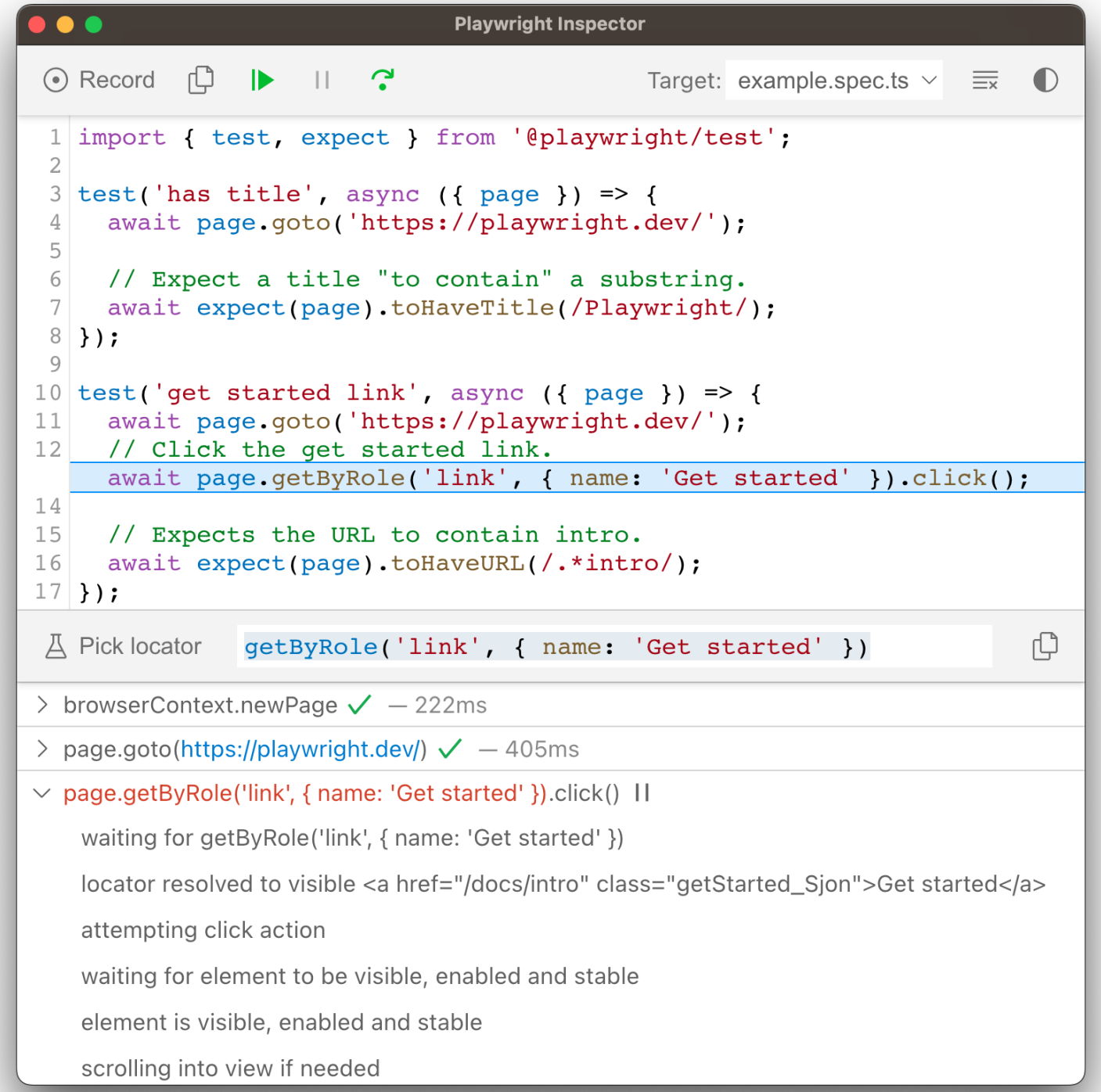
Debugging Tests

Playwright Inspector

The Playwright Inspector is a GUI tool to help you debug your Playwright tests. It allows you to step through your tests, live edit locators, pick locators and see actionability logs.



