Playwright

# Welcome!

www.playwright.dev

February 7th, 2024

Ghislain Gabriëlse & Jurgen De Reu

# The Menu

- Framework
- Tooling
- Basic Usage
- Advanced Usage
- Bonus

# Framework

- What is Playwright?
- Comparison between Chrome DevTools and WebDriver
- Advantages of using Playwrigh

# What is Playwright?

- Playwright is a powerful web automation tool that supports Chromium, Webkit, and Firefox browsers.
- It offers support for popular programming languages including JavaScript, Typescript, Python, .NET, and Java.
- Playwright allows for emulation of Chrome for Android and Mobile Safari.

# Chrome DevTools vs WebDriver

- Playwright leverages the Chrome DevTools for Chrome/Chromium browser automation.
- For Firefox and Webkit, Playwright provides a similar custom implementation.
- Compared to WebDriver, Chrome DevTools generally offers faster performance.
- Chrome DevTools allows for granular control over the browser.
- One major benefit of Playwright is that you don't need to manage webdrivers!

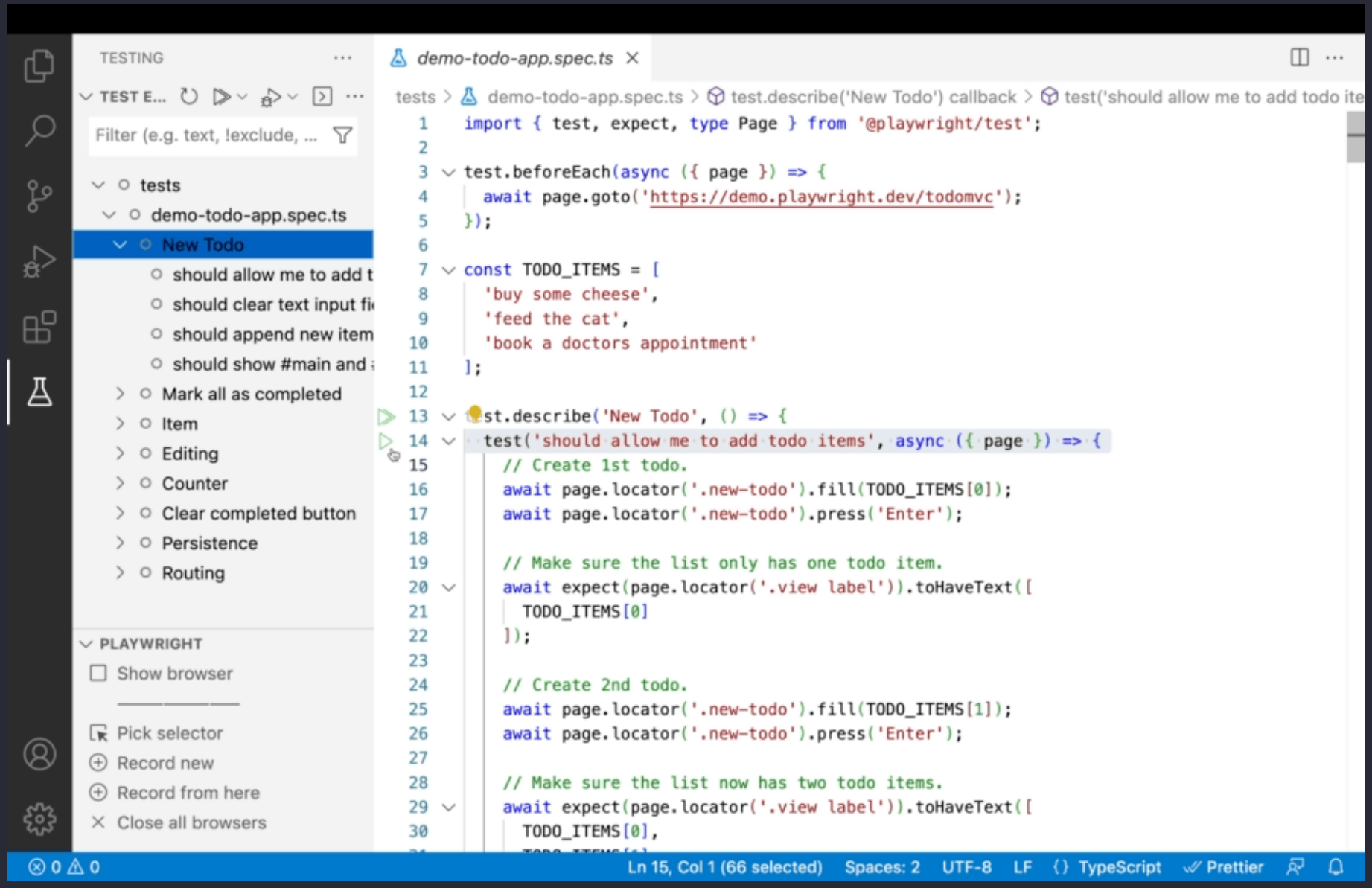
# Why Playwright?

- Web First Assertions: *retry* until timeout
- Locators: Supercharged Element references with *actionability checks!*
- Sharding, Parallelization, automatic browser management all out of the box!
- Context Management, without the hassle.
- Heaps of tools to to make your developer experience better!

