COMPONENT TESTS

 YOU WILL LEARN

- why component testing
- Cypress native component testing
- Playwright "bridge" component testing

COMPONENT TESTING

- https://on.cypress.io/component
  - https://glebbahmutov.com/blog/how-cypress-component-testing-was-born/
- https://playwright.dev/docs/test-components

PLAYWRIGHT STATUS

- @playwright/experimental-ct-react
- @playwright/experimental-ct-svelte
- @playwright/experimental-ct-vue

Framework	UI Library	Bundler
React with Vite	React 18-19	Vite 4-6
React with Webpack	React 18-19	Webpack 4-5
Next.js 14-15	React 18-19	Webpack 5
Vue with Vite	Vue 3	Vite 4-6
Vue with Webpack	Vue 3	Webpack 4-5
Angular	Angular 17-19	Webpack 5
Svelte with Vite Alpha	Svelte 5	Vite 4-6
Svelte with Webpack Alpha	Svelte 5	Webpack 4-5

The following integrations are built and maintained by Cypress community members.

Framework	UI Library	Bundler
Qwik Community	Qwik	Vite

Cypress v14 Component

BUTTON COMPONENT

```
const Button = ({ customClass, label, onClick }) => {
  return (
    <button
      className={`btn${buttonTypeClass}${buttonSize}${extraClass}`}
      onClick={onClick}
    >
      ...
    </button>
  )
}
```

How would you test all possible behaviors of this button?

```
const Button = ({ customClass, label, onClick }) => {
```

Testing such button through E2E is hard, since the pages might only use a limited number of button's features.

Component tests to the rescue!

COMPONENT TESTS

- small tests mounting a single front-end component (React, Vue, Angular, Svelte, etc)
- interacting with the "live" component
- checking the expected results:
  - DOM changes
  - network calls
  - prop callbacks

E2E VS COMPONENT TESTS

- slow vs fast
- user story vs component
- web app vs component

EXAMPLE

- clone repo `https://github.com/bahmutov/taste-the-sauce-vite`
- check out branch `g1`
- `npm install`
- `npx playwright install`

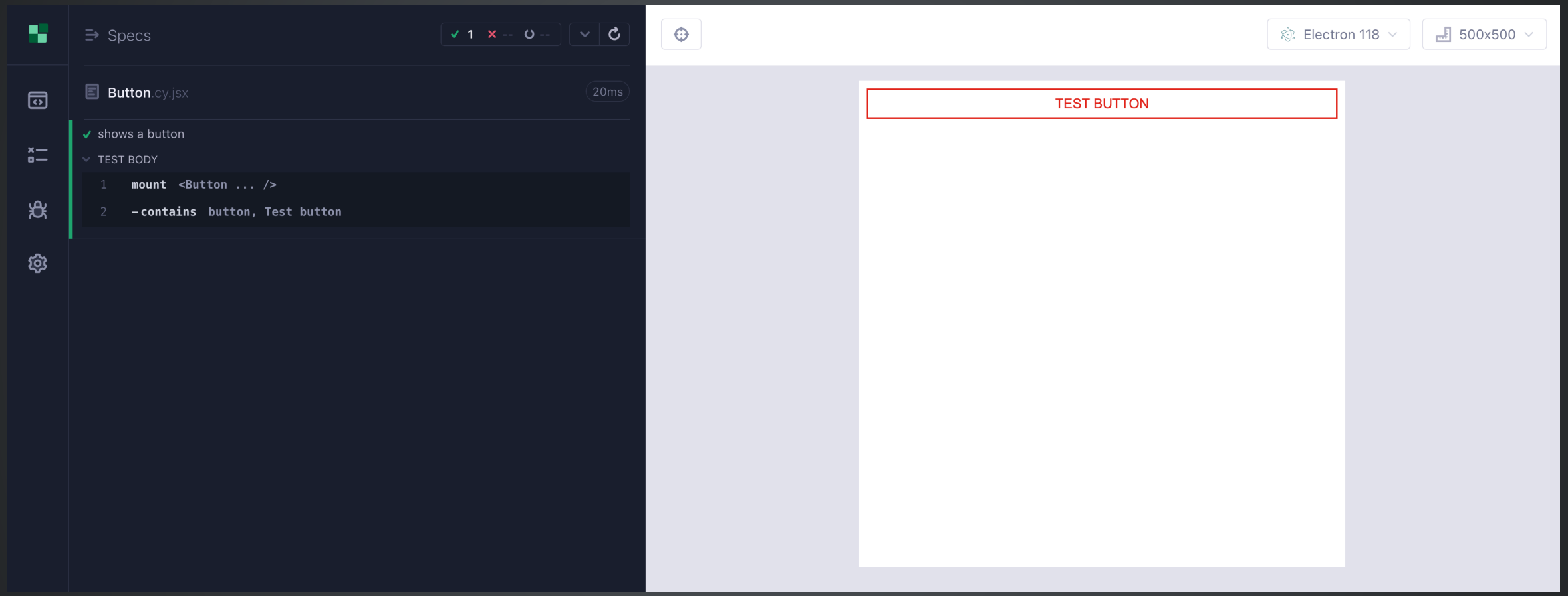
TODO: WRITE CYPRESS COMPONENT TEST

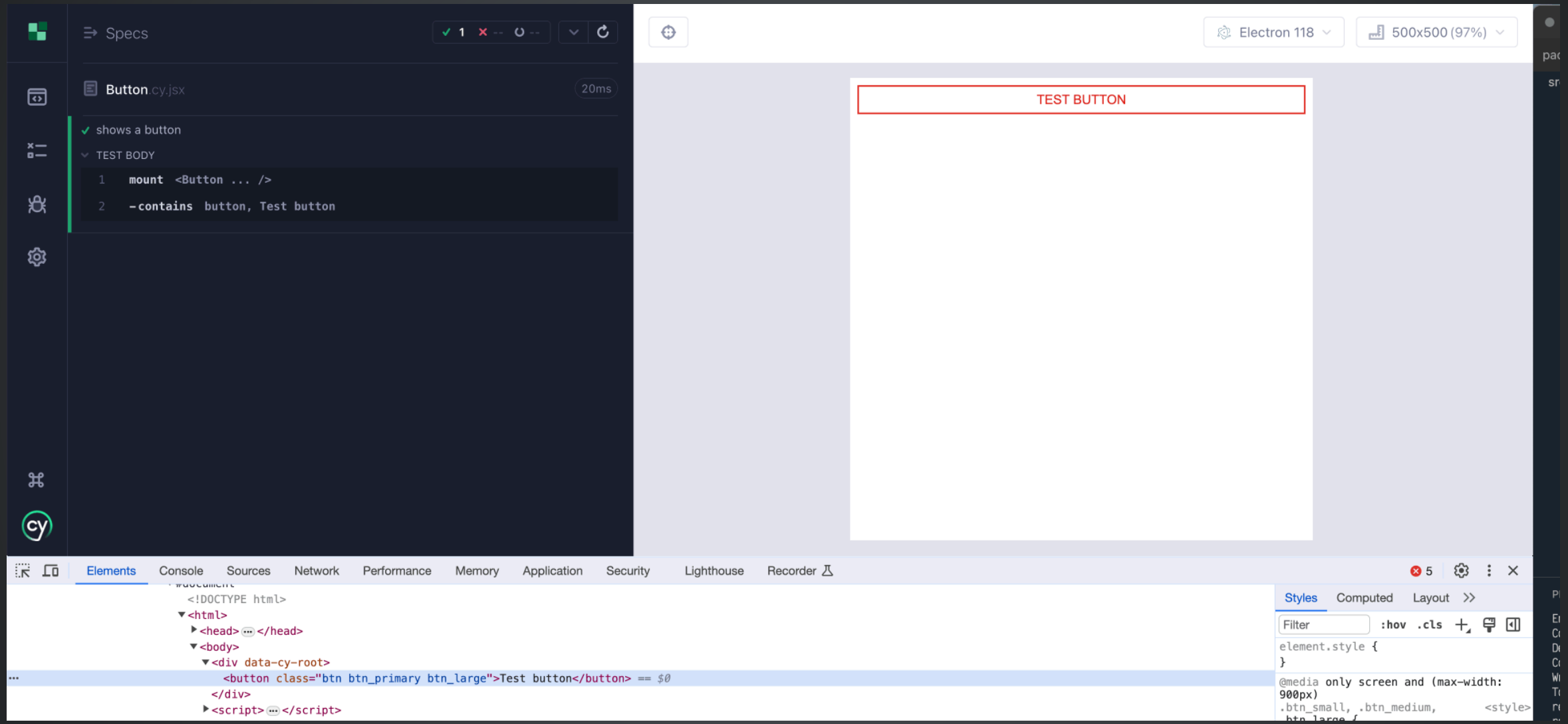
```
// src/components/Button.cy.jsx

import React from 'react'
import Button from './Button'

it('shows a button', () => {
  // use the cy.mount command to mount the Button component
  // with the prop `label` set to 'Test button'
  // confirm the page contains a button with the text 'Test button'
})
```

Tip: run Cypress in the component mode `npx cypress open --component`

Tip 2: look at the component support file `cypress/support/component.jsx`

```jsx
// src/components/Button.cy.jsx

import React from 'react'
import Button from './Button'

it('shows a button', () => {
  // use the cy.mount command to mount the Button component
  // with the prop `label` set to 'Test button'
  cy.mount(<Button label="Test button" />)
  // confirm the page contains a button with the text 'Test button'
  cy.contains('button', 'Test button')
})
```


Specs
1
Button.cy.jsx
20ms
shows a button
TEST BODY
1 mount <Button ... />
2 -contains button, Test button
Electron 118
500x500
TEST BUTTON


Specs
1
Button.cy.jsx
20ms
shows a button
TEST BODY
1 mount <Button ... />
2 -contains button, Test button
Electron 118
500x500 (97%)
TEST BUTTON
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
5
<!DOCTYPE html>
<html>
<head>…</head>
<body>
<div data-cy-root>
<button class="btn btn_primary btn_large">Test button</button> == $0