Run in debug mode

Set the `PWDEBUG` environment variable to run your Playwright tests in debug mode. This configures Playwright for debugging and opens the inspector. Additional useful defaults are configured when `PWDEBUG=1` is set:

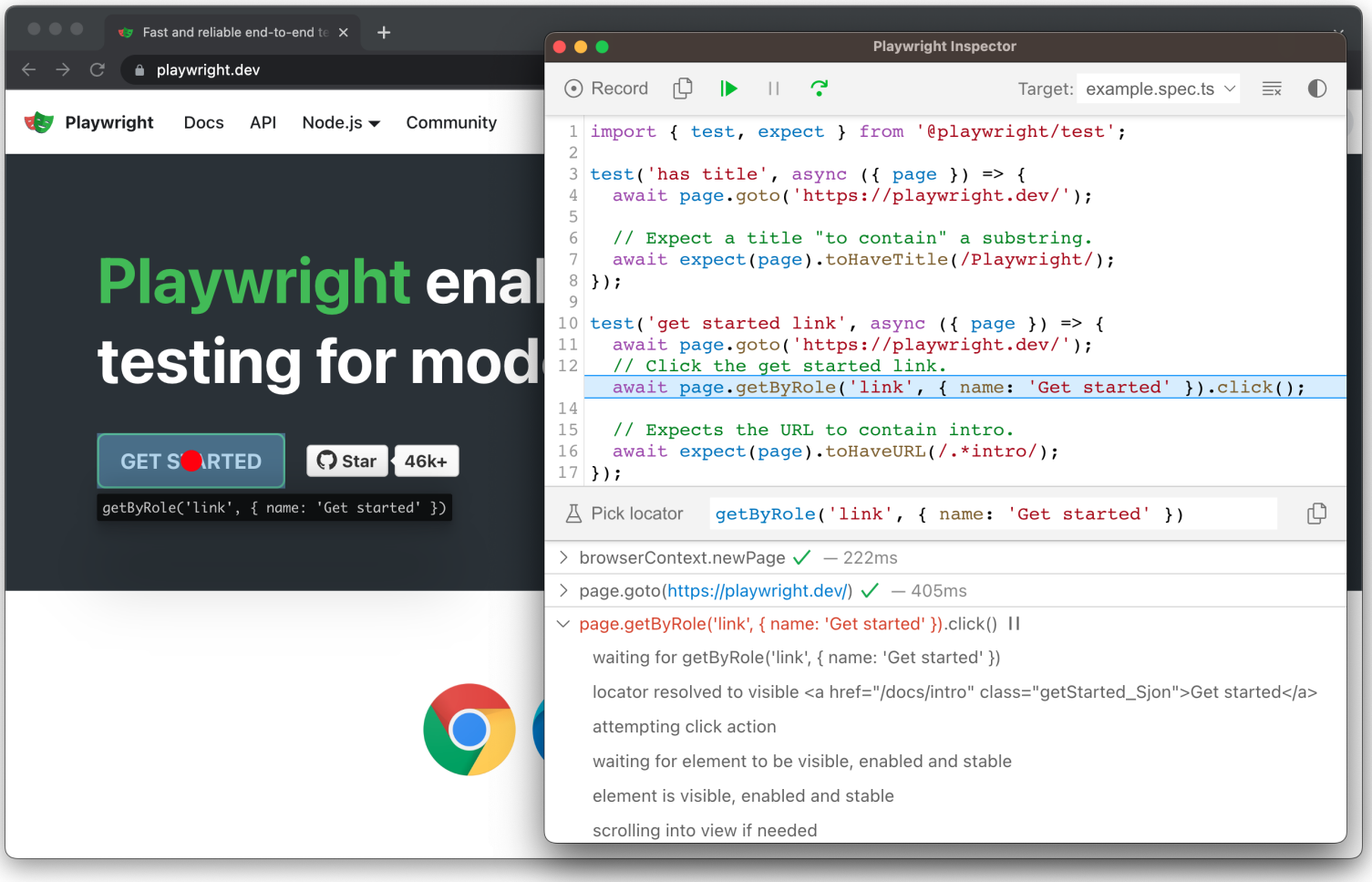
- Browsers launch in headed mode
- Default timeout is set to 0 (= no timeout)

Bash PowerShell Batch

```
PWDEBUG=1 pytest -s
```

Stepping through your tests

You can play, pause or step through each action of your test using the toolbar at the top of the Inspector. You can see the current action highlighted in the test code, and matching elements highlighted in the browser window.