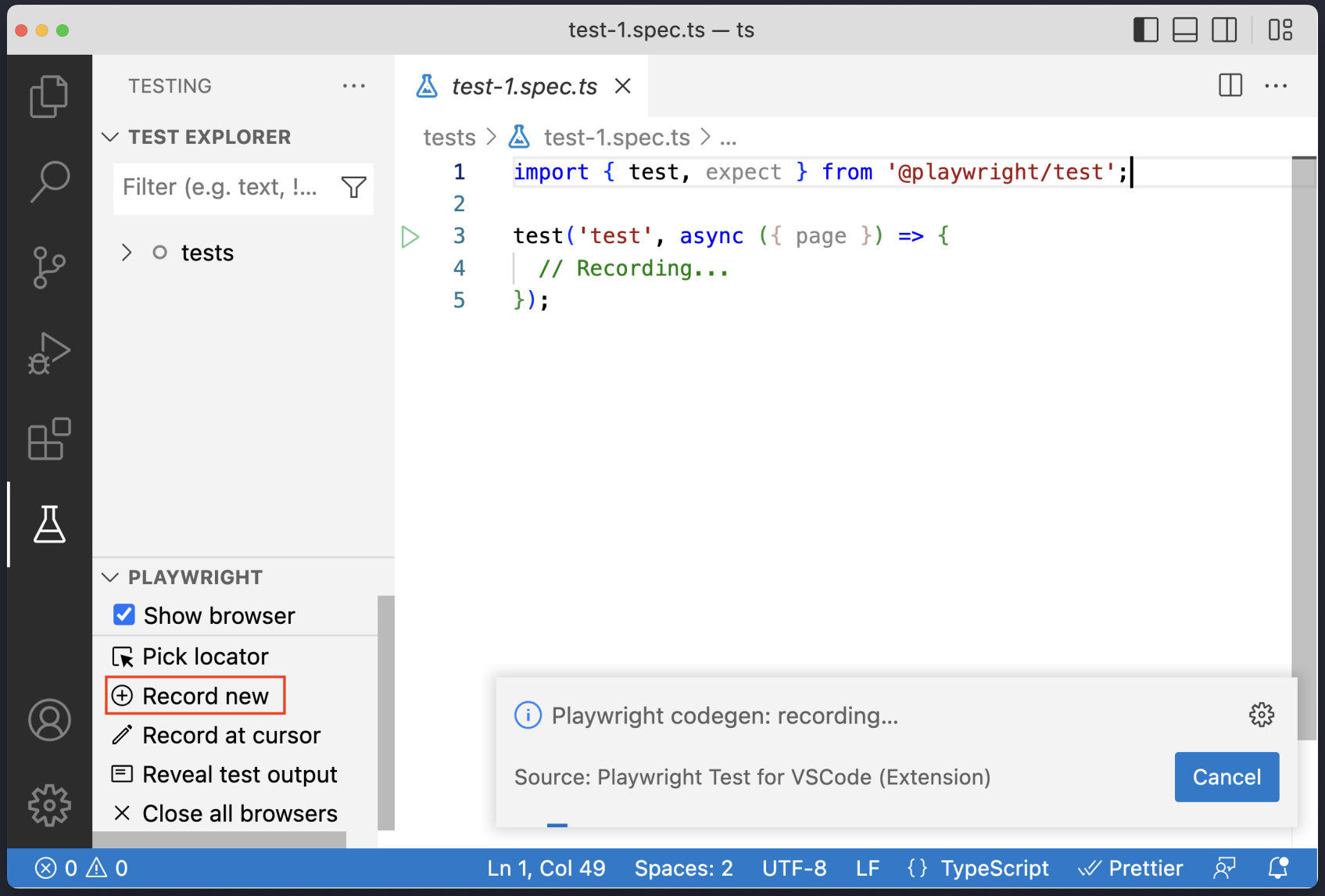
# Tooling

- VSCode Extension
- Code Generator
- Debugging
- Trace Viewer
- UI mode

# VSCode extension

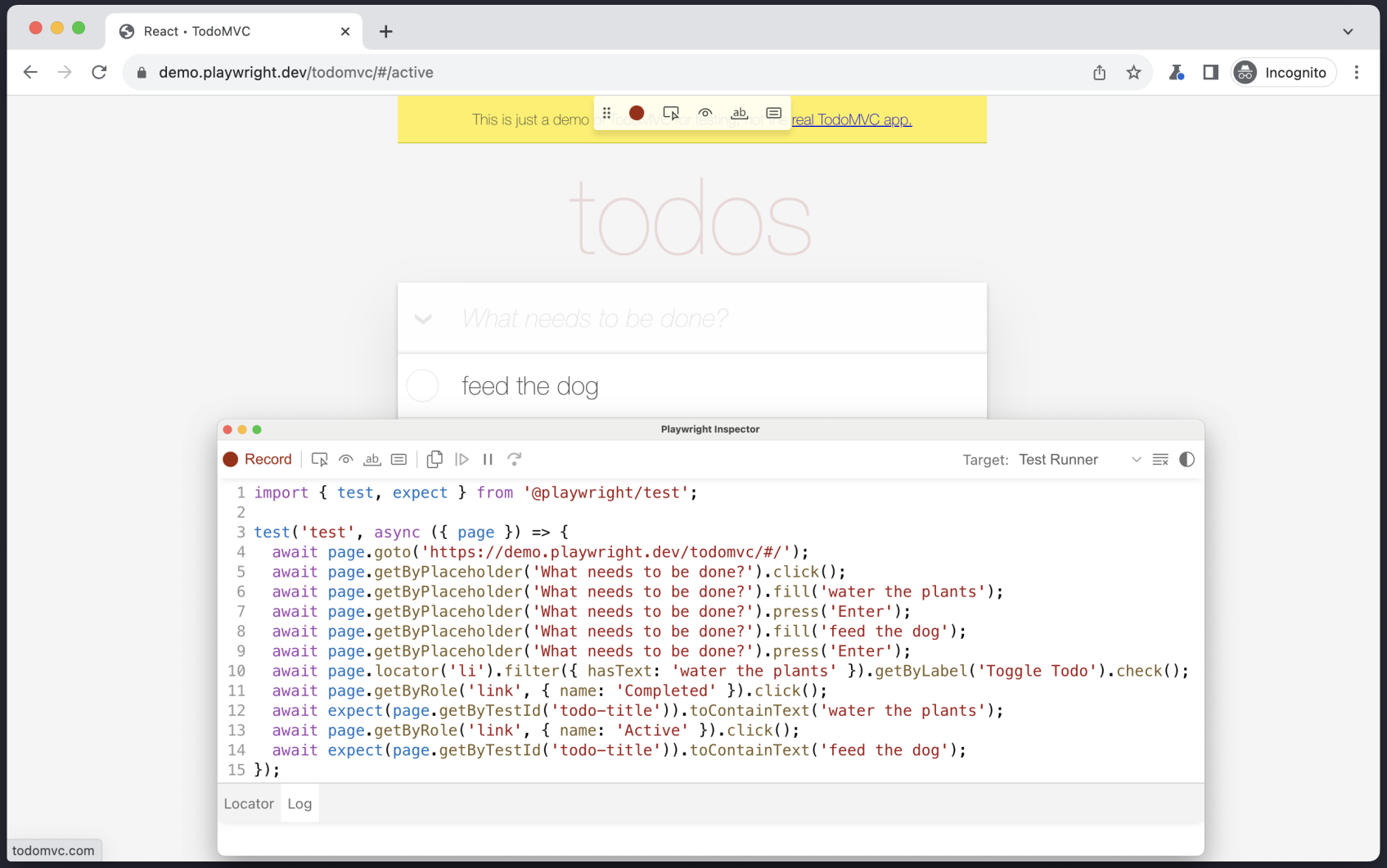

TESTING
TEST E...
Filter (e.g. text, !exclude, ...
tests
demo-todo-app.spec.ts
New Todo
should allow me to add t
should clear text input fi
should append new item
should show #main and
Mark all as completed
Item
Editing
Counter
Clear completed button
Persistence
Routing
PLAYWRIGHT
Show browser
Pick selector
Record new
Record from here
Close all browsers
demo-todo-app.spec.ts
tests > demo-todo-app.spec.ts > test.describe('New Todo') callback > test('should allow me to add todo ite
1 import { test, expect, type Page } from '@playwright/test';
2
3 test.beforeEach(async ({ page }) => {
4 await page.goto('https://demo.playwright.dev/todomvc');
5 });
6
7 const TODO_ITEMS = [
8 'buy some cheese',
9 'feed the cat',
10 'book a doctors appointment'
11 ];
12
13 test.describe('New Todo', () => {
14 test('should allow me to add todo items', async ({ page }) => {
15 // Create 1st todo.
16 await page.locator('.new-todo').fill(TODO_ITEMS[0]);
17 await page.locator('.new-todo').press('Enter');
18
19 // Make sure the list only has one todo item.
20 await expect(page.locator('.view label')).toHaveText([
21 TODO_ITEMS[0]
22 ]);
23
24 // Create 2nd todo.
25 await page.locator('.new-todo').fill(TODO_ITEMS[1]);
26 await page.locator('.new-todo').press('Enter');
27
28 // Make sure the list now has two todo items.
29 await expect(page.locator('.view label')).toHaveText([
30 TODO_ITEMS[0],
0 0
Ln 15, Col 1 (66 selected)
Spaces: 2
UTF-8
LF
TypeScript
Prettier

# Codegen with VSCode extension


test-1.spec.ts — ts
TESTING
TEST EXPLORER
Filter (e.g. text, !...
tests
test-1.spec.ts
tests > test-1.spec.ts > ...
1 import { test, expect } from '@playwright/test';
2
3 test('test', async ({ page }) => {