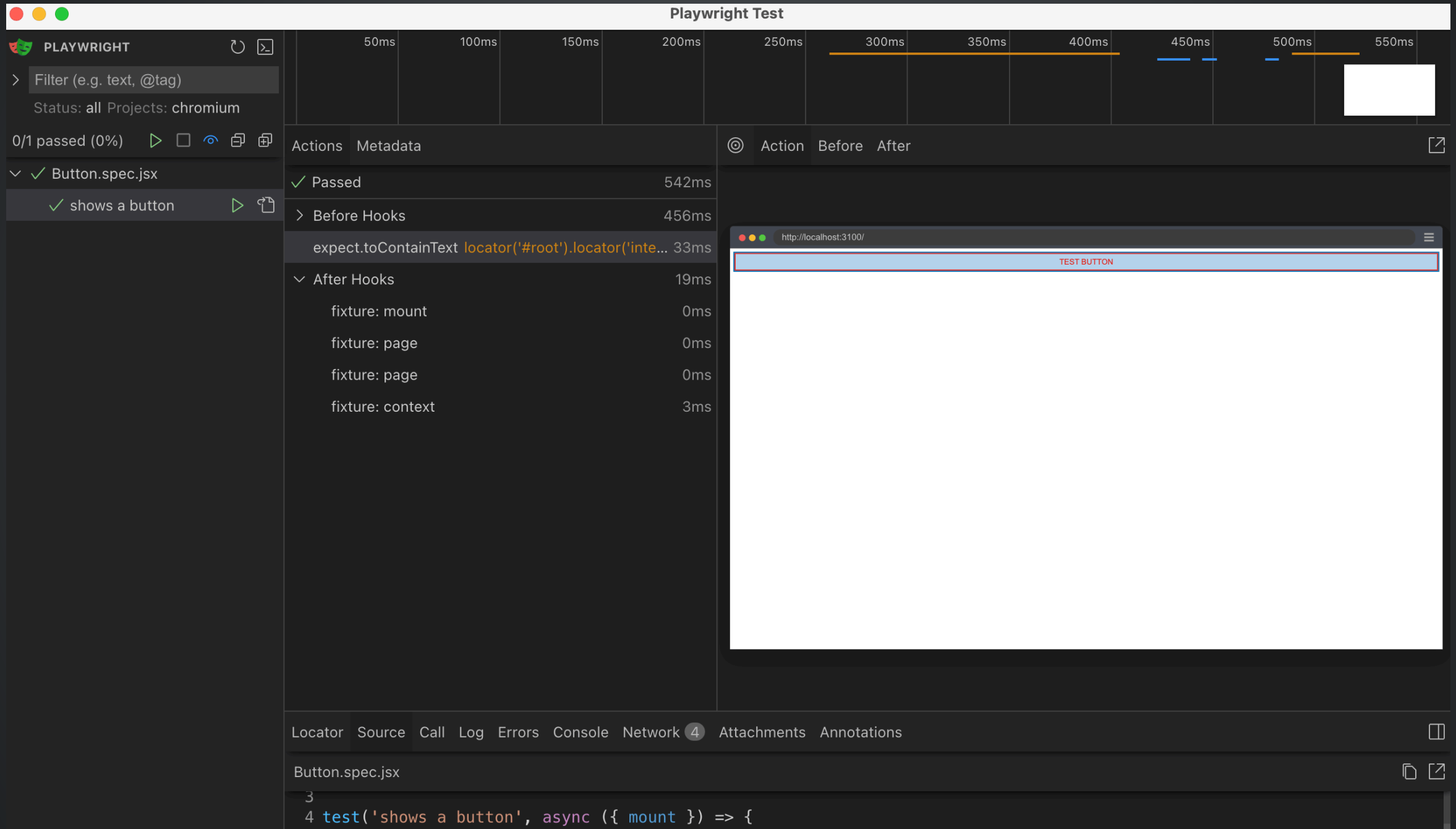
<script>…</script>
Styles
Computed
Layout
Filter
:hov .cls
element.style {
}
@media only screen and (max-width: 900px)
.btn_small, .btn_medium,
<style>

TODO: WRITE PLAYWRIGHT COMPONENT TEST

```
// src/components/Button.spec.jsx

import { test, expect } from '@playwright/experimental-ct-react17'
import Button from './Button'

test('shows a button', async ({ mount }) => {
  // use the mount command to mount the Button component
  // with the prop `label` set to 'Test button'
  // confirm the component contains text 'Test button'
})
```

Tip: run the component spec in UI mode with `npm run test-ct -- --ui`. **Important** ⚠️: the `ctPort` in the Pw config must point at an unused port.

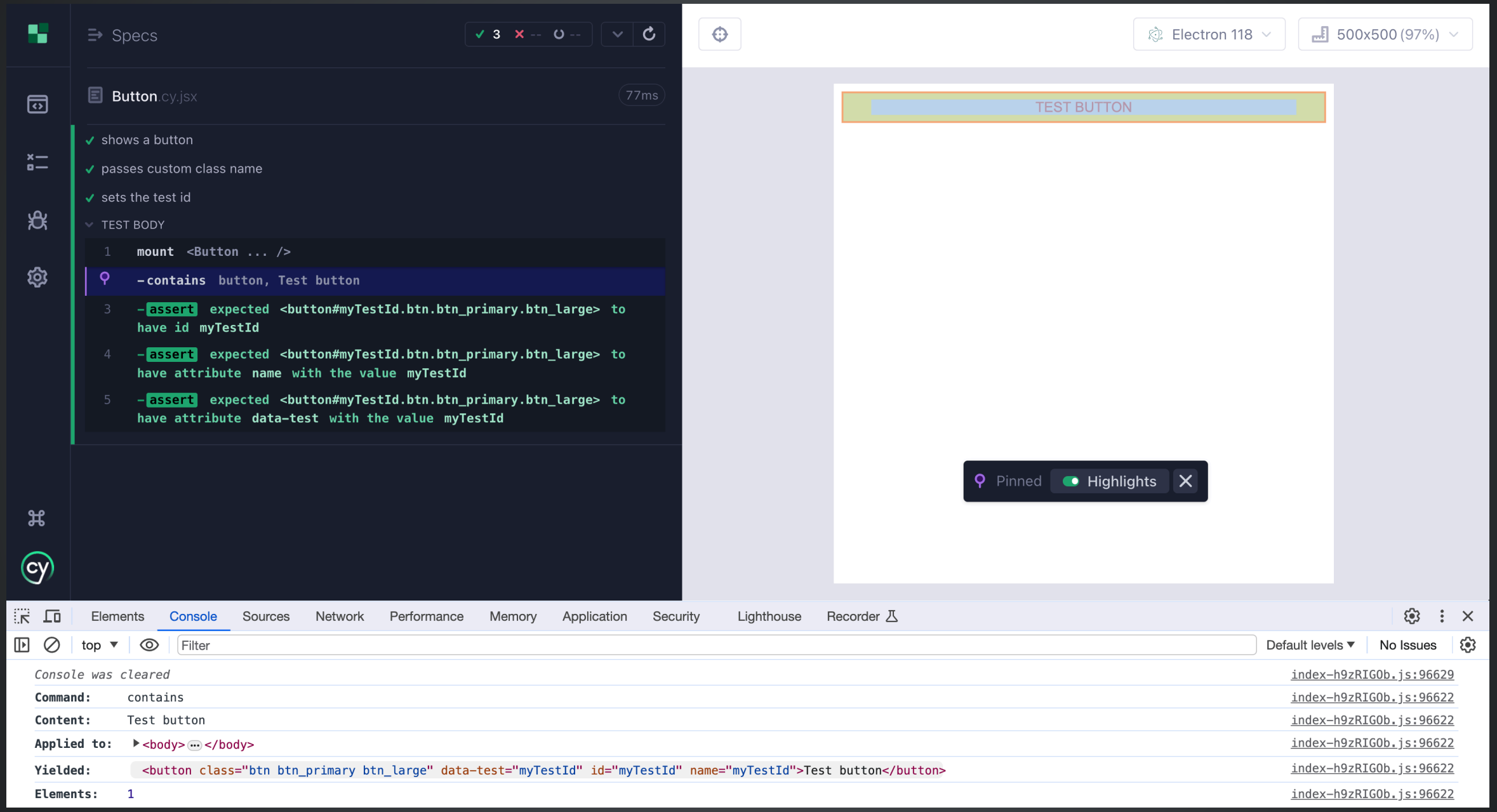
```jsx
// src/components/Button.spec.jsx

import { test, expect } from '@playwright/experimental-ct-react17'
import Button from './Button'

test('shows a button', async ({ mount }) => {
  // use the mount command to mount the Button component
  // with the prop `label` set to 'Test button'
  const component = await mount(<Button label="Test button" />)
  // confirm the component contains text 'Test button'
  await expect(component).toContainText('Test button')
})
```


Playwright Test
PLAYWRIGHT
Filter (e.g. text, @tag)
Status: all Projects: chromium
0/1 passed (0%)
Button.spec.jsx
shows a button
50ms
100ms
150ms
200ms
250ms
300ms
350ms
400ms
450ms
500ms
550ms
Actions
Metadata
Passed 542ms
Before Hooks 456ms
expect.toContainText locator('#root').locator('inte... 33ms
After Hooks 19ms
fixture: mount 0ms
fixture: page 0ms
fixture: page 0ms
fixture: context 3ms
Action
Before
After
http://localhost:3100/
TEST BUTTON
Locator
Source
Call
Log
Errors
Console
Network 4
Attachments
Annotations
Button.spec.jsx
3
4 test('shows a button', async ({ mount }) => {

