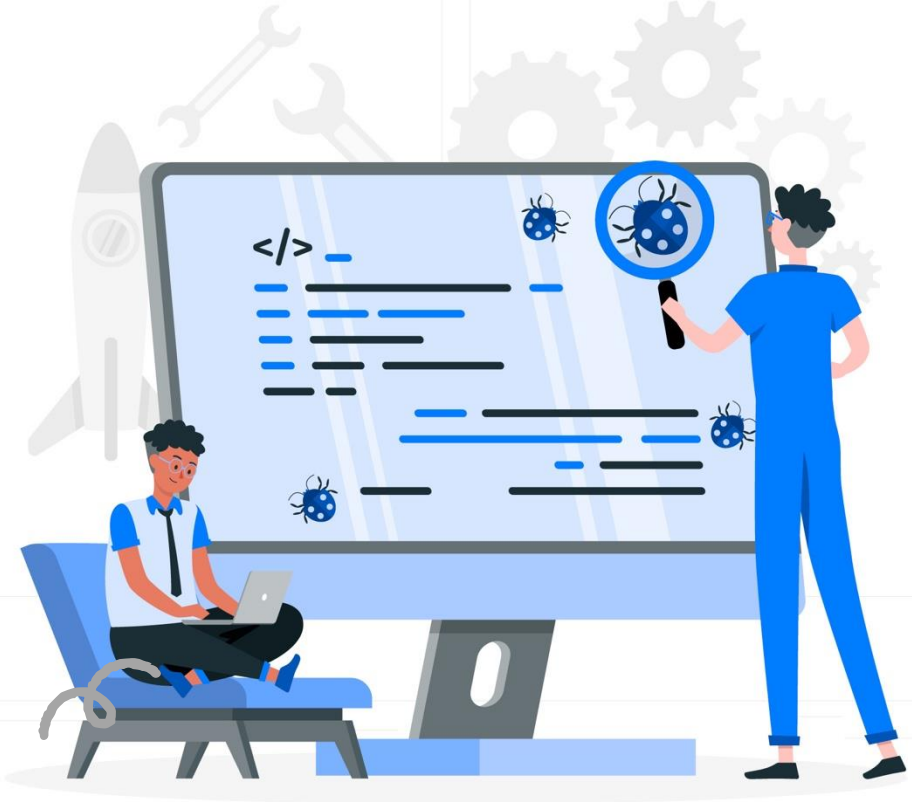
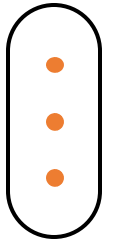
Workshop

Playwright



Playwright

Why End-to-End Testing



End-to-end (E2E) testing is the practice of testing a software application's flow from beginning to end to ensure the behavior is as expected.

Key Reasons to Use E2E Testing

- **Mimics Real User Behavior:** E2E tests follow the exact same path a user would take on your website. They click buttons, fill out forms, and navigate between pages, ensuring the entire user journey works flawlessly.
- **Catches Bugs That Matter:** While unit tests and integration tests are important, they can't catch bugs that occur from the interaction between multiple components. E2E tests are crucial for finding these critical, user-facing bugs.
- **Builds Developer Confidence:** When your E2E tests pass, you can be confident that your application is stable and ready to be deployed to production. This reduces the risk of new features breaking existing ones.

What is Playwright?

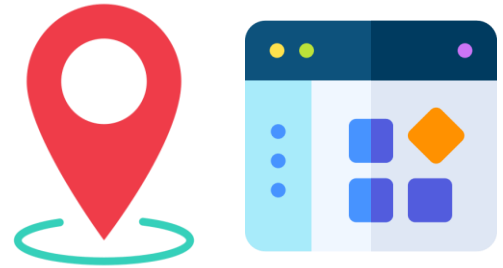
Playwright is a powerful and reliable open-source framework for End-to-End testing. It was developed by Microsoft to create an even more advanced and reliable web testing tool, addressing many of the limitations of existing frameworks.

Key Features

- **Multi-Browser:** Supports Chromium (Chrome), WebKit (Safari), and Firefox.
- **Cross-Platform:** Works seamlessly on Windows, macOS, and Linux.
- **Language-Agnostic:** Write tests in your preferred language: JavaScript/TypeScript, Python, Java, or C#.
- **Packed with Tools:** It comes with a built-in Codegen to generate tests, an Inspector for debugging, and a Trace Viewer to analyze test failures.