4 // Recording...
5 });
PLAYWRIGHT
Show browser
Pick locator
Record new
Record at cursor
Reveal test output
Close all browsers
Playwright codegen: recording...
Source: Playwright Test for VSCode (Extension)
Cancel
0 0
Ln 1, Col 49
Spaces: 2
UTF-8
LF
TypeScript
Prettier

# Codegen through CLI

```
npx playwright codegen demo.playwright.dev/todomvc
```


React • TodoMVC
demo.playwright.dev/todomvc/#/active
Incognito
This is just a demo
real TodoMVC app.
todos
What needs to be done?
feed the dog
Playwright Inspector
Record
Target: Test Runner
1 import { test, expect } from '@playwright/test';
2
3 test('test', async ({ page }) => {
4 await page.goto('https://demo.playwright.dev/todomvc/#/');
5 await page.getByPlaceholder('What needs to be done?').click();
6 await page.getByPlaceholder('What needs to be done?').fill('water the plants');
7 await page.getByPlaceholder('What needs to be done?').press('Enter');
8 await page.getByPlaceholder('What needs to be done?').fill('feed the dog');
9 await page.getByPlaceholder('What needs to be done?').press('Enter');
10 await page.locator('li').filter({ hasText: 'water the plants' }).getByLabel('Toggle Todo').check();
11 await page.getByRole('link', { name: 'Completed' }).click();
12 await expect(page.getByTestId('todo-title')).toContainText('water the plants');
13 await page.getByRole('link', { name: 'Active' }).click();
14 await expect(page.getByTestId('todo-title')).toContainText('feed the dog');
15 });
Locator
Log
todomvc.com

# Debugging

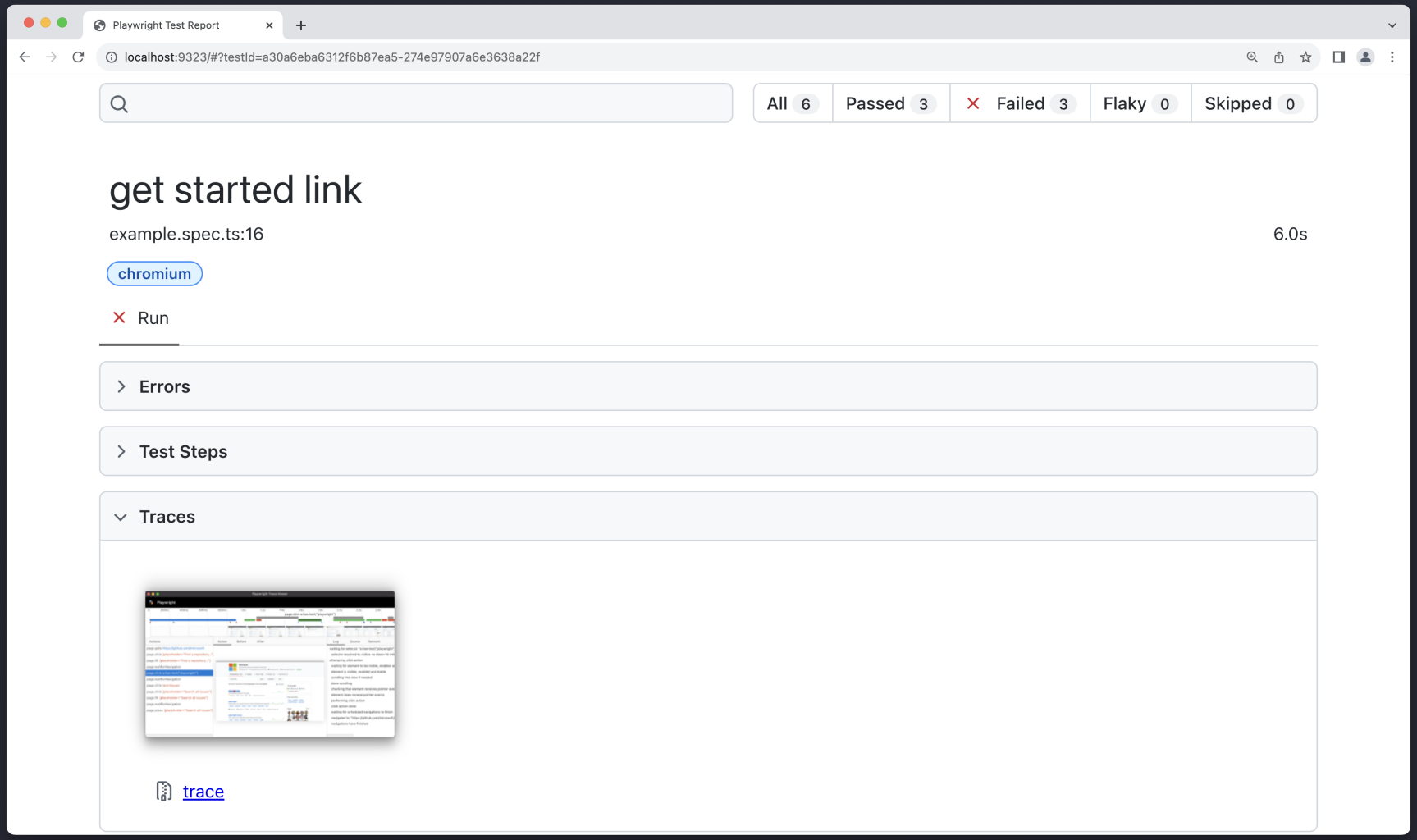
- Various ways to debug tests
  - Breakpoints in VSCode when using the Playwright Test Extension
  - Playwright inspector (run tests with --debug flag). Set Breakpoints with `await page.pause();`
  - UI mode where you can easily walk through each step of the test, see logs, errors, network requests, inspect the DOM snapshot