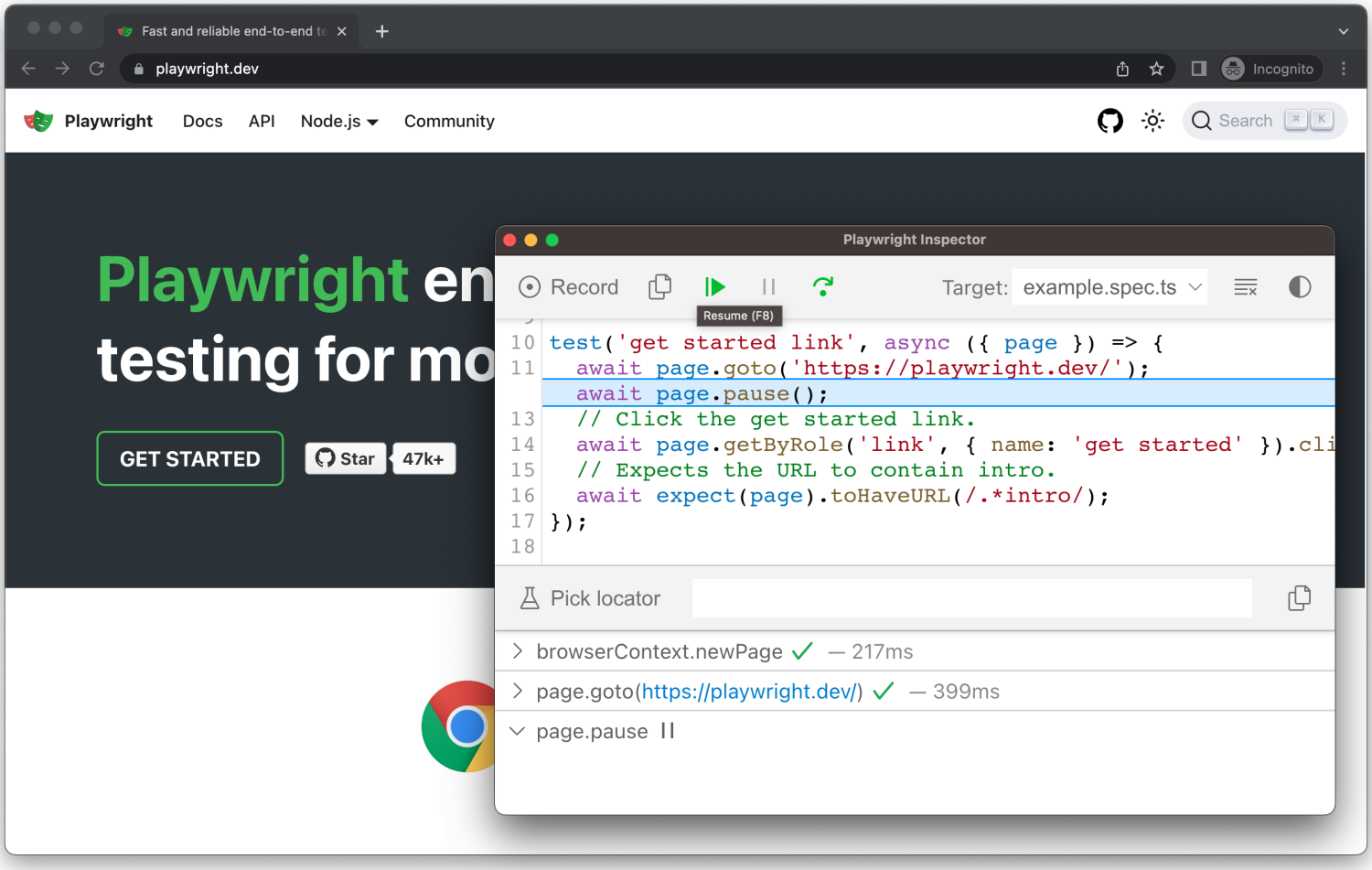
Run a test from a specific breakpoint

To speed up the debugging process you can add a page.pause() method to your test. This way you won't have to step through each action of your test to get to the point where you want to debug.