Key Components of a Test Script(1/3)

Every Playwright test script is built on three fundamental pillars. Understanding these components is essential for writing effective and reliable tests.



1. Locator: Finding Elements

A **Locator** is how Playwright finds elements on the page. Instead of using complex selectors, Playwright's locators are designed to be resilient to changes, making your tests more stable.

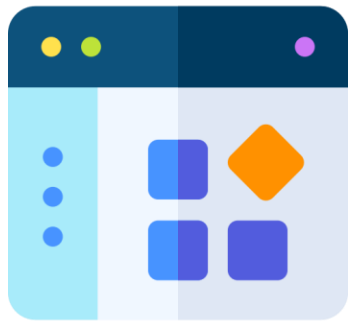
- **Example:** `page.getByPlaceholder('What needs to be done?')`
- **Purpose:** This locator finds the input field that has the exact placeholder text "What needs to be done?".

Key Components of a Test Script(2/3)

2. Action: Interacting with Elements

An **Action** is what you do with the located element. This simulates user interactions like typing, clicking, or hovering.

- **Example:** `.fill('Playwright Workshop')` followed by `.press('Enter')`
- **Purpose:** It types text into the input field and then presses the "Enter" key.



Key Components of a Test Script(3/3)

3. Assertion: Verifying the Outcome

An **Assertion** is the final and most crucial step. It verifies that the application's state is what you expect after an action is performed. This is how you confirm if the test passed or failed.

- **Example:** `expect(page.locator('.todo-list li')).toHaveText('Playwright Workshop’);`
- **Purpose:** It checks if the new to-do item appears in the list with the correct text.

Example Test Script

example.spec.js

tests > example.spec.js > ...

```
1   // @ts-check
2   import { test, expect } from '@playwright/test';
3
4 > test('has title', async ({ page }) => {…
9   });
10
11  test('get started link', async ({ page }) => {
12    await page.goto('https://playwright.dev/');
13
14    // Click the get started link.
15    await page.getByRole('link', { name: 'Get started' }).click();
16
17    // Expects page to have a heading with the name of Installation.
18    await expect(page.getByRole('heading', { name: 'Installation' })).toBeVisible();
19  });
20
```