# Trace Viewer

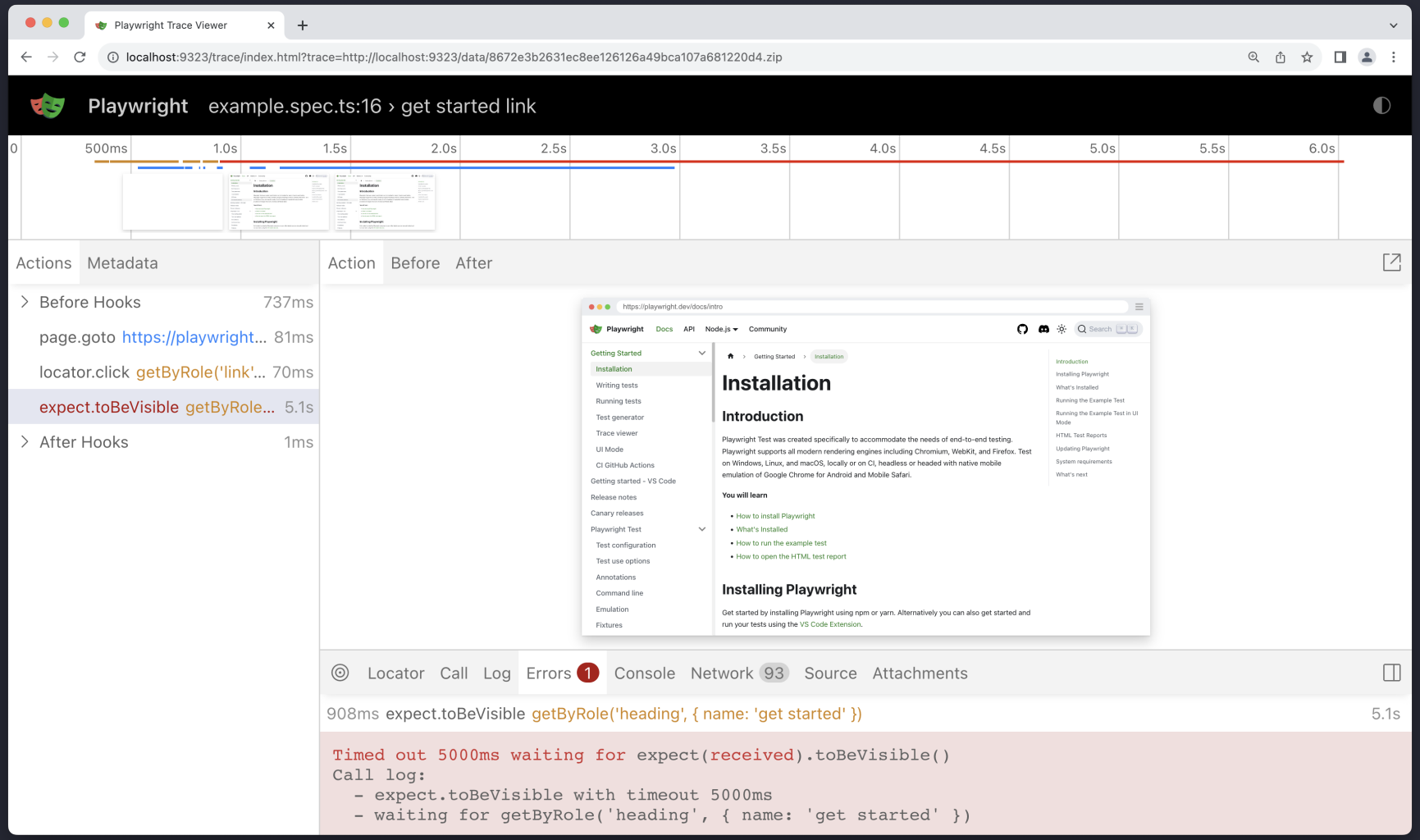
Playwright Trace Viewer is a GUI tool that lets you explore recorded Playwright traces of your tests meaning you can go back and forward through each action of your test and visually see what was happening during each action.

```
import { defineConfig } from '@playwright/test';
export default defineConfig({
  retries: process.env.CI ? 1 : 0, // set to 1 when running on CI
  // ...
  use: {
    trace: 'retain-on-failure', // Retain traces on failure
  },
});
```