TODO: TEST OTHER PROPS

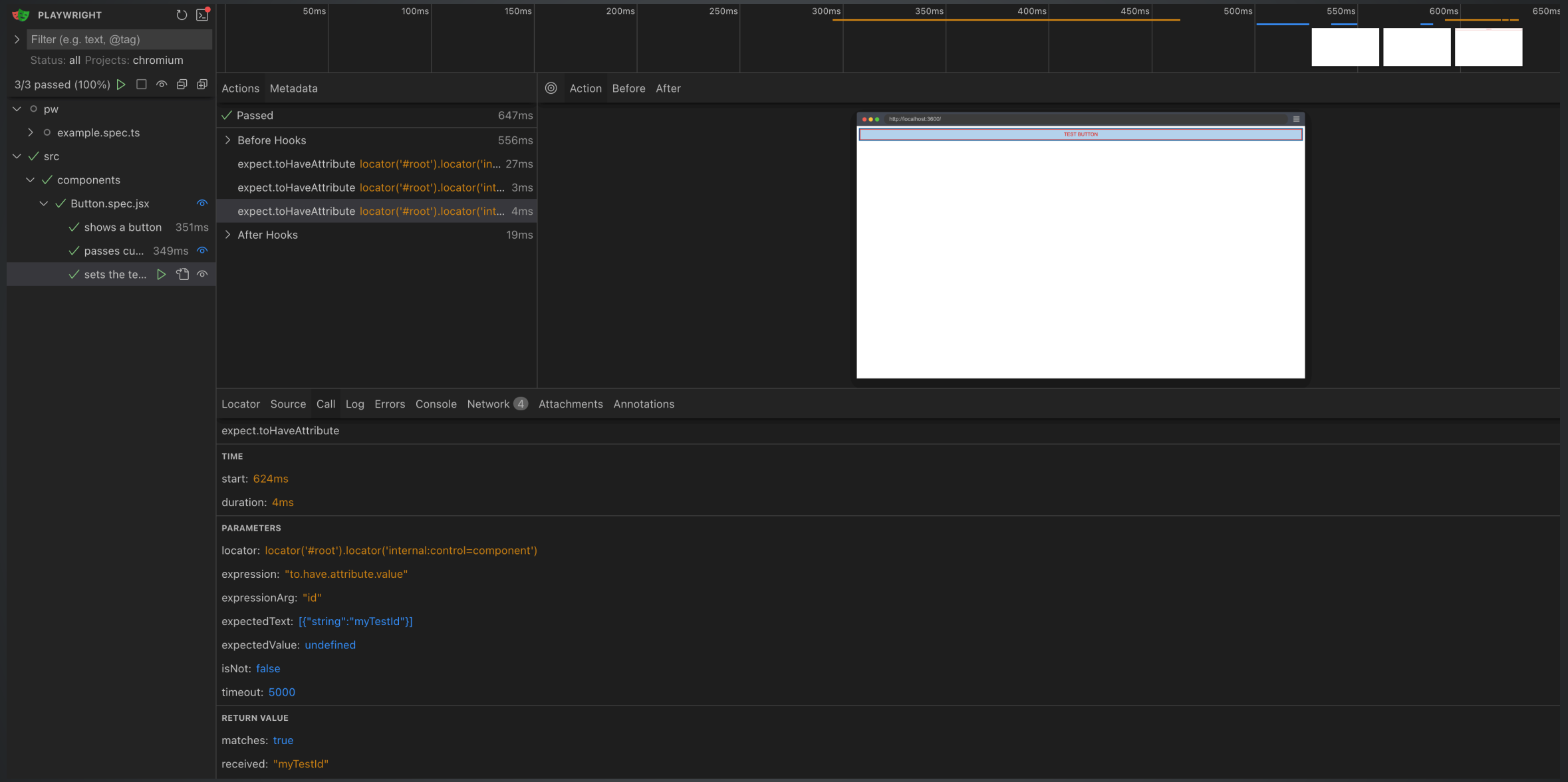
```
// src/components/Button.cy.jsx

it('passes custom class name', () => {
  // mount the Button with the customClass prop set to "myClass"
  // confirm the page contains a button with the class "myClass"
})

it('sets the test id', () => {
  // mount the Button with the testId prop set to "myTestId"
  // confirm the button with the text "Test button" has
  // the data-test, name, and id set to "myTestId"
})
```

```
// src/components/Button.cy.jsx
it('passes custom class name', () => {
  cy.mount(<Button label="Test button" customClass="myClass" />)
  cy.get('button.myClass')
})

it('sets the test id', () => {
  cy.mount(<Button label="Test button" testId="myTestId" />)
  cy.contains('button', 'Test button')
    .should('have.id', 'myTestId')
    .and('have.attr', 'name', 'myTestId')
    .and('have.attr', 'data-test', 'myTestId')
})
```


Specs
3
Button.cy.jsx
77ms
shows a button
passes custom class name
sets the test id
TEST BODY
1 mount <Button ... />
-contains button, Test button
3 -assert expected <button#myTestId.btn.btn_primary.btn_large> to have id myTestId
4 -assert expected <button#myTestId.btn.btn_primary.btn_large> to have attribute name with the value myTestId
5 -assert expected <button#myTestId.btn.btn_primary.btn_large> to have attribute data-test with the value myTestId
Electron 118
500x500 (97%)
TEST BUTTON
Pinned
Highlights
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
top
Filter
Default levels
No Issues
Console was cleared
index-h9zRIGOb.js:96629
Command: contains
index-h9zRIGOb.js:96622
Content: Test button
index-h9zRIGOb.js:96622
Applied to: <body> ... </body>
index-h9zRIGOb.js:96622
Yielded: <button class="btn btn_primary btn_large" data-test="myTestId" id="myTestId" name="myTestId">Test button</button>
index-h9zRIGOb.js:96622
Elements: 1
index-h9zRIGOb.js:96622

```
// src/components/Button.spec.jsx

test('passes custom class name', async ({ mount }) => {
  // mount the Button with the customClass prop set to "myClass"
  // confirm the page contains a button with the class "myClass"
})

test('sets the test id', async ({ mount }) => {
  // mount the Button with the testId prop set to "myTestId"
  // confirm the button with the text "Test button" has
  // the data-test, name, and id set to "myTestId"
})
```

```jsx
// src/components/Button.spec.jsx

test('passes custom class name', async ({ mount }) => {
  const component = await mount(
    <Button label="Test button" customClass="myClass" />
  )
  await expect(component).toHaveClass(/myClass/)
})

test('sets the test id', async ({ mount }) => {
  const component = await mount(
    <Button label="Test button" testId="myTestId" />
  )
  await expect(component).toHaveAttribute('data-test', 'myTestId')
  await expect(component).toHaveAttribute('name', 'myTestId')
  await expect(component).toHaveAttribute('id', 'myTestId')
```


PLAYWRIGHT
Filter (e.g. text, @tag)
Status: all Projects: chromium
3/3 passed (100%)
pw
example.spec.ts
src
components
Button.spec.jsx
shows a button 351ms
passes cu... 349ms
sets the te...
50ms 100ms 150ms 200ms 250ms 300ms 350ms 400ms 450ms 500ms 550ms 600ms 650ms
Actions Metadata
Passed 647ms
Before Hooks 556ms
expect.toHaveAttribute locator('#root').locator('in... 27ms
expect.toHaveAttribute locator('#root').locator('int... 3ms
expect.toHaveAttribute locator('#root').locator('int... 4ms
After Hooks 19ms
Action Before After
http://localhost:3600/
TEST BUTTON
Locator Source Call Log Errors Console Network 4 Attachments Annotations
expect.toHaveAttribute
TIME
start: 624ms
duration: 4ms
PARAMETERS
locator: locator('#root').locator('internal:control=component')
expression: "to.have.attribute.value"
expressionArg: "id"
expectedText: [{"string":"myTestId"}]
expectedValue: undefined
isNot: false
timeout: 5000
RETURN VALUE
matches: true
received: "myTestId"

CONFIG DIFFERENCES FOR COMPONENT TESTING

Cypress has a single config for both E2E and component testing

```
// cypress.config.js
import { defineConfig } from 'cypress'
export default defineConfig({
  e2e: {},
  component: {}
})
```

Playwright has separate configs and test syntax

```
// Playwright e2e config file
import { defineConfig, devices } from '@playwright/test'
// Playwright component config file
import { defineConfig, devices } from '@playwright/experimental-ct-react17'
```

Playwright provides different `test` and `expect` for component testing

```
// Playwright E2E specs
import { test, expect } from '@playwright/test'
// Playwright component spec
import { test, expect } from '@playwright/experimental-ct-react17'
```

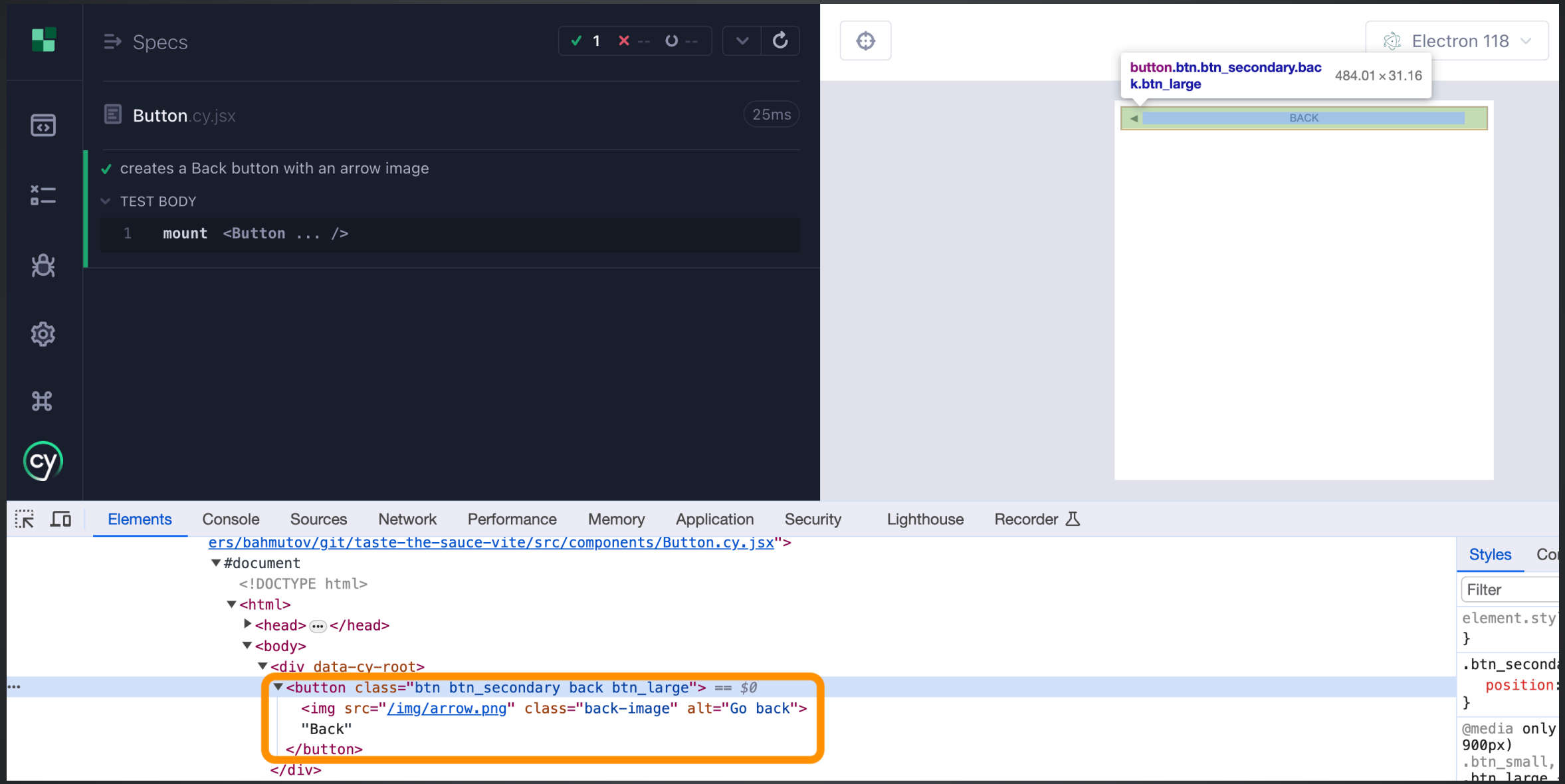
USE CONSTANTS

- `git checkout g2`
- `npm install && npx playwright install`

Finish this Cypress component test

```
// src/components/Button.cy.jsx
it('creates a Back button with an arrow image', () => {
  // mount the Button with the type prop set to "back"
  //
  // confirm that inside the button element with class "btn"
  // there is an image with alt text "Go back"
  // and the image loads its source without errors
})
```