Name

use to perform actions and assert expectations

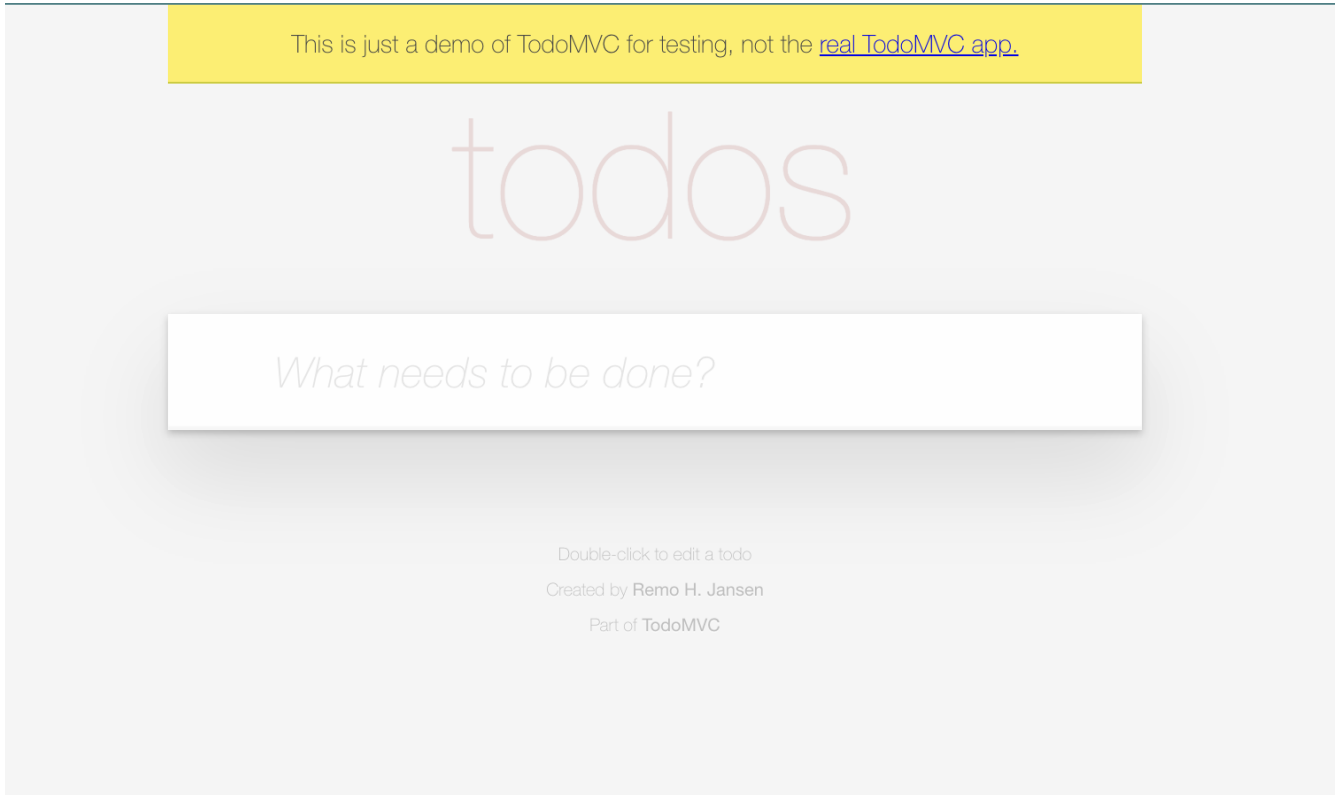
Navigation method

Action

Locator

Assertion

Todos



https://demo.playwright.dev/todomvc/#/

Create Playwright Project

```
$ mkdir playwright-todo-workshop
```

```
$ cd playwright-todo-workshop
```

```
$ npm init playwright@latest
```