Sync Async

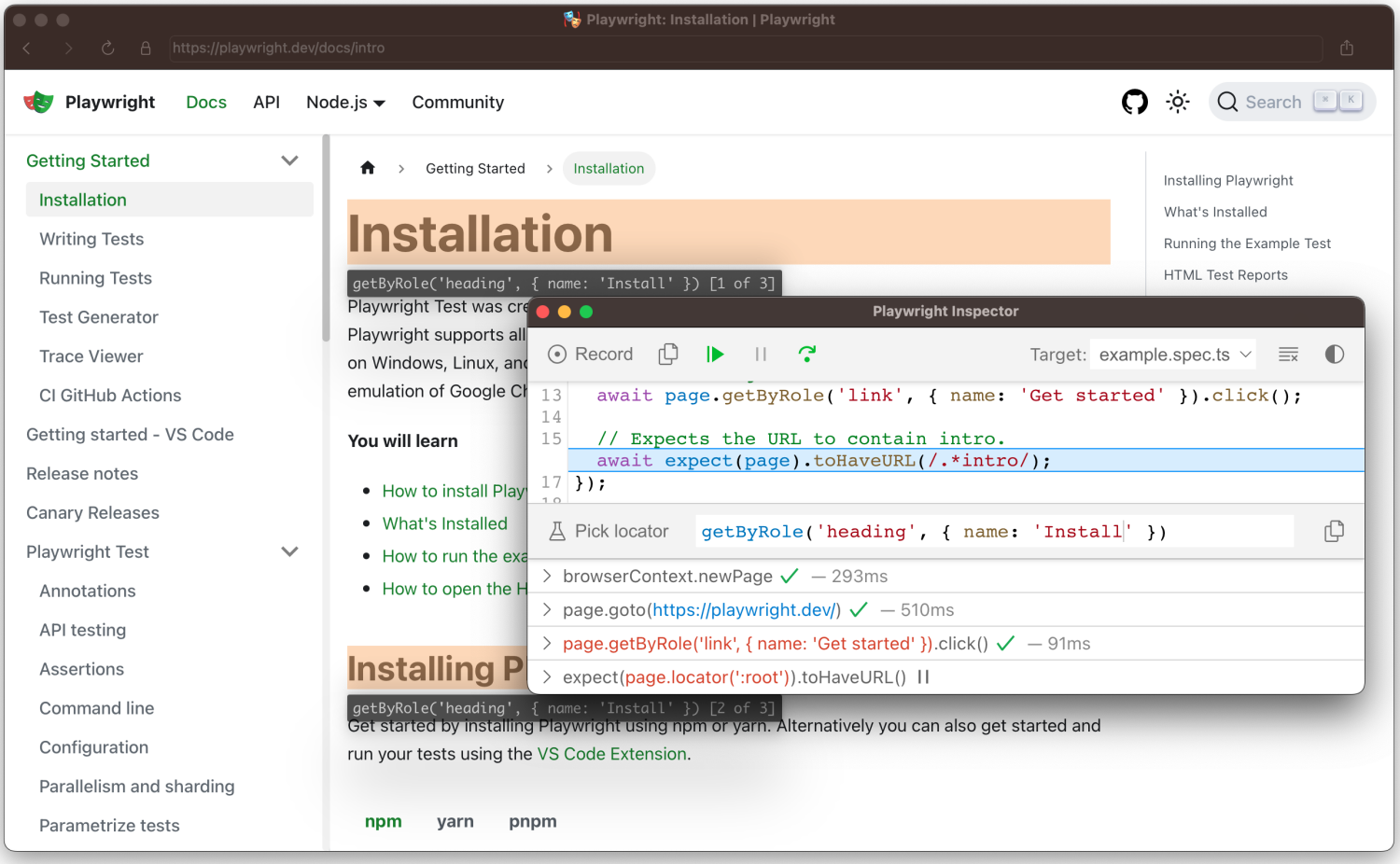
```
page.pause()
```

Once you add a `page.pause()` call, run your tests in debug mode. Clicking the "Resume" button in the Inspector will run the test and only stop on the `page.pause()`.