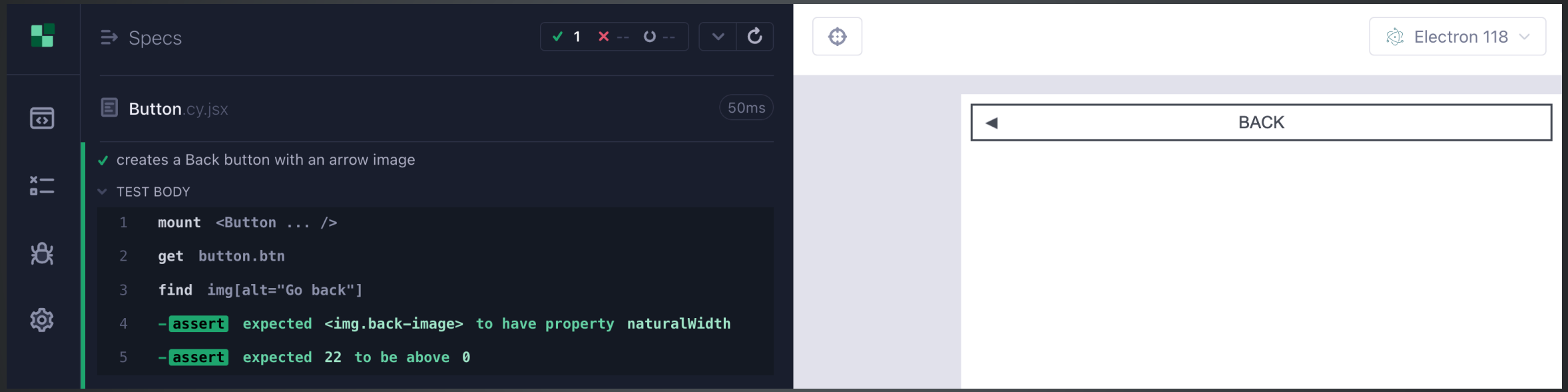
Tip: look at the image element's properties in the DevTools to check if it successfully loads (network, decoding, rendering)


Specs
1
Button.cy.jsx
25ms
creates a Back button with an arrow image
TEST BODY
1 mount <Button ... />
Electron 118
button.btn.btn_secondary.back.btn_large
484.01 × 31.16
BACK
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
ers/bahmutov/git/taste-the-sauce-vite/src/components/Button.cy.jsx">
#document
<!DOCTYPE html>
<html>
<head> ... </head>
<body>
<div data-cy-root>
<button class="btn btn_secondary back btn_large"> == $0
<img src="/img/arrow.png" class="back-image" alt="Go back">
"Back"
</button>

Styles
Filter
element.sty
}
.btn_second
position:
}
@media only
900px)
.btn_small,

CYPRESS COMPONENT TEST

```
import Button from './Button'
import { BUTTON_TYPES } from './Button'
it('creates a Back button with an arrow image', () => {
  // mount the Button with the type prop set to "back"
  cy.mount(<Button label="Back" type={BUTTON_TYPES.BACK} />)
  // confirm that inside the button element with class "btn"
  // there is an image with alt text "Go back"
  // and the image loads its source without errors
  cy.get('button.btn')
    .find('img[alt="Go back"]')
    .should('have.prop', 'naturalWidth')
    .should('be.greaterThan', 0)
})
```

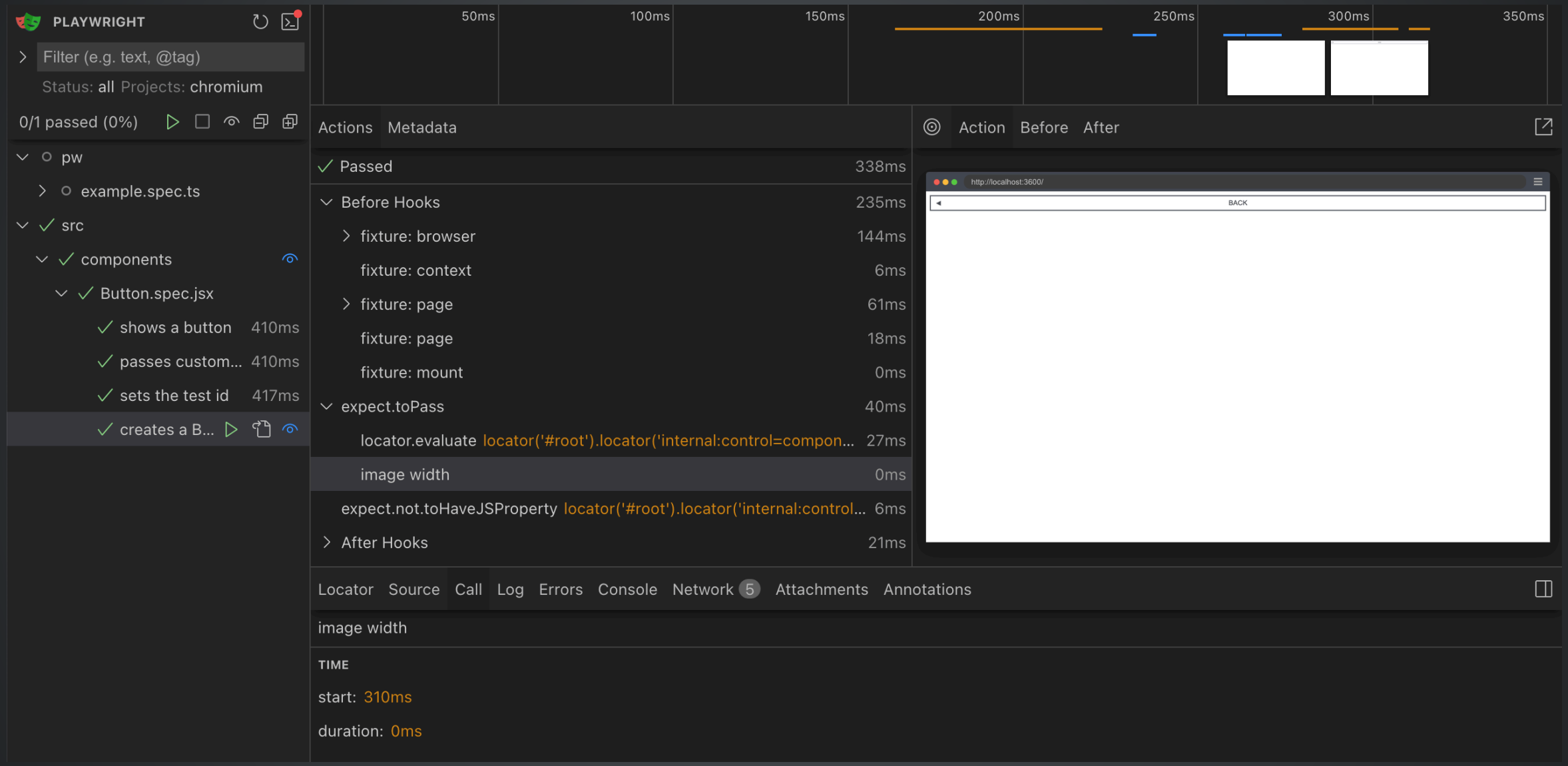

Specs
1
--
--
Button.cy.jsx
50ms
creates a Back button with an arrow image
TEST BODY
1 mount <Button ... />
2 get button.btn
3 find img[alt="Go back"]
4 -assert expected <img.back-image> to have property naturalWidth
5 -assert expected 22 to be above 0
Electron 118
BACK

USE CONSTANTS

Finish this Playwright component test

```
// src/components/Button.spec.jsx
import { test, expect } from '@playwright/experimental-ct-react17'
import Button from './Button'
test('creates a Back button with an arrow image', async ({ mount }) => {
  // mount the Button with the type prop set to "back"
  // confirm that inside the button element with class "btn"
  // there is an image with alt text "Go back"
  // and the image loads its source without errors
})
```

```
test('creates a Back button with an arrow image', async ({ mount }) => {
  const component = await mount(
    <Button label="Back" type={BUTTON_TYPES.BACK} />
  )
  const image = component.locator('img[alt="Go back"]')
  await expect(async () => {
    const width = await image.evaluate((node) => node.naturalWidth)
    expect(width, 'image width').toBeGreaterThan(0)
  }).toPass()
  // this solution could also work in this case
  await expect(image).not.toHaveJSProperty('naturalWidth', 0)
})
```


PLAYWRIGHT
Filter (e.g. text, @tag)
Status: all Projects: chromium
0/1 passed (0%)
pw
example.spec.ts
src
components
Button.spec.jsx
shows a button 410ms
passes custom... 410ms
sets the test id 417ms
creates a B...
50ms
100ms
150ms
200ms
250ms
300ms
350ms
Actions
Metadata
Passed 338ms
Before Hooks 235ms
fixture: browser 144ms
fixture: context 6ms
fixture: page 61ms
fixture: page 18ms
fixture: mount 0ms
expect.toPass 40ms
locator.evaluate locator('#root').locator('internal:control=compon... 27ms
image width 0ms
expect.not.toHaveJSProperty locator('#root').locator('internal:control... 6ms
After Hooks 21ms
Action
Before
After
http://localhost:3600/
BACK
Locator
Source
Call
Log
Errors
Console
Network 5
Attachments
Annotations
image width
TIME
start: 310ms
duration: 0ms

QUESTIONS:

- what do you see when you run Cypress component test?
- what do you see when you run Playwright component test?

PASSING FUNCTIONAL PROPS

```
const Button = ({
  customClass,
  label,
  onClick,
}) => {
  return (
    <button
      className={`btn${buttonTypeClass}${buttonSize}${extraClass}`}
      onClick={onClick}
```

Let's test the `onClick` prop when the user clicks the button.