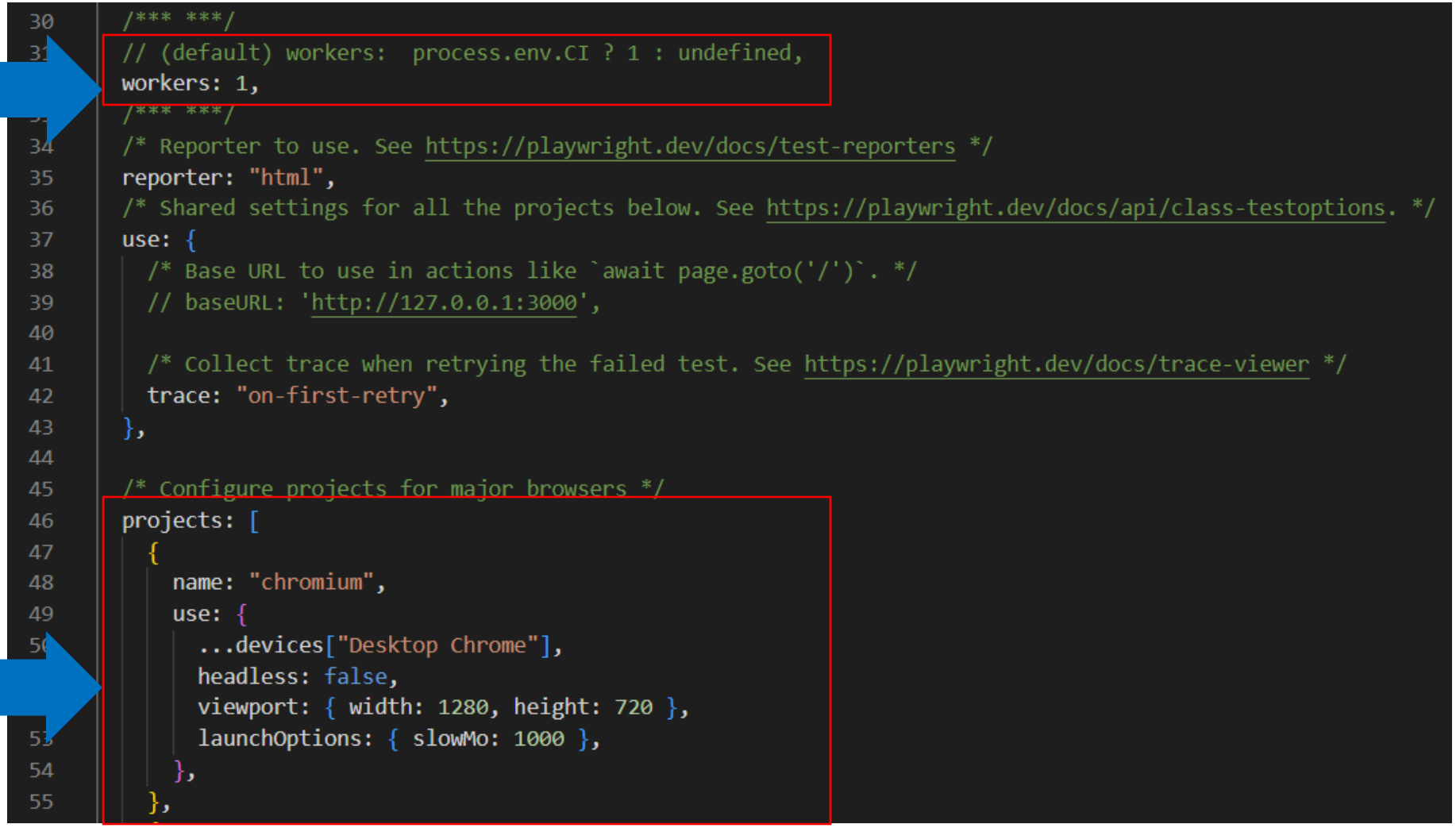
Setup Questions

- "Do you want to use TypeScript or JavaScript?" JavaScript
- "Where to put your end-to-end tests?” Press Enter
- "Add a GitHub Actions workflow?” No
- "Install Playwright browsers (chromium, firefox, webkit)?" Yes

Playwright Project

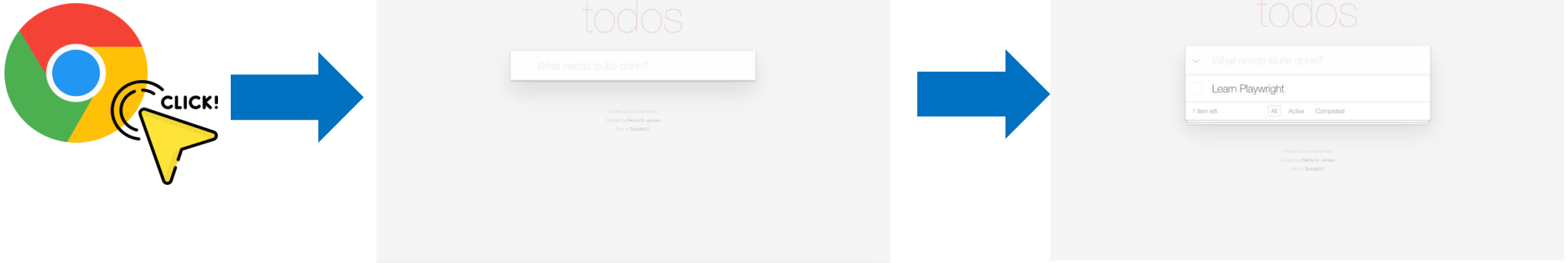
- **node_modules/:** This folder contains all of the project's dependencies.
- **tests/:** This is where you'll put your test script files.
- **tests-examples/:** This folder holds example tests to help you get started.
- **package.json:** This file stores project metadata and lists all of its dependencies.
- **playwright.config.js:** This is the main configuration file for Playwright.

Playwright Config

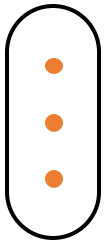


```
30  /*** ***/
31  // (default) workers:  process.env.CI ? 1 : undefined,
    workers: 1,
    /*** ***/
34  /* Reporter to use. See https://playwright.dev/docs/test-reporters */
35  reporter: "html",
36  /* Shared settings for all the projects below. See https://playwright.dev/docs/api/class-testoptions. */
37  use: {
38    /* Base URL to use in actions like `await page.goto('/')`. */
39    // baseURL: 'http://127.0.0.1:3000',
40
41    /* Collect trace when retrying the failed test. See https://playwright.dev/docs/trace-viewer */
42    trace: "on-first-retry",
43  },
44
45  /* Configure projects for major browsers */
46  projects: [
47    {
48      name: "chromium",
49      use: {
50        ...devices["Desktop Chrome"],
          headless: false,
          viewport: { width: 1280, height: 720 },
          launchOptions: { slowMo: 1000 },
54      },
55    },
```

Test Scenario



Create test file& test script



todo.spec.js

tests > todo.spec.js > ...