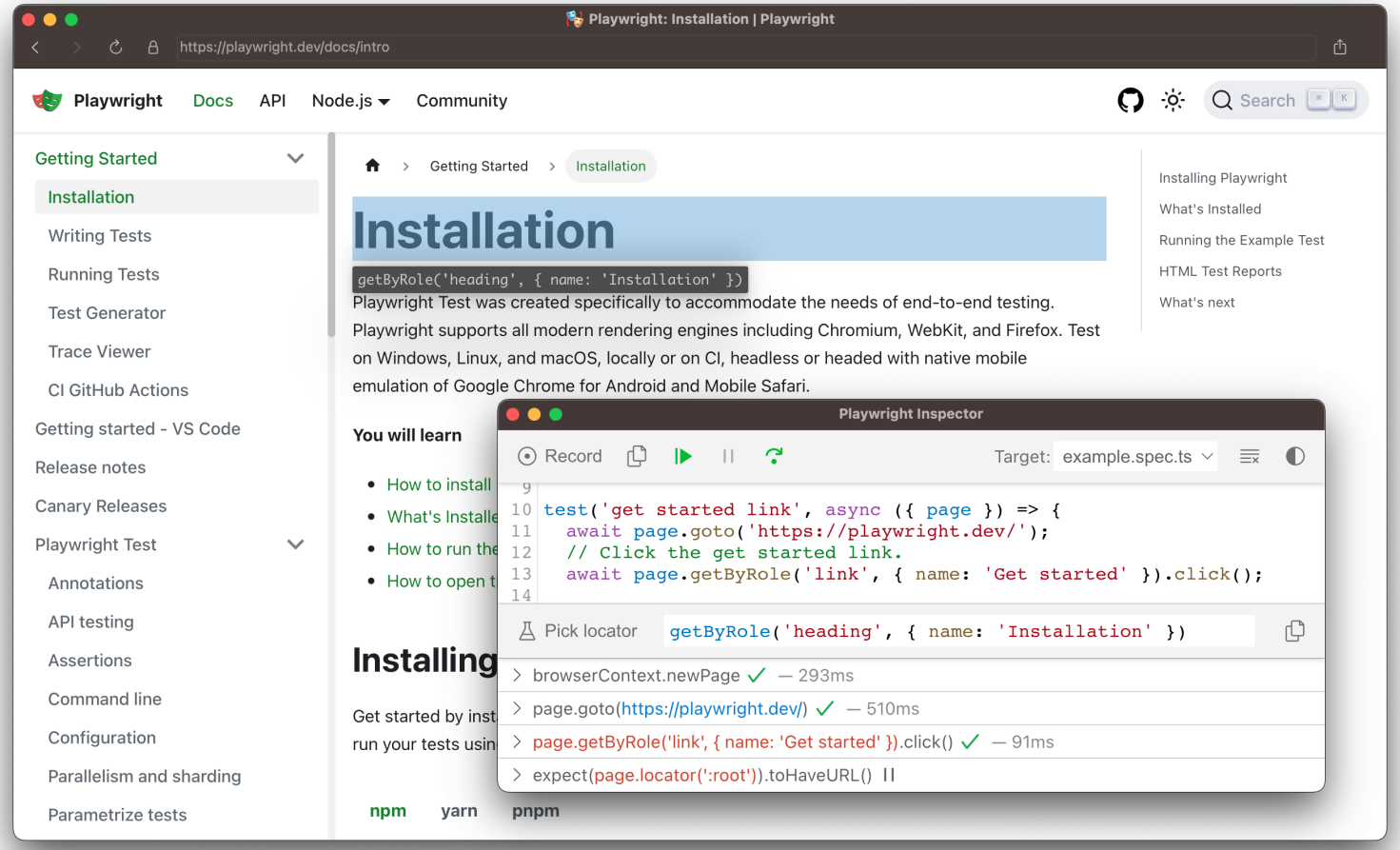
Live editing locators

While running in debug mode you can live edit the locators. Next to the 'Pick Locator' button there is a field showing the locator that the test is paused on. You can edit this locator directly in the **Pick Locator** field, and matching elements will be highlighted in the browser window.



Picking locators

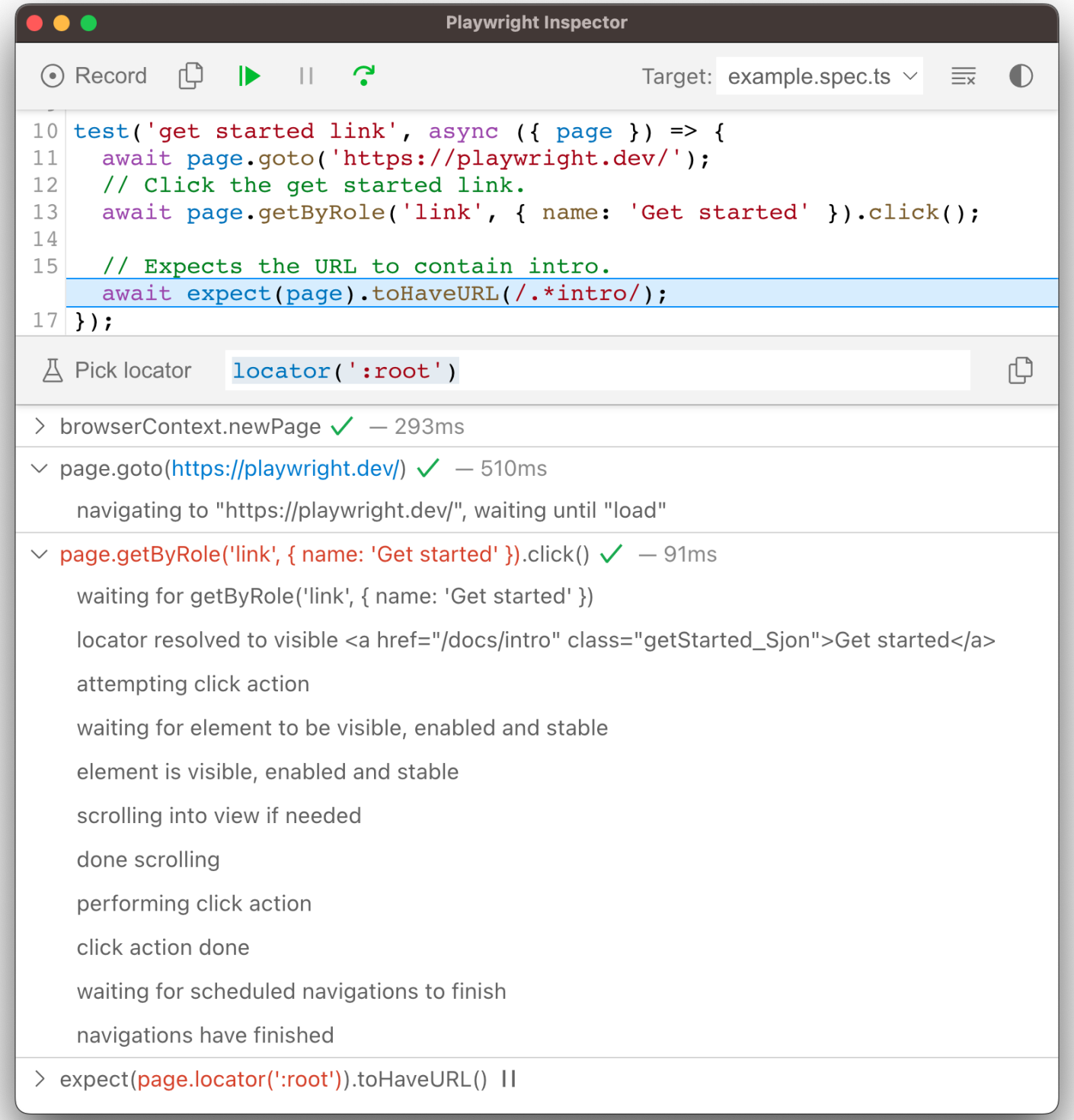
While debugging you might need to choose a more resilient locator. You can do this by clicking on the **Pick Locator** button and hovering over any element in the browser window. While hovering over an element you will see the code needed to locate this element highlighted below. Clicking an element in the browser will add the locator into the field where you can then either tweak it or copy it into your code.