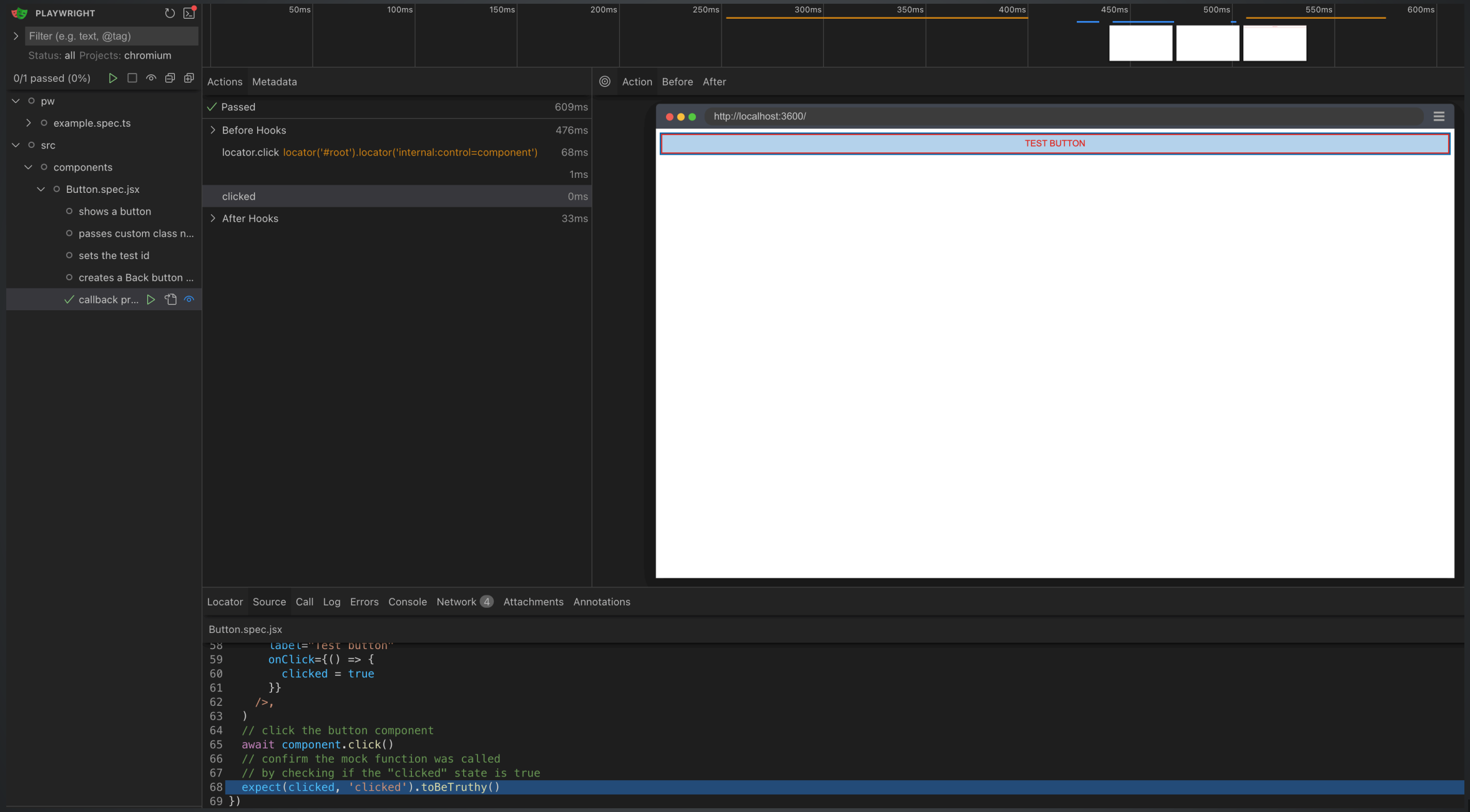
FINISH THE PLAYWRIGHT TEST

- branch g3
- `npm install && npx playwright install`

```
// src/components/Button.spec.jsx
test('callback prop is called on click', async ({ mount }) => {
  // keep track of the clicked state
  let clicked = false
  // mount the Button with the onClick prop set to a small function
  // that changes "clicked" to true
  //
  // click the button component
  // confirm the mock function was called
  // by checking if the "clicked" state is true
})
```

Question: what do you see during the test?

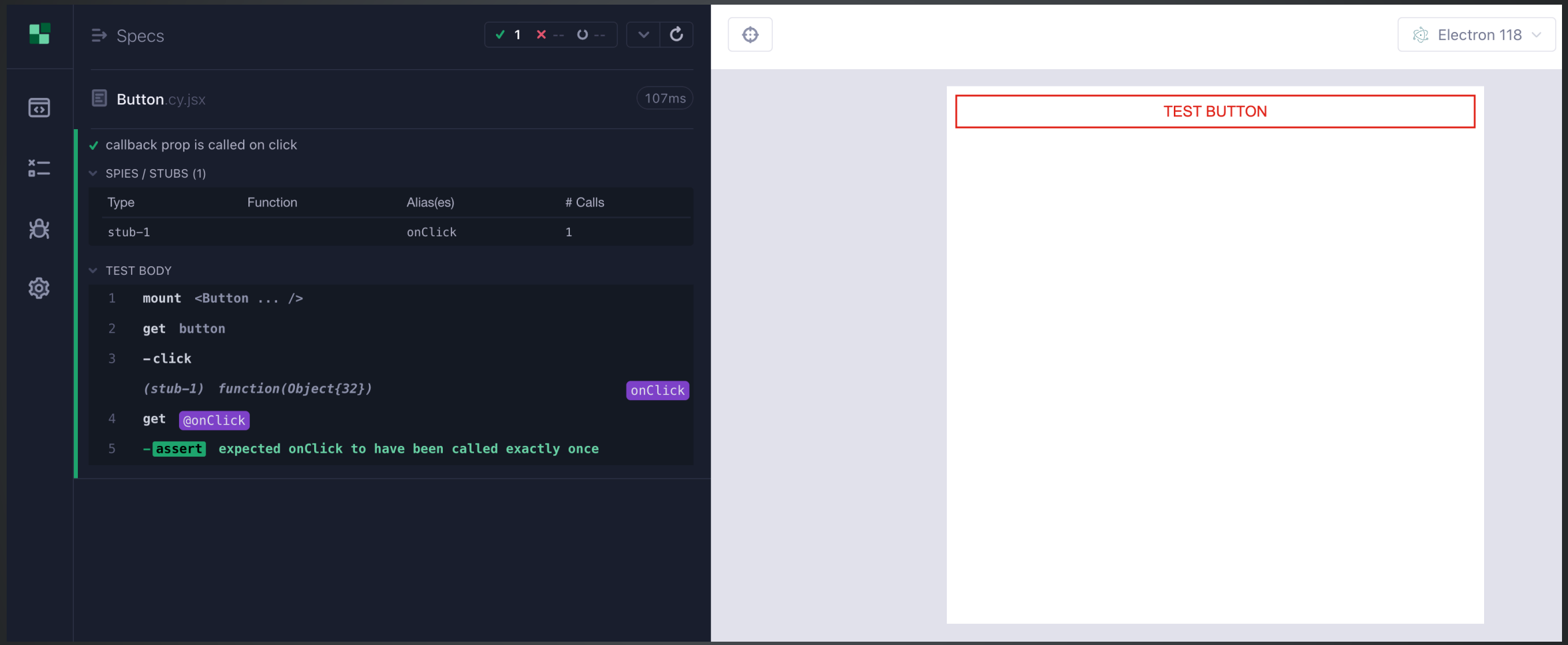
```
// src/components/Button.spec.jsx
test('callback prop is called on click', async ({ mount }) => {
  let clicked = false
  const component = await mount(
    <Button
      label="Test button"
      onClick={() => {
        clicked = true
      }}
    />
  )
  await component.click()
  expect(clicked, 'clicked').toBeTruthy()
})
```


PLAYWRIGHT
Filter (e.g. text, @tag)
Status: all Projects: chromium
0/1 passed (0%)
pw
example.spec.ts
src
components
Button.spec.jsx
shows a button
passes custom class n...
sets the test id
creates a Back button ...
callback pr...
50ms
100ms
150ms
200ms
250ms
300ms
350ms
400ms
450ms
500ms
550ms
600ms
Actions
Metadata
Passed 609ms
Before Hooks 476ms
locator.click locator('#root').locator('internal:control=component') 68ms
1ms
clicked 0ms
After Hooks 33ms
Action
Before
After
http://localhost:3600/
TEST BUTTON
Locator
Source
Call
Log
Errors
Console
Network 4
Attachments
Annotations
Button.spec.jsx
label="Test Button"
onClick={() => {
clicked = true
}}
/>,
)
// click the button component
await component.click()
// confirm the mock function was called
// by checking if the "clicked" state is true
expect(clicked, 'clicked').toBeTruthy()
})

FINISH THE CYPRESS TEST

```
// src/components/Button.cy.jsx
it('callback prop is called on click', () => {
  // mount the Button with the onClick function stub
  // https://on.cypress.io/stub
  // give the stub an alias "onClick"
  // https://on.cypress.io/as
  //
  // click the button component
  //
  // confirm the stub function was called
})
```

```
// src/components/Button.cy.jsx
it('callback prop is called on click', () => {
  cy.mount(<Button label="Test button" onClick={cy.stub().as('onClick')} />)
  cy.get('button').click()
  cy.get('@onClick').should('have.been.calledOnce')
})
```


Specs
1
--
--
Button.cy.jsx
107ms
callback prop is called on click
SPIES / STUBS (1)
Type
Function
Alias(es)
# Calls
stub-1
onClick
1
TEST BODY
1 mount <Button ... />
2 get button
3 -click
(stub-1) function(Object{32})
onClick
4 get @onClick
5 -assert expected onClick to have been called exactly once
Electron 118
TEST BUTTON

ASYNC FUNCTIONAL PROPS

What if the component calls the functional prop `onClick` after some delay?

```
// Instead of this
<button onClick={onClick}>
// We have an async call
onClick={() => setTimeout(onClick, 1000)}
```

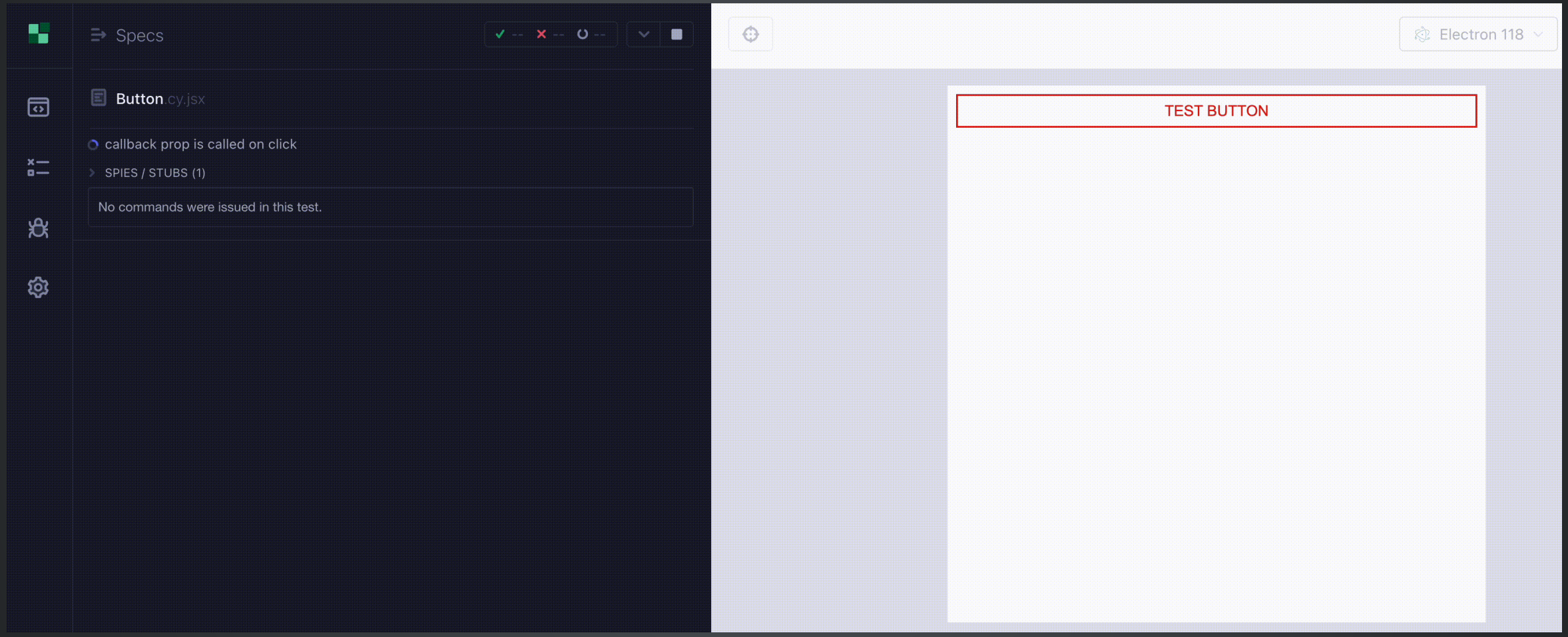
- check out branch `g4`
- `npm install`
- `npx playwright install`

Open test runners in the component testing modes; the `Button.jsx` component calls `onClick` after 1-second delay.