# Trace Viewer

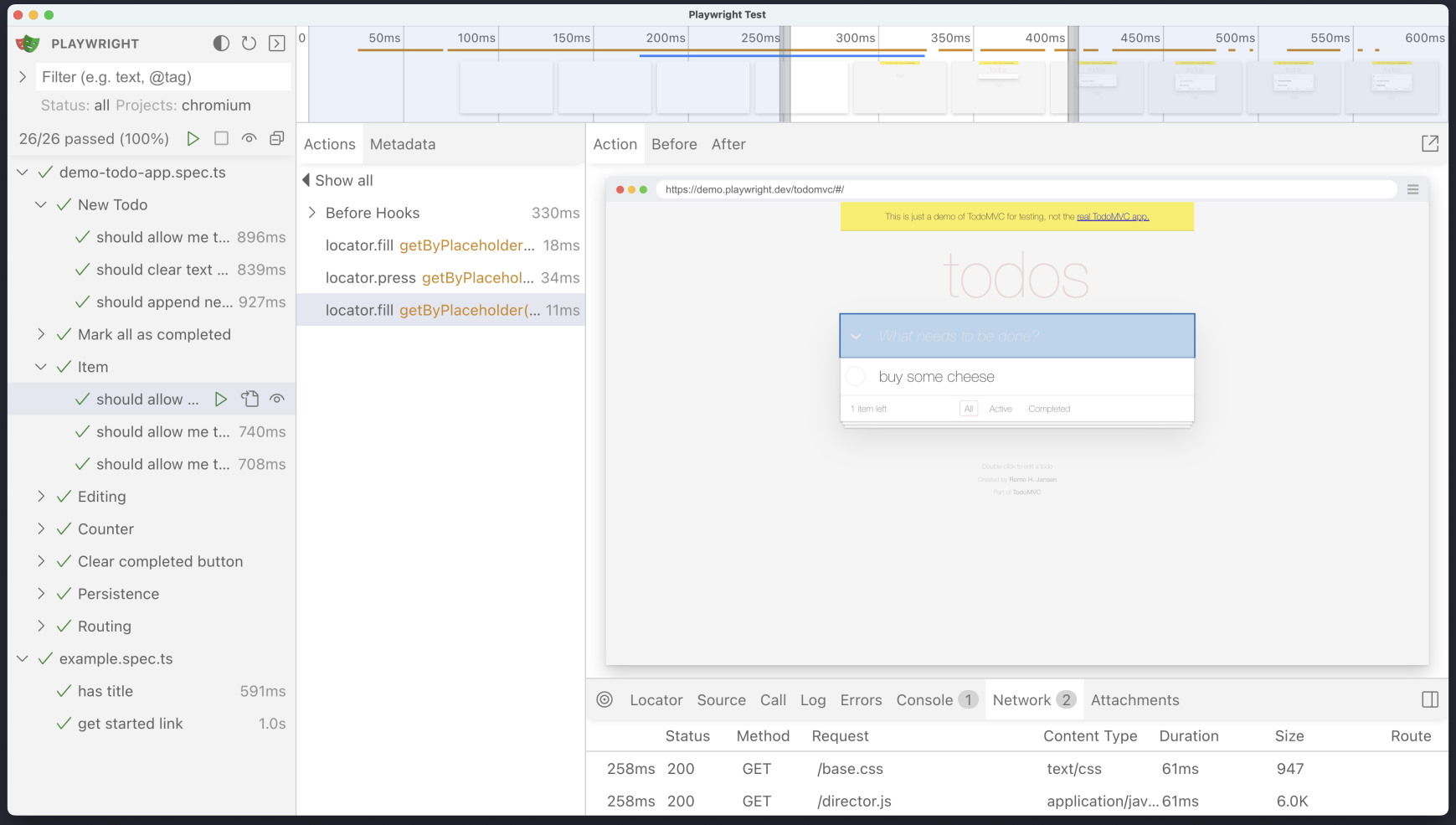


# Trace Viewer



# UI Mode

# npx playwright test --ui



# Basic Usage

- Tests in Playwright
- Playwright API, interacting with your App.
- Locators & ElementHandles
- Web First Assertions
- Setup & Configuration
- Parallelization
- Reports

# Tests in Playwright

```
1  import { test, expect } from '@playwright/test';
2
3  test('has title', async ({ page }) => {
4    await test.step('Navigate to Playwright.dev', async () => {
5        await page.goto('https://playwright.dev/');
6    });
7    await test.step('Page should have title', async () => {
8        // Expect a title "to contain" a substring.
9        await expect(page).toHaveTitle(/Playwright/);
10   });
11 });
```

# Playwright API - Actions

```
Text Input: fill('Peter');
checkbox: setChecked(true);
Dropdown: selectOption('blue');
Click: click();
Key Presses: pressSequentially('Hello World!');
Key Pres: press('Enter');
Drag & Drop: dragTo(page.locator('#item-to-drop-at'));
File upload: setInputFiles(path.join(__dirname, 'myfile.pdf'));
```

*Before interacting with an element, Playwright will perform actionability checks, e.g. check visbility.*

# Locators

Playwright offers two methods for referencing elements

- ElementHandles 👎
- Locators 👍

ElementHandles point to specific elements at a specific point in time. When Using Locators, an up-to-date element is fetched every single time you use it.

# Examples

```
const handle = await page.$('text=Submit');
// ...
await handle.hover();
await handle.click();
```

```
const locator = page.getByText('Submit');
// ...
await locator.hover();
await locator.click();
```

# Best Practices

- Test user-visible behavior
  - Prefer user-facing attributes to XPath or CSS selectors
- Make tests as isolated as possible
- Avoid testing third-party dependencies, only test what *you* control.

# Web First Assertions

- Regular Assertions usually only assert *once*
- Lazy loading or slower websites may result in *flaky* tests.
- Web First assertion(s) continously retry the assertion until the condition is met (or a timeout)
- Input for a web first assertion is a *locator*

# Web First Assertion - Example

```
1 // 👍 Expect a locator to be visible
2 await expect(page.getByText('welcome')).toBeVisible();
3
4
5 // 👎 Expect true/false to be true
6 expect(await page.getByText('welcome').isVisible()).toBe(true);
```

It is considered a best practice to use web first assertions as much as possible!