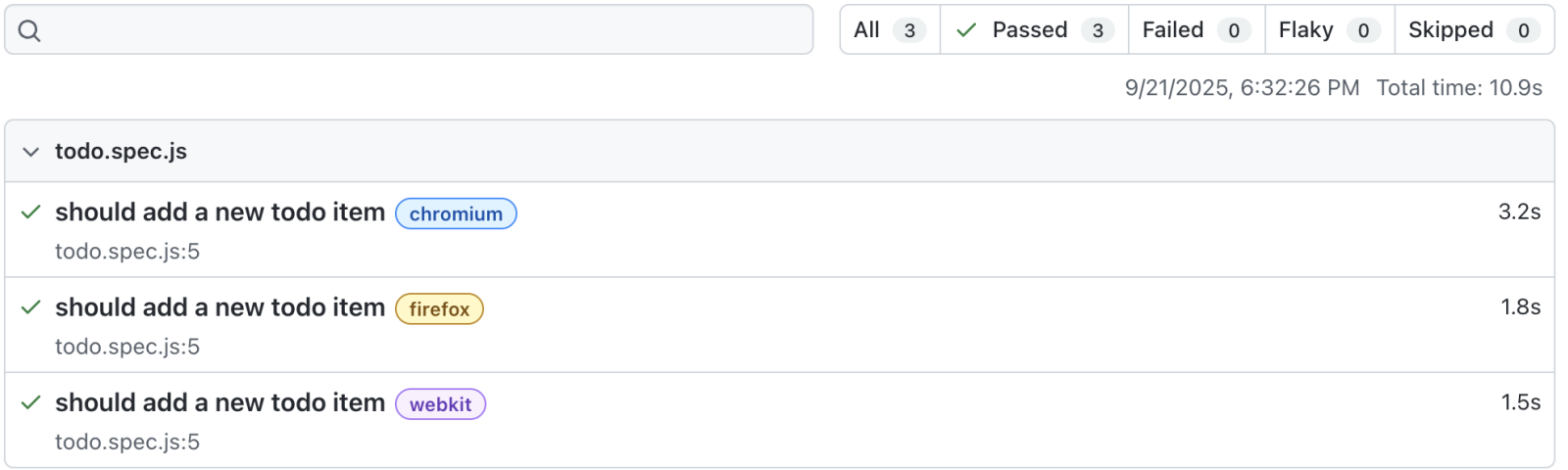
```
1  // tests/todo.spec.js
2  const { test, expect } = require('@playwright/test');
3
4  // Create a new test case with a descriptive name.
5  test('should add a new todo item', async ({ page }) => {
6      // 1. Action: Navigate to the To-Do MVC website.
7      await page.goto('https://demo.playwright.dev/todomvc/#/');
8
9      // 2. Locator + Action: Find the input field and type a new to-do item.
10     const newTodoInput = page.getByPlaceholder('What needs to be done?');
11     await newTodoInput.fill('Learn Playwright');
12     await newTodoInput.press('Enter');
13
14     // 3. Assertion: Verify that the new item appears in the list.
15     const todoItem = page.locator('.todo-list li');
16     await expect(todoItem).toHaveText('Learn Playwright');
17 });
```