MODIFY CYPRESS TEST

```
// src/components/Button.cy.jsx
it('callback prop is called on click', () => {
  // mount the Button with the onClick function stub
  // https://on.cypress.io/stub
  // give the stub an alias "onClick"
  // https://on.cypress.io/as
  cy.mount(<Button label="Test button" onClick={cy.stub().as('onClick')} />)
  // click the button component
  cy.get('button').click()
  // confirm the stub function was called
  cy.get('@onClick').should('have.been.calledOnce')
})
```

Cypress solution: no changes necessary.


Specs
Button.cy.jsx
callback prop is called on click
SPIES / STUBS (1)
No commands were issued in this test.
Electron 118
TEST BUTTON

MODIFY PLAYWRIGHT TEST

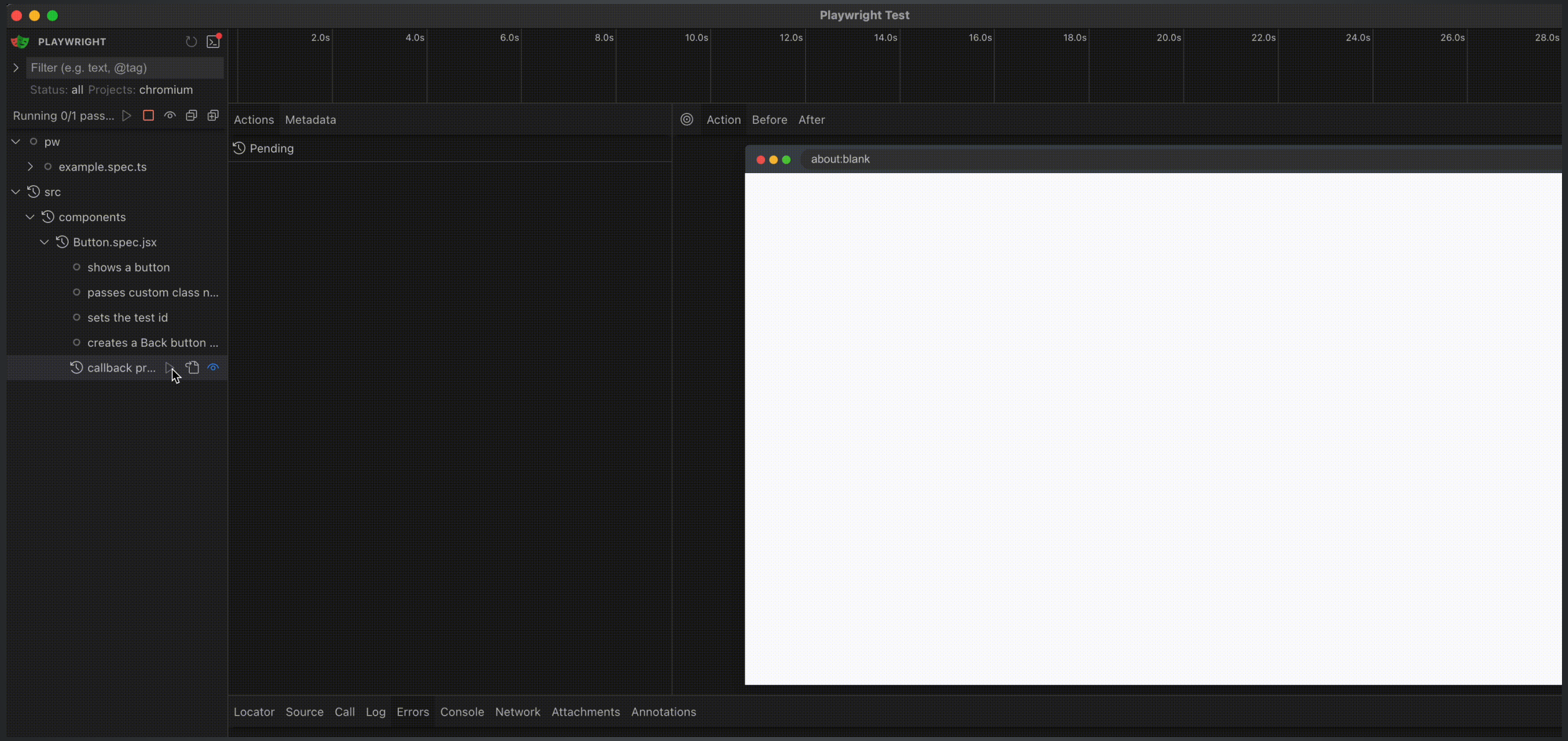
```
// src/components/Button.spec.jsx
test('callback prop is called on click', async ({ mount }) => {
  let clicked = false
  const component = await mount(
    <Button
      label="Test button"
      onClick={() => {
        clicked = true
      }}
    />
  )
  await component.click()
  expect(clicked, 'clicked').toBeTruthy()
})
```

Tip: look at auto-retrying assertions in Playwright docs

https://playwright.dev/docs/test-assertions#auto-retrying-assertions

You must make the assertion `expect(clicked, 'clicked').toBeTruthy()` retry

```
await expect
  .poll(
    () => {
      return clicked
    },
    { message: 'clicked' }
  )
  .toBeTruthy()
```


Playwright Test
PLAYWRIGHT
Filter (e.g. text, @tag)
Status: all Projects: chromium
Running 0/1 pass...
pw
example.spec.ts
src
components
Button.spec.jsx
shows a button
passes custom class n...
sets the test id
creates a Back button ...
callback pr...
2.0s
4.0s
6.0s
8.0s
10.0s
12.0s
14.0s
16.0s
18.0s
20.0s
22.0s
24.0s
26.0s
28.0s
Actions
Metadata
Pending
Action
Before
After
about:blank
Locator
Source
Call
Log
Errors
Console
Network
Attachments
Annotations

USING SINON.JS LIBRARY WITH PLAYWRIGHT

Let's add a powerful library for functional spies and stubs to make Playwright component tests easy to write.

- check out branch `g5`
- `npm install`
- `npx playwright install`

The dependencies include the `sinon` library

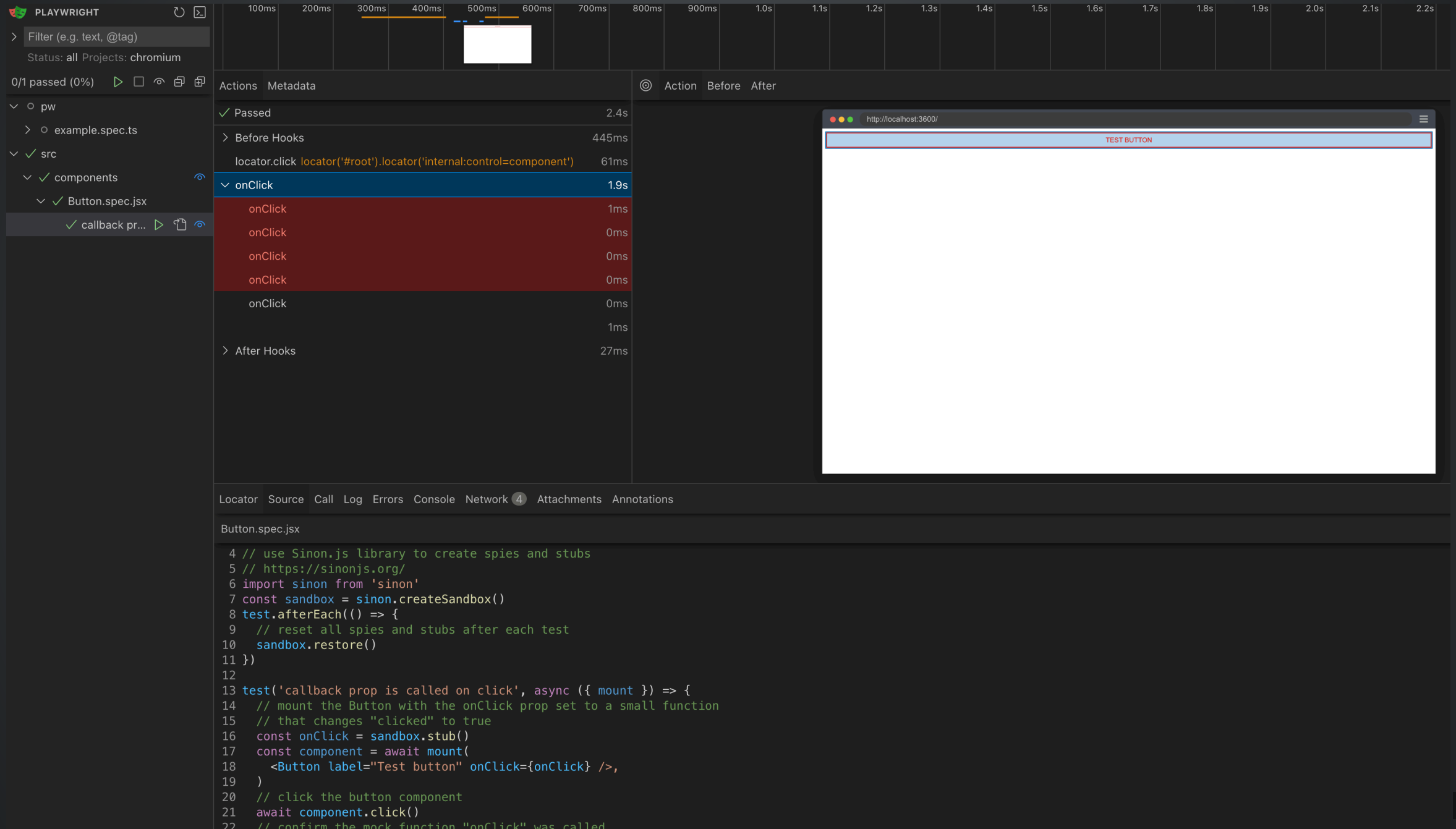
USE SINON WITH PLAYWRIGHT

```
// use Sinon.js library to create spies and stubs
// https://sinonjs.org/
import sinon from 'sinon'
const sandbox = sinon.createSandbox()
test.afterEach(() => {
  // reset all spies and stubs after each test
  sandbox.restore()
})
```