Playwright will look at your page and figure out the best locator, prioritizing role, text and test id locators. If Playwright finds multiple elements matching the locator, it will improve the locator to make it resilient and uniquely identify the target element, so you don't have to worry about failing tests due to locators.

Actionability logs

By the time Playwright has paused on a click action, it has already performed actionability checks that can be found in the log. This can help you understand what happened during your test and what Playwright did or tried to do. The log tells you if the element was visible, enabled and stable, if the locator resolved to an element, scrolled into view, and so much more. If actionability can't be reached, it will show the action as pending.



Trace Viewer

Playwright Trace Viewer is a GUI tool that lets you explore recorded Playwright traces of your tests. You can go back and forward through each action on the left side, and visually see what was happening during the action. In the middle of the screen, you can see a DOM snapshot for

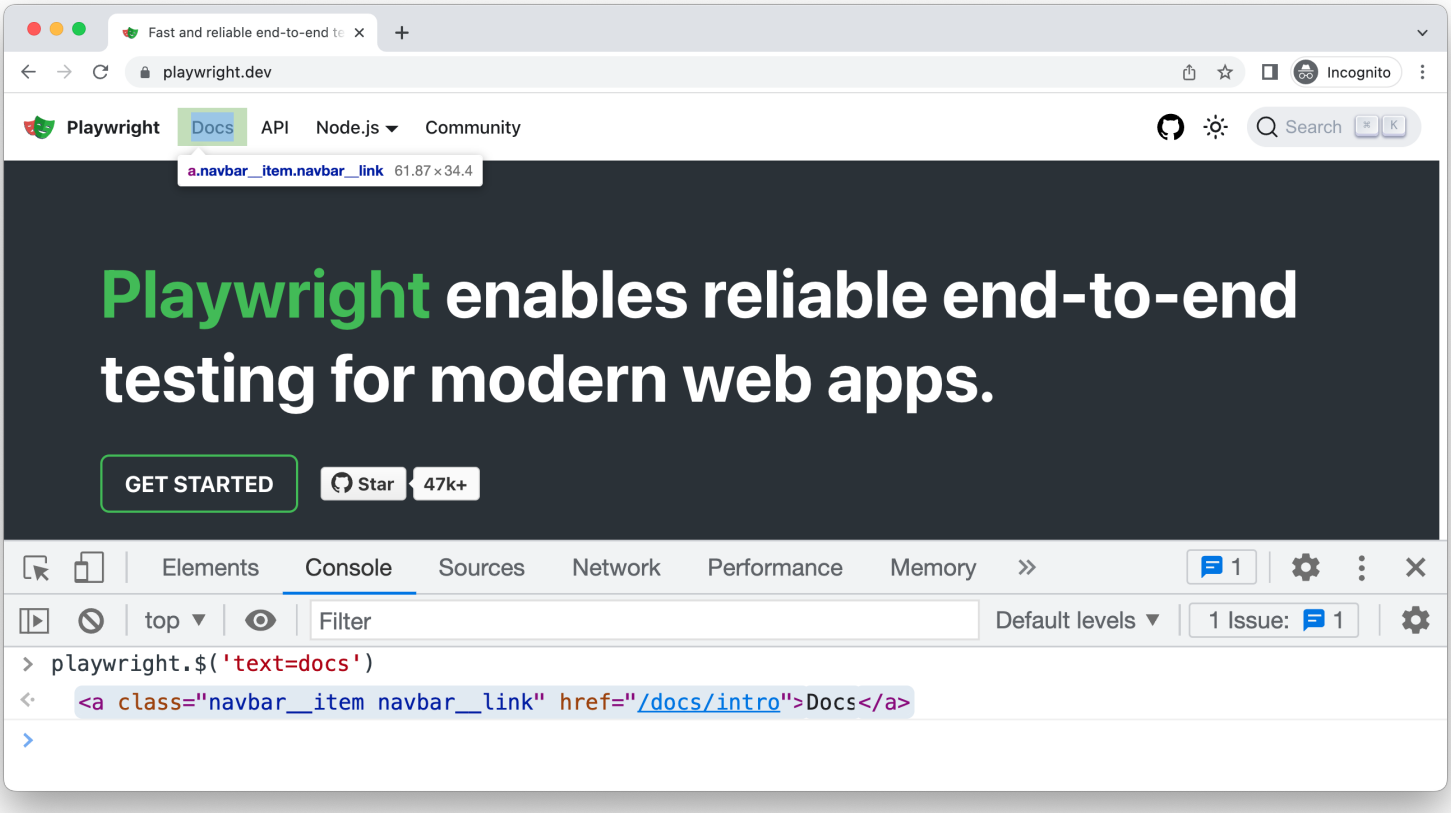
the action. On the right side you can see action details, such as time, parameters, return value and log. You can also explore console messages, network requests and the source code.

