>         record a trace for the session and save it to a file
  --timezone <time zone>          time zone to emulate, for example "Europe/Rome"
  --timeout <timeout>             timeout for Playwright actions in milliseconds (default: "10000")
  --user-agent <ua string>        specify user agent string
  --viewport-size <size>          specify browser viewport size in pixels, for example "1280, 720"
  -h, --help                      display help for command
```

npx playwright pdf --help

SAVE PAGE AS PDF
CUSTOMIZE YOUR BROWSER, CONTEXT!

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

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

- PROJECTS
- RETRIES
- FIXTURES (use)
- TIMEOUT
- DIRECTORIES (various)
- METADATA

OR YOU CAN DYNAMICALLY CONFIG THIS VIA CLI!

SKETCHNOTES BY @NITYA

HTTPS://BIT.LY/LEARN-PLAYWRIGHT