Finish the test

```
// src/components/Button.spec.jsx
test('callback prop is called on click', async ({ mount }) => {
  // mount the Button with the onClick prop set to a small function
  // that changes "clicked" to true
  // Tip: create the onClick function stub using the Sinon sandbox
  // click the button component
  // confirm the mock function "onClick" was called
})
```

```
// src/components/Button.spec.jsx
test('callback prop is called on click', async ({ mount }) => {
  // mount the Button with the onClick prop set to a small function
  // that changes "clicked" to true
  const onClick = sandbox.stub()
  const component = await mount(
    <Button label="Test button" onClick={onClick} />
  )
  // click the button component
  await component.click()
  // confirm the mock function "onClick" was called
  await expect.poll(() => onClick.calledOnce, { message: 'onClick' }).toBe(true)
})
```


PLAYWRIGHT
Filter (e.g. text, @tag)
Status: all Projects: chromium
0/1 passed (0%)
pw
example.spec.ts
src
components
Button.spec.jsx
callback pr...
100ms 200ms 300ms 400ms 500ms 600ms 700ms 800ms 900ms 1.0s 1.1s 1.2s 1.3s 1.4s 1.5s 1.6s 1.7s 1.8s 1.9s 2.0s 2.1s 2.2s
Actions Metadata
Passed 2.4s
Before Hooks 445ms
locator.click locator('#root').locator('internal:control=component') 61ms
onClick 1.9s
onClick 1ms
onClick 0ms
onClick 0ms
onClick 0ms
onClick 0ms
1ms
After Hooks 27ms
Action Before After
http://localhost:3600/
TEST BUTTON
Locator Source Call Log Errors Console Network 4 Attachments Annotations
Button.spec.jsx
4 // use Sinon.js library to create spies and stubs
5 // https://sinonjs.org/
6 import sinon from 'sinon'
7 const sandbox = sinon.createSandbox()
8 test.afterEach(() => {
9 // reset all spies and stubs after each test
10 sandbox.restore()
11 })
12
13 test('callback prop is called on click', async ({ mount }) => {
14 // mount the Button with the onClick prop set to a small function
15 // that changes "clicked" to true
16 const onClick = sandbox.stub()
17 const component = await mount(
18 <Button label="Test button" onClick={onClick} />,
19)
20 // click the button component
21 await component.click()
22 // confirm the mock function "onClick" was called

CHECK CALL ARGUMENTS

```
test('callback prop is called with arguments', async ({ mount }) => {
  // the Button component calls the "onClick" prop with a string
  // confirm the correct string is passed when the button is clicked
  // Tip: use the "stub.calledOnceWithExactly" method to check
})
```

```
test('callback prop is called with arguments', async ({ mount }) => {
  // the Button component calls the "onClick" prop with a string
  // confirm the correct string is passed when the button is clicked
  // Tip: use the "stub.calledOnceWithExactly" method to check
  const onClick = sandbox.stub()
  const component = await mount(
    <Button label="Test button" onClick={onClick} />
  )
  await component.click()
  await expect
    .poll(() => onClick.calledOnceWithExactly('Hello from button'), {
      message: 'onClick'
    })
    .toBe(true)
})
```

COMPARE SOLUTIONS

```
// Cypress
cy.mount(<Button label="Test button" onClick={cy.stub().as('onClick')} />)
cy.get('button').click()
cy.get('@onClick').should(
  'have.been.calledOnceWithExactly',
  'Hello from button'
)
// Playwright
const onClick = sandbox.stub()
const component = await mount(<Button label="Test button" onClick={onClick} />)
await component.click()
await expect
  .poll(() => onClick.calledOnceWithExactly('Hello from button'), {
    message: 'onClick'
  })
  .toBe(true)
```

COMPONENT TEST PAGE

Each test component is mounted on a special page

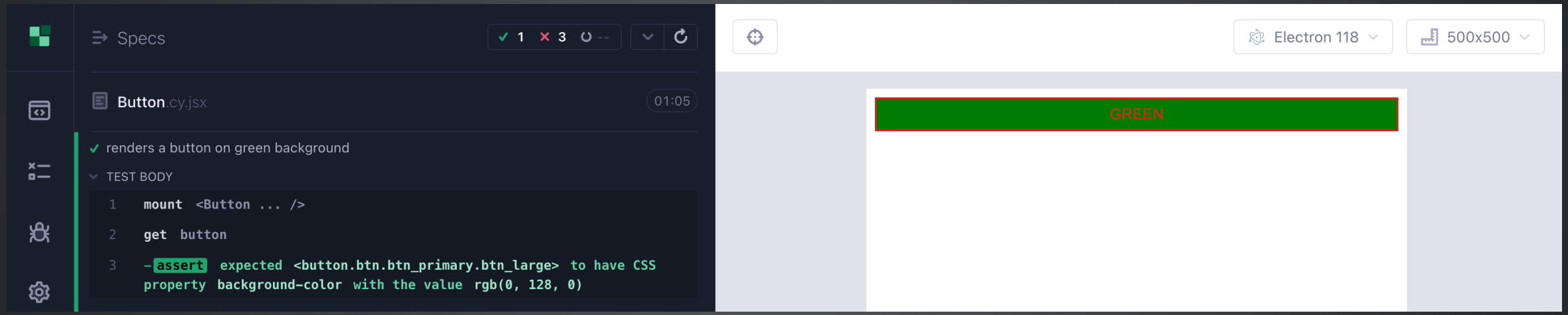
- CY `cypress/support/component-index.html`
- PW `playwright/index.html`

Let's modify it: use branch `g6`

CY: GREEN BUTTON

```
// src/components/Button.cy.jsx
it('renders a button on green background', () => {
  cy.mount(<Button label="Green" />)
  // confirm the button has a green assertion
  // Tip: use the "have.css" Chai-jQuery assertion
  // https://www.chaijs.com/plugins/chai-jquery/
})
```

Tip: modify the style in `cypress/support/component-index.html`


Specs
1
3
--
Button.cy.jsx
01:05
renders a button on green background
TEST BODY
1 mount <Button ... />
2 get button
3 -assert expected <button.btn.btn_primary.btn_large> to have CSS property background-color with the value rgb(0, 128, 0)
Electron 118
500x500
GREEN

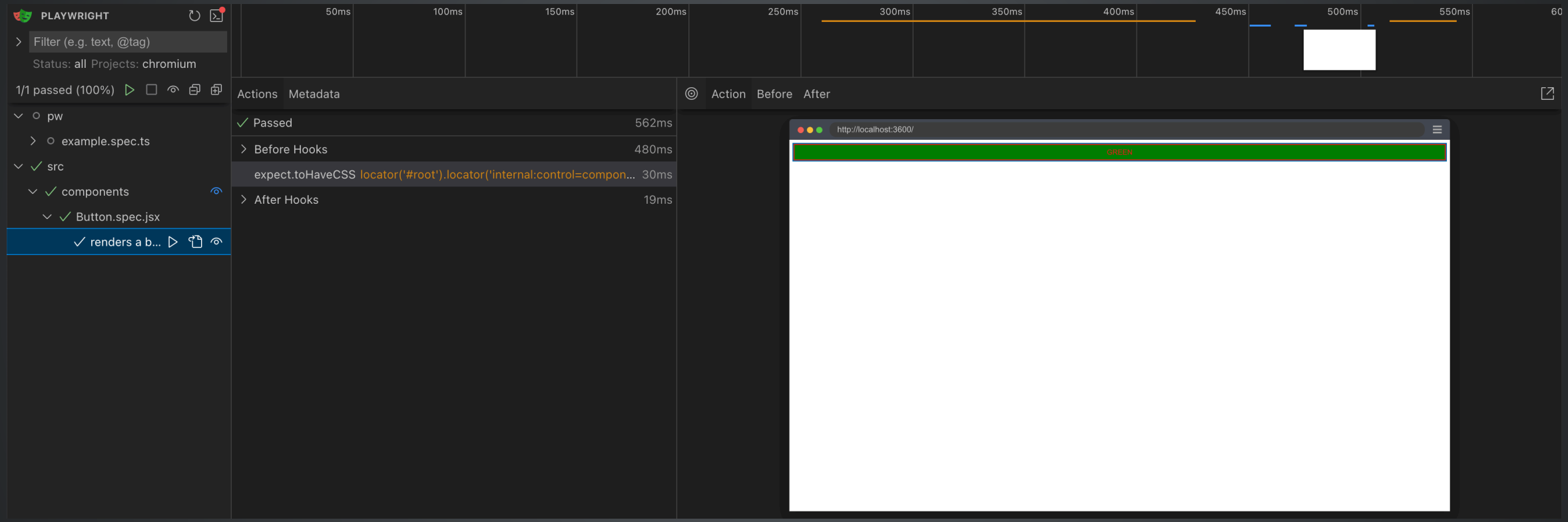
```
// src/components/Button.cy.jsx

it('renders a button on green background', () => {
  cy.mount(<Button label="Green" />)
  // confirm the button has a green assertion
  // Tip: use the "have.css" Chai-jQuery assertion
  // https://www.chaijs.com/plugins/chai-jquery/
  cy.get('button').should('have.css', 'background-color', 'rgb(0, 128, 0)')
})
```

PW: GREEN BUTTON

```
// src/components/Button.spec.jsx

test('renders a button on green background', async ({ mount }) => {
  const component = await mount(<Button label="Green" />)
  // confirm the button has a green background
  // Tip: use "toHaveCSS" assertion
  // https://playwright.dev/docs/api/class-locatorassertions
})
```

Tip: add style to the file `playwright/index.html`

```
// src/components/Button.spec.jsx

test('renders a button on green background', async ({ mount }) => {
  const component = await mount(<Button label="Green" />)
  // confirm the button has a green background
  // Tip: use "toHaveCSS" assertion
  // https://playwright.dev/docs/api/class-locatorassertions
  await expect(component).toHaveCSS('background-color', 'rgb(0, 128, 0)')
})
```