# Playwright API - API Requests

```
1  const REPO = 'test-repo-1';
2  const USER = 'github-username';
3
4  test('should create a bug report', async ({ request }) => {
5    const newIssue = await request.post(`/repos/${USER}/${REPO}/issues`, {
6      data: {
7        title: '[Bug] report 1',
8        body: 'Bug description',
9      }
10   });
11   expect(newIssue.ok()).toBeTruthy();
12   const JsonResp = await newIssue.json();
13   console.log(JsonResp)
14 });
```

# Playwright API - API Requests

- Additional configuration for things like headers and authentication in `playwright.config.ts`

```ts
import { defineConfig } from '@playwright/test';
export default defineConfig({
  use: {
    // All requests we send go to this API endpoint.
    baseURL: 'https://api.github.com',
    extraHTTPHeaders: {
      // We set this header per GitHub guidelines.
      'Accept': 'application/vnd.github.v3+json',
      // Add authorization token to all requests.
      // Assuming personal access token available in the environment.
      'Authorization': `token ${process.env.API_TOKEN}`,
    },
  }
});
```

# Setup & Configuration

- Initial set up as easy as running: `npm init playwright@latest`
- Configuration exposed through `TestConfig` in `playwright.config.ts`
  - Parallelization
  - Browsers
  - Reporters
  - Global Timeouts

# Parallelization

Playwright runs *files* in parallel. Running *tests* in parallel requires configuration

```
import { test } from '@playwright/test';

test.describe.configure({ mode: 'parallel' });

test('runs in parallel 1', async ({ page }) => { /* ... */ });
test('runs in parallel 2', async ({ page }) => { /* ... */ });
```

```
import { defineConfig } from '@playwright/test';

export default defineConfig({
  fullyParallel: true,
});
```

# Reports - Configuring them

- You can either set the reporter through the CLI: `npx playwright test --reporter=line`
- Or through the `playwright.config.ts`:

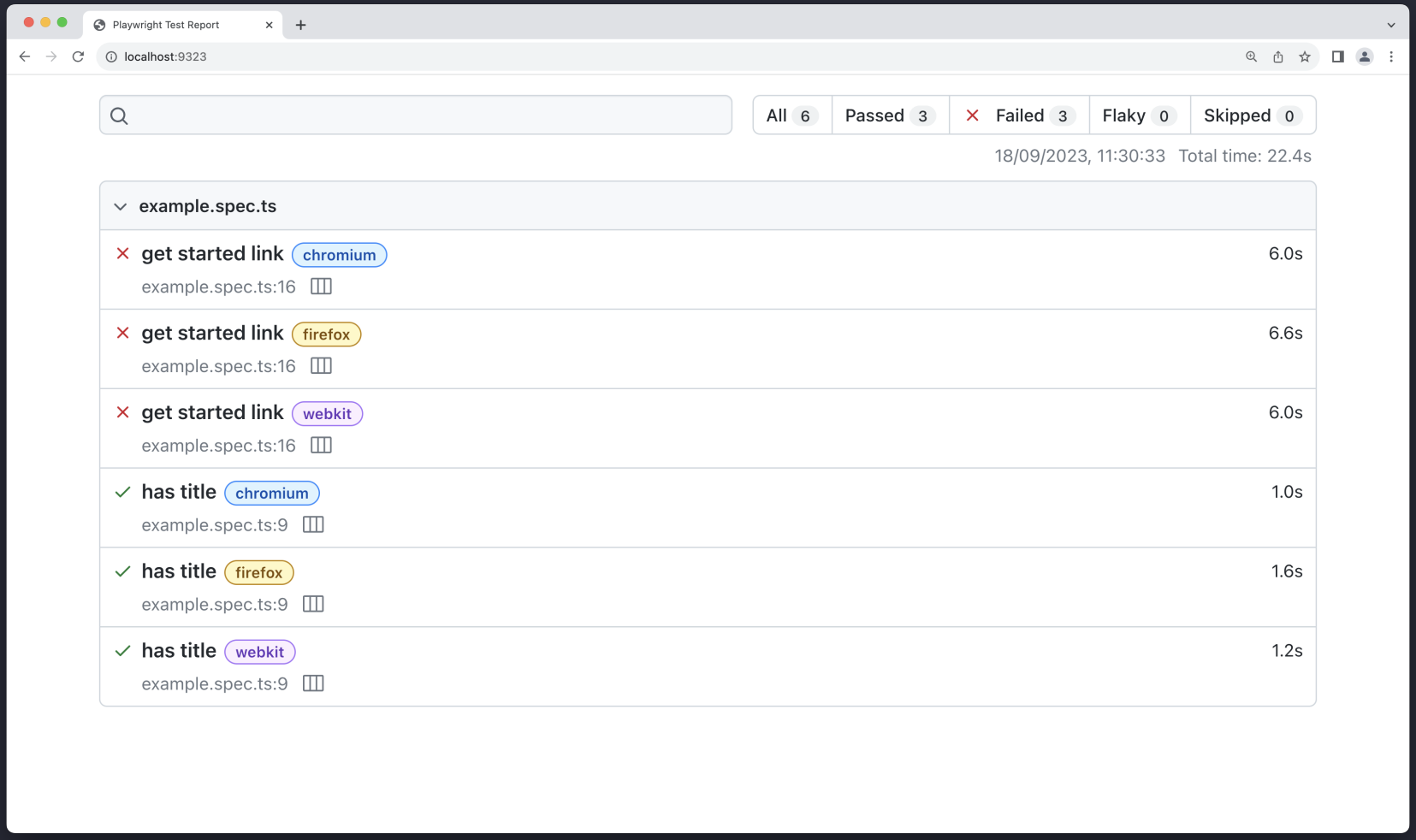
```ts
import { defineConfig } from '@playwright/test';

export default defineConfig({
  reporter: [
    ['list'],
    ['json', {  outputFile: 'test-results.json' }]
  ],
});
```

- You can provide 1 or more reporters.

# HTML reporter

- Contains Test & Step Information
- If enabled, trace files per testcase.