Run Test

```
$ npx playwright test tests/todo.spec.js
```

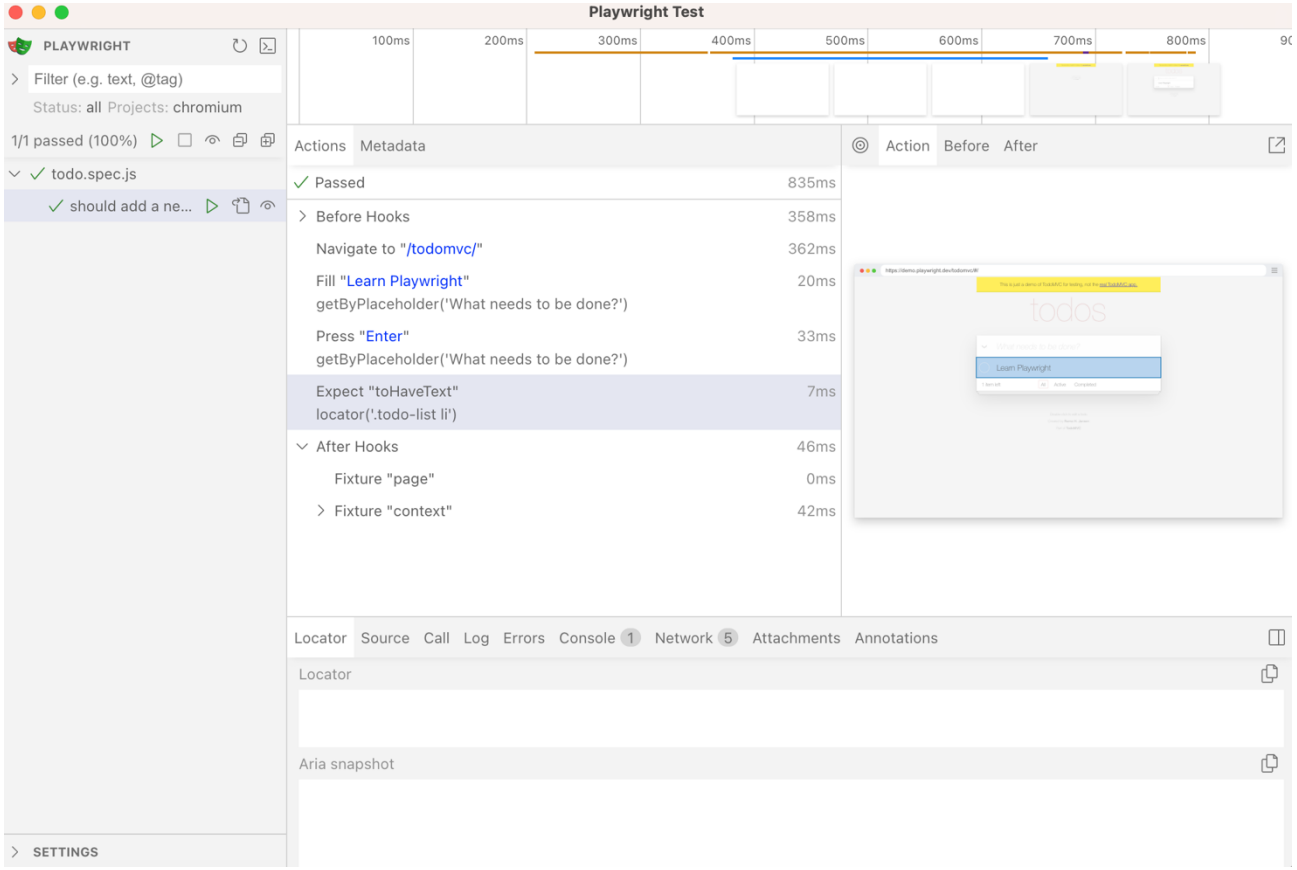
```
Running 3 tests using 3 workers
  3 passed (10.9s)

To open last HTML report run:

  npx playwright show-report
```

```
$ npx playwright show-report
```