To learn more about how to record traces and use the Trace Viewer, check out the Trace Viewer guide.

Browser Developer Tools

When running in Debug Mode with `PWDEBUG=console`, a `playwright` object is available in the Developer tools console. Developer tools can help you to:

- Inspect the DOM tree and **find element selectors**
- **See console logs** during execution (or learn how to read logs via API)
- Check **network activity** and other developer tools features



To debug your tests using the browser developer tools start by setting a breakpoint in your test to pause the execution using the page.pause() method.

Sync Async

```
page.pause()
```

Once you have set a breakpoint in your test you can then run your test with `PWDEBUG=console`.

Bash PowerShell Batch

```
PWDEBUG=console pytest -s
```

Once Playwright launches the browser window you can open the developer tools. The `playwright` object will be available in the console panel.

playwright.$(selector)

Query the Playwright selector, using the actual Playwright query engine, for example:

```
playwright.$('.auth-form >> text=Log in');

<button>Log in</button>
```

playwright.$$(selector)

Same as `playwright.$`, but returns all matching elements.

```
playwright.$$('li >> text=John')

[<li>, <li>, <li>, <li>]
```

playwright.inspect(selector)

Reveal element in the Elements panel.

```
playwright.inspect('text=Log in')
```

playwright.locator(selector)

Create a locator and query matching elements, for example:

```
playwright.locator('.auth-form', { hasText: 'Log in' });

Locator ()
  - element: button
  - elements: [button]
```

playwright.selector(element)

Generates selector for the given element. For example, select an element in the Elements panel and pass `$0`:

```
playwright.selector($0)

"div[id="glow-ingress-block"] >> text=/.*Hello.*/"
```

Verbose API logs

Playwright supports verbose logging with the `DEBUG` environment variable.

Bash | PowerShell | Batch

```
DEBUG=pw:api pytest -s
```

> **NOTE**
>
> **For WebKit**: launching WebKit Inspector during the execution will prevent the Playwright script from executing any further and will reset pre-configured user agent and device emulation.

Headed mode

Playwright runs browsers in headless mode by default. To change this behavior, use `headless: false` as a launch option.

You can also use the `slow_mo` option to slow down execution (by N milliseconds per operation) and follow along while debugging.

Sync | Async

```
# Chromium, Firefox, or WebKit
chromium.launch(headless=False, slow_mo=100)
```