# Community Plugins

# Playwright HTML Report

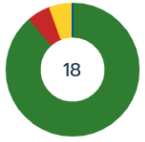
Dashboard | Tests

## Run info

| | |
|---|---|
| Project | *QA Tests* |
| *Release* | *9.87.6* |
| *Test Environment* | *TEST* |
| Execution Start Time | *Mon, 27 Nov 2023 11:18:11 GMT* |
| Execution End Time | *Mon, 27 Nov 2023 11:18:31 GMT* |
| Total Execution Time | *0h 0m 20s 0ms* |

## Summary

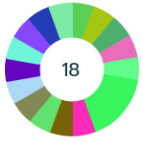
Chart




| Status | Tests | Percentage |
|---|---|---|
| Passed | 16 | *88.89%* |
| Failed | 1 | *5.56%* |
| Timed Out | 1 | *5.56%* |
| Skipped | 0 | *0%* |
| Interrupted | 0 | *0%* |

## Files

Chart




| Filename | Tests | Percentage |
|---|---|---|
| image-download.test.ts | 1 | *5.56%* |
| more-simple.test.ts | 1 | *5.56%* |
| open-two-sites.test.ts | 1 | *5.56%* |
| simple-10.test.ts | 1 | *5.56%* |
| simple-11.test.ts | 1 | *5.56%* |
| other-tests/other.test.ts | 3 | *16.67%* |
| simple-2.test.ts | 1 | *5.56%* |
| simple-3.test.ts | 1 | *5.56%* |

## Browsers

Chart


| Browser | Tests | Percentage |
|---|---|---|
| chromium | 18 | *100%* |

## Status by test file

| Filename | Tests | Execution Time | Passed | Failed | Timed Out | Skipped | Interrupted |
|---|---|---|---|---|---|---|---|
| image-download.test.ts | 1 | 0h 0m 1s 0ms | 1 (*100%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) |
| more-simple.test.ts | 1 | 0h 0m 2s 0ms | 1 (*100%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) |
| open-two-sites.test.ts | 1 | 0h 0m 2s 0ms | 1 (*100%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) |
| simple-10.test.ts | 1 | 0h 0m 2s 0ms | 1 (*100%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) |
| simple-11.test.ts | 1 | 0h 0m 2s 0ms | 1 (*100%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) |
| other-tests/other.test.ts | 3 | 0h 0m 8s 0ms | 3 (*100%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) |
| simple-2.test.ts | 1 | 0h 0m 2s 0ms | 1 (*100%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) |
| simple-3.test.ts | 1 | 0h 0m 2s 0ms | 1 (*100%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) |
| simple-4.test.ts | 1 | 0h 0m 2s 0ms | 1 (*100%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) |
| simple-7.test.ts | 1 | 0h 0m 2s 0ms | 1 (*100%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) |
| image-validation.test.ts | 1 | 0h 0m 2s 0ms | 0 (*0%*) | 1 (*100%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) |
| simple-8.test.ts | 1 | 0h 0m 2s 0ms | 1 (*100%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) |
| simple-9.test.ts | 1 | 0h 0m 2s 0ms | 1 (*100%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) |
| very-simple.test.ts | 1 | 0h 0m 1s 0ms | 1 (*100%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) |
| simple-5.test.ts | 1 | 0h 0m 2s 0ms | 1 (*100%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) | 0 (*0%*) |
| simple-6.test.ts | 1 | 0h 0m 10s 0ms | 0 (*0%*) | 0 (*0%*) | 1 (*100%*) | 0 (*0%*) | 0 (*0%*) |

## Status by browser

| Browser | Tests | Execution Time | Passed | Failed | Timed Out | Skipped | Interrupted |
|---|---|---|---|---|---|---|---|
| chromium | 18 | 0h 0m 19s 0ms | 16 (*88.89%*) | 1 (*5.56%*) | 1 (*5.56%*) | 0 (*0%*) | 0 (*0%*) |

# Assignment 1.

Checkout Repository:
https://github.com/Ghislain89/PlaywrightWorkshop

- Run `npm install`
- Run `npm run build`
- Start webApp by running `npm run start`

The webApp should start on localhost:3000

# Assignment 1

- Write a testcase to automate the registration and login process for a new account.
- Set up Playwright to save HTML reports for your tests.
- Configure Playwright to always capture and store execution traces for better debugging and analysis.
- Duplicate the testcase a few times
- Enable parallel execution for these testcases
- Did you encounter any issues?

# Advanced Usage

- Network Monitoring & Manipulation
- Reusing authentication sessions
- Page objects
- Fixtures

# Network Monitoring & Manipulation

# Waiting for Requests

```
1  // Start waiting for request before clicking. Note no await.
2  const requestPromise = page.waitForRequest('https://example.com/resource');
3  await page.getByText('trigger request').click();
4  const request = await requestPromise;
5
6  // Alternative way with a predicate. Note no await.
7  const requestPromise = page.waitForRequest(request =>
8    request.url() === 'https://example.com' && request.method() === 'GET',
9  );
10 await page.getByText('trigger request').click();
11 const request = await requestPromise;
```

# Waiting for Responses

```
1  // Start waiting for response before clicking. Note no await.
2  const responsePromise = page.waitForResponse('https://example.com/resource')
3  await page.getByText('trigger response').click();
4  const response = await responsePromise;
5
6  // Alternative way with a predicate. Note no await.
7  const responsePromise = page.waitForResponse(response =>
8    response.url() === 'https://example.com' && response.status() === 200
9  );
10 await page.getByText('trigger response').click();
11 const response = await responsePromise;
```

# Mocking Responses

```
await page.route('**/api/fetch_data', route => route.fulfill({
  status: 200, // Set an explicit response code
  body: testData, // Put whatever data you need here!
}));
await page.goto('https://example.com');
```

# Modifying Responses

```
test('gets the json from api and adds a new fruit', async ({ page }) => {
  // Get the response and add to it
  await page.route('*/**/api/v1/fruits', async route => {
    const response = await route.fetch();
    const json = await response.json();
    json.push({ name: 'Playwright', id: 100 });
    // Fulfill using the original response, while patching the response body
    // with the given JSON object.
    await route.fulfill({  status: 200, response, json });
  });

  // Go to the page
  await page.goto('https://demo.playwright.dev/api-mocking');

  // Assert that the new fruit is visible
```

# Assignment 2

Attempting to create a user that already exists results in a HTTP Statuscode *409* on the register endpoint

- Duplicate the testcase you created earlier
- Apply response modification to trigger a 409 and a toasters that shows the user already exists.