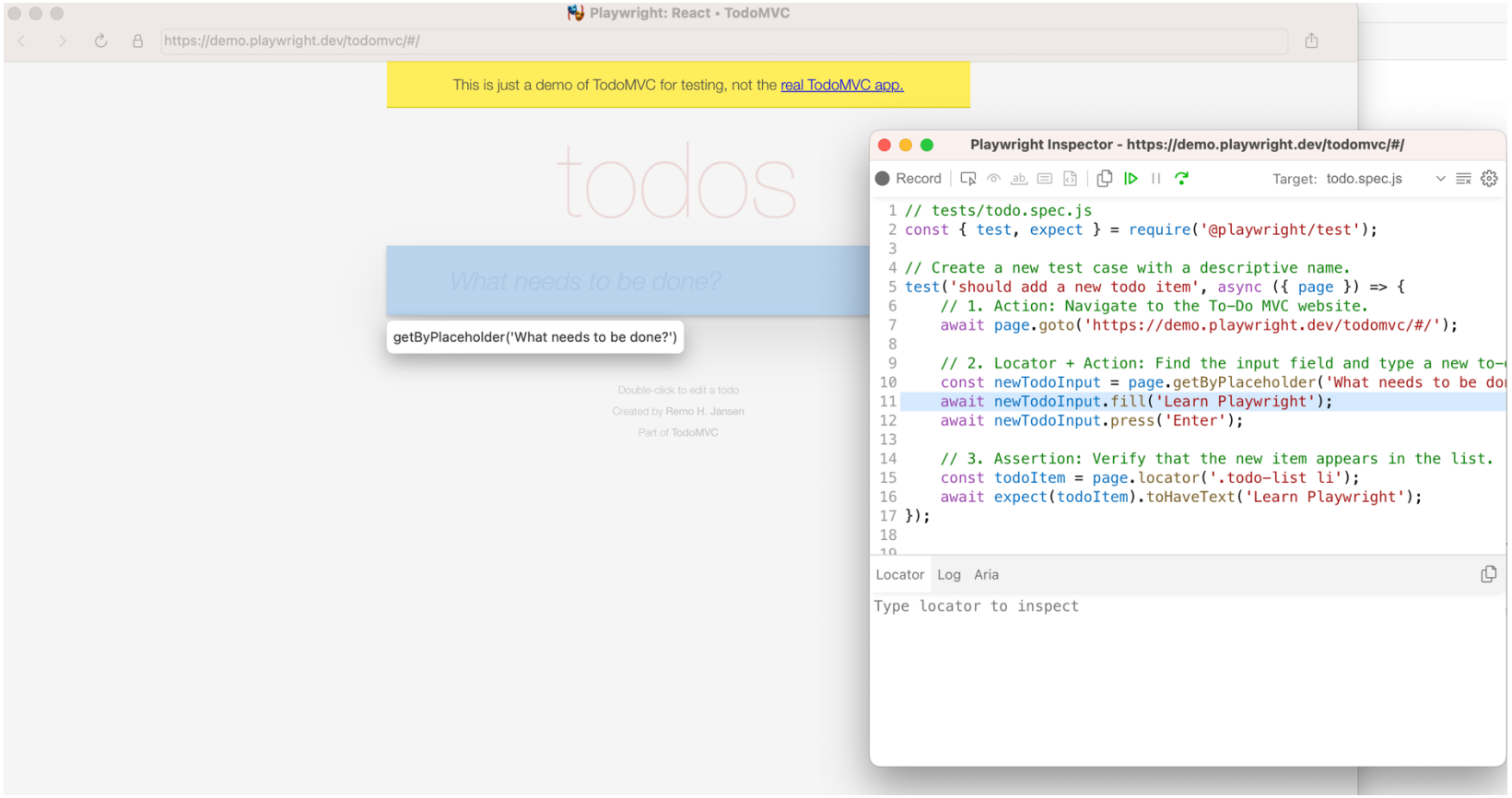
Run Test with UI

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



Run Test with Debug Mode

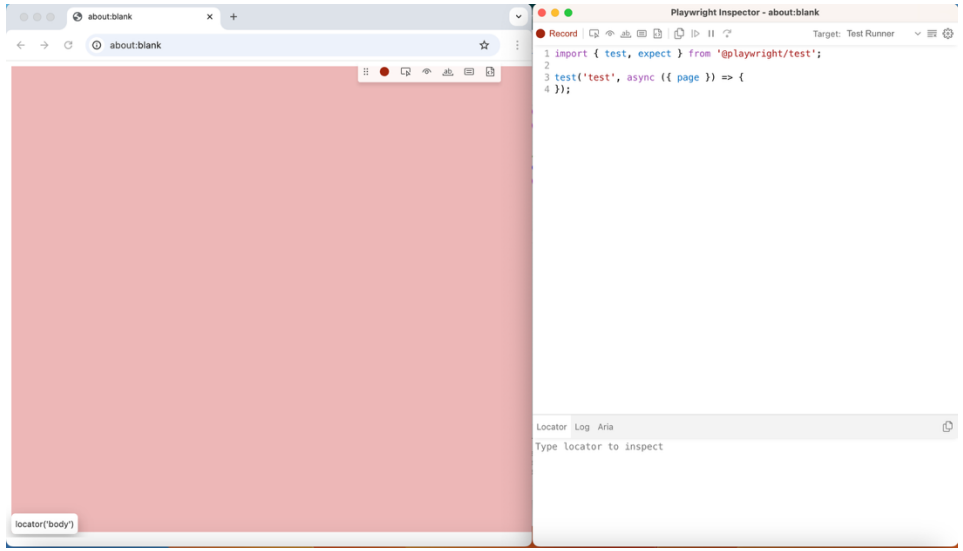
```
$ npx playwright test tests/todo.spec.js --debug
```



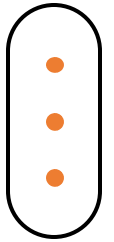
Codegen

```
$ npx playwright codegen
```

```
https://demo.playwright.dev/todomvc/#/
```



Practice



Add one more to-do item

Add “Delete to do item” test case