*Especially for toasters, web first assertions are your friend!*

# Reusing authentication sessions

# Storing Authentication State

```ts
//auth.setup.ts
import { test as setup, expect } from '@playwright/test';

const authFile = 'playwright/.auth/user.json';

setup('authenticate', async ({ page }) => {
  await page.goto('https://github.com/login');
  await page.getByLabel('Username or email address').fill('username');
  await page.getByLabel('Password').fill('password');
  await page.getByRole('button', { name: 'Sign in' }).click();
  await page.waitForURL('https://github.com/');
  await expect(page.getByRole('button', { name: 'View profile and more' })).
  await page.context().storageState({ path: authFile });
});
```

# Using Authentication State

```
import { defineConfig, devices } from '@playwright/test';

export default defineConfig({
  projects: [
    // Setup project
    { name: 'setup', testMatch: /.*\.setup\.ts/ },

    {
      name: 'chromium',
      use: {
        ...devices['Desktop Chrome'],
        // Use prepared auth state.
        storageState: 'playwright/.auth/user.json',
      },
      dependencies: ['setup'],
```

# Assignment 3

- Write a setup testcase that stores authentication state
- Update the testcase you created earlier to make use of this authenticated state
- Run your testcase again!

# Page Objects (React Example)

- Page Object Classes describing interactions for *each* component
- Page Object Classes describing interactions for *each* page, typically uses 1 or more components
- Tests only use interactions/functions and do not reference elements directly.
- *KISS*

Let's look at an example!

≡

# Fixtures

- Fixtures can contain just about anything you want
  - Testdata
  - Helper(s)
  - Page Objects!
- Fixtures are scoped to the consuming testcase.
- The *page* we've been seeing in many slides, is one of Playwrights built in fixtures.

# Page Objects without fixtures

```
test.describe('todo tests', () => {
  let todoPage;

  test.beforeEach(async ({ page }) => {
    todoPage = new TodoPage(page);
    await todoPage.goto();
    await todoPage.addToDo('item1');
    await todoPage.addToDo('item2');
  });
  test('should add an item', async () => {
    await todoPage.addToDo('my item');
    // ...
  });
});
```

# Page Objects with fixtures

```ts
import { test as base } from '@playwright/test';
import { TodoPage } from './todo-page';

// Extend basic test by providing a "todoPage" fixture.
const test = base.extend<{ todoPage: TodoPage }>({
  todoPage: async ({ page }, use) => {
    const todoPage = new TodoPage(page);
    await todoPage.goto();
    await todoPage.addToDo('item1');
    await todoPage.addToDo('item2');
    await use(todoPage);
    await todoPage.removeAll();
  },
});
```

# Page Objects with fixtures

```
test('should add an item', async ({ todoPage }) => {
  await todoPage.addToDo('my item');
  // ...
});

test('should remove an item', async ({ todoPage }) => {
  await todoPage.remove('item1');
  // ...
});
```

# Assignment 4

- Refactor your testcase to use the page objects I defined.
- Try to use Fixtures!
- Expand the testcase to also add a Todo and assert that your todo was succesfully created.

*hint*: Some preperation has already been done

# Bonus

- Visual Regression Testing
- Accessability Testing
- Component Testing [out of scope for today]
- CI Integration
- Linting

# Visual Regression Testing

```
1 import { test, expect } from '@playwright/test';
2
3 test('example test', async ({ page }) => {
4   await page.goto('https://playwright.dev');
5   await expect(page).toHaveScreenshot();
6 });
```

*"Error: A snapshot doesn't exist at example.spec.ts-snapshots/example-test-1-chromium-darwin.png, writing actual."*

# Assignment 5

- Create a testcase that takes a screenshot of the login page
- Run it once to generate the snapshot
- Run it a second time, does it pass?

# Visual Regression Testing - The Challenges

- Dynamic or volatile elements change often, causing mismatches
  - Apply Custom CSS to hide these elements
  - Mask elements by locator
- Snapshots are unique per browser and operating system.
- Locally generated screenshots (on Windows/macOS) won't pass in (Linux) CI.

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

test('example test', async ({ page }) => {
  await page.goto('https://playwright.dev');
  await expect(page).toHaveScreenshot({ stylePath: path.join(__dirname, 'scre
});
```

```
await page.goto('https://playwright.dev');
await expect(page).toHaveScreenshot({
  mask: [page.locator('img')],
  maskColor: '#00FF00', // green
});
```

# Assignment 5

- Mask the volatile element!
- Update the snapshot
- Run it a second time, does your test pass?

# Accessability Testing

- Playwright needs an additional library like *Axe* to run accessability tests.

```
1  import { test, expect } from '@playwright/test';
2  import AxeBuilder from '@axe-core/playwright'; // 1
3
4  test.describe('homepage', () => { // 2
5    test('should not have any automatically detectable accessibility issues',
6      await page.goto('https://your-site.com/'); // 3
7      const accessibilityScanResults = await new AxeBuilder({ page }).analyze(
8      expect(accessibilityScanResults.violations).toEqual([]); // 5
9    });
10 });
```

# Assignment 7

- Create a testcase that adds a few todos.
- Once the Todos are created, run an accessability scan with Axe.
- How many violations are there?

# CI Integration

- Setting everything up manually

```
# Install NPM packages
npm ci

# Install Playwright browsers and dependencies
npx playwright install --with-deps
```

- Using official Docker Container

```yaml
on:
  push:
    branches: [ main, master ]
  pull_request:
    branches: [ main, master ]
jobs:
  playwright:
    name: 'Playwright Tests'
    runs-on: ubuntu-latest
    container:
      image: mcr.microsoft.com/playwright:v1.41.1-jammy
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: 18
```

# Assingment 8

- Try and get the Github Actions workflow running!

# Linting

- Mistakes are easy to make
- Best practices are hard to implement, even harder to maintain

Linting *can* help with this. Playwright supplies an ESLint plugin that enforces some best practices example:

Prevent a `test.only` from being comitted

- Plugin is *very* opinionated, might not work well for everyone.

Find it on NPM: `eslint-plugin-playwright`

≡

# Online Sources

1. https://playwright.dev
2. Discord: https://playwright.dev/community/welcome#community-discord
3. Stack overflow: https://stackoverflow.com/tags/playwright
4. Playwright Solutions: https://playwrightsolutions.com/ 5 Community Reporter: https://rodrigoodhin.gitlab.io/playwright-html/#/1.1.5/screenshots

≡

# Thank You!

Repositories, assignments our implementation and PDF of the presentation will be shared via e-mail!

Feedback or Questions? Let us know!

Ghislain@DeTesters.nl Jurgen@DeTesters.nl