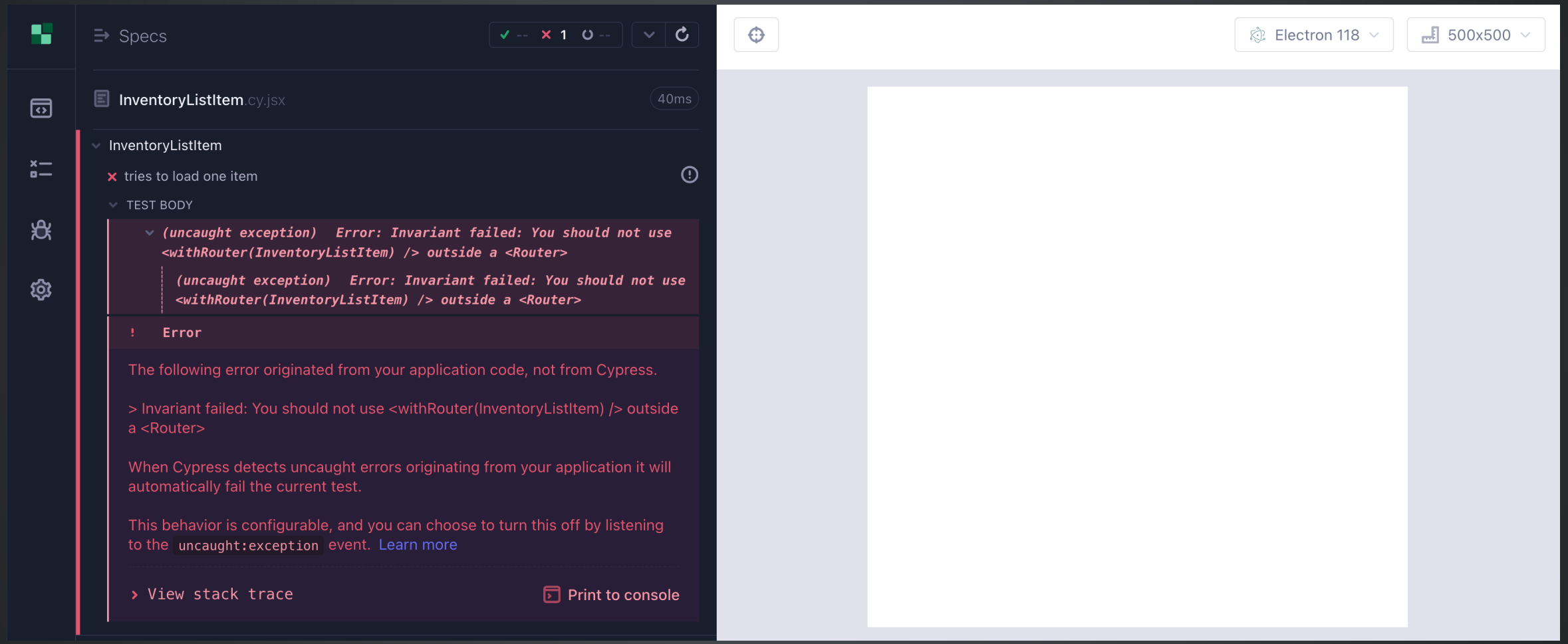

PLAYWRIGHT
Filter (e.g. text, @tag)
Status: all Projects: chromium
1/1 passed (100%)
pw
example.spec.ts
src
components
Button.spec.jsx
renders a b...
50ms
100ms
150ms
200ms
250ms
300ms
350ms
400ms
450ms
500ms
550ms
Actions
Metadata
Passed 562ms
Before Hooks 480ms
expect.toHaveCSS locator('#root').locator('internal:control=compon... 30ms
After Hooks 19ms
Action
Before
After
http://localhost:3600/
GREEN

COMPONENT SCAFFOLDING

- clone repo `https://github.com/bahmutov/taste-the-sauce-vite`
- check out branch `g7`
- `npm install`
- `npx playwright install`

TRY TESTING InventoryListItem

```
// src/components/InventoryListItem.cy.jsx

describe('InventoryListItem', () => {
  // shows the failure when we try mounting a component
  // that expects to be inside a router
  it('tries to load one item', () => {
    const item = InventoryData[3]
    cy.mount(<InventoryListItem {...item} />)
  })
})
```


Specs
-- 1 --
InventoryListItem.cy.jsx
40ms
InventoryListItem
tries to load one item
TEST BODY
(uncaught exception) Error: Invariant failed: You should not use <withRouter(InventoryListItem) /> outside a <Router>
(uncaught exception) Error: Invariant failed: You should not use <withRouter(InventoryListItem) /> outside a <Router>
! Error
The following error originated from your application code, not from Cypress.
> Invariant failed: You should not use <withRouter(InventoryListItem) /> outside a <Router>
When Cypress detects uncaught errors originating from your application it will automatically fail the current test.
This behavior is configurable, and you can choose to turn this off by listening to the uncaught:exception event. Learn more
View stack trace
Print to console
Electron 118
500x500

TODO: mount component inside the router

```
it('loads one item (with router)', () => {
  // take one of the items loaded from the inventory data list
  // and mount the "InventoryListItem" component
  // inside a router
  // <BrowserRouter initialEntries={[]}><Route>...</Route></BrowserRouter>
  const item = InventoryData[3]
  // assert that the item's name is there
  cy.contains('.inventory_item_name', item.name)
})
```

```
it('loads one item (with router)', () => {
  const item = InventoryData[3]
  cy.mount(
    <BrowserRouter initialEntries={[]}>
      <Route>
        <InventoryListItem {...item} />
      </Route>
    </BrowserRouter>
  )
  cy.contains('.inventory_item_name', item.name)
})
```

Specs

✓ 1 ✕ -- ○ --

Button.cy.jsx 47ms

InventoryListItem

✓ loads one item

TEST BODY

```
1  mount <BrowserRouter2 ... />
   -contains .inventory_item_name, Test.allTheThings() T-Shirt (Red)
```

Electron 118

Test.allTheThings() T-Shirt (Red)

This classic Sauce Labs t-shirt is perfect to wear when cozying up to your keyboard to automate a few tests. Super-soft and comfy ringspun combed cotton.

$15.99

ADD TO CART

TODO: SIMPLY THE TEST

Instead of global `cy.mount` command use `cy.mountWithRouter` from `cypress/support/component.jsx`

```
Cypress.Commands.add('mountWithRouter', (Component) => {
  return mount(
    <BrowserRouter initialEntries={[]}>
      <Route>{Component}</Route>
    </BrowserRouter>
  )
})
```

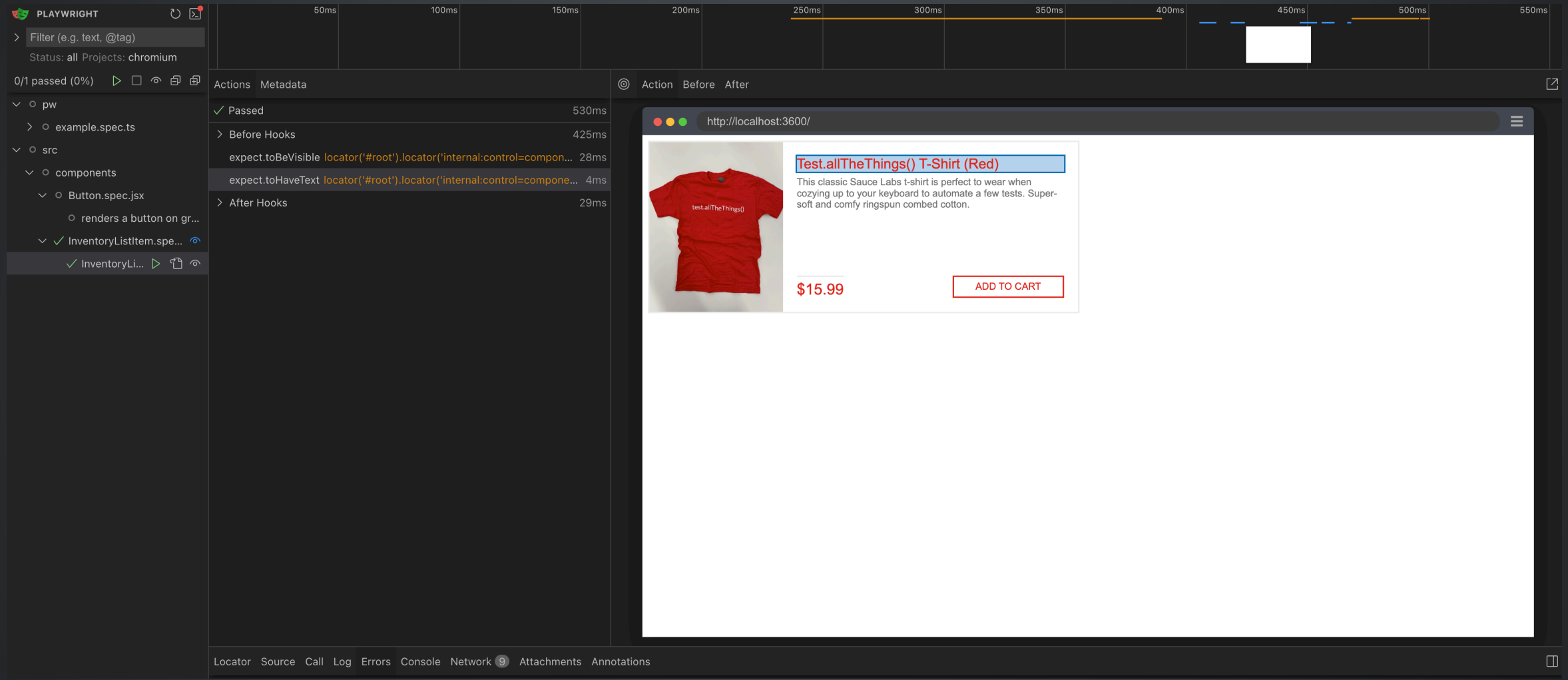
Update your test to use `cy.mountWithRouter`

```
// src/components/InventoryListItem.cy.jsx

it('loads one item', () => {
  const item = InventoryData[3]
  cy.mountWithRouter(<InventoryListItem {...item} />)
  cy.contains('.inventory_item_name', item.name)
})
```

PLAYWRIGHT MOUNT

```
// src/components/InventoryListItem.spec.jsx
test('InventoryListItem loads one item', async ({ mount }) => {
  // take one of the items loaded from the inventory data list
  // and mount the "InventoryListItem" component
  // inside a router
  // <BrowserRouter initialEntries={[]}><Route>...</Route></BrowserRouter>
  const item = InventoryData[3]
  // assert the component is visible
  // assert that the item's name is there
})
```

```
test('InventoryListItem loads one item', async ({ mount }) => {
  const item = InventoryData[3]
  const component = await mount(
    <BrowserRouter initialEntries={[]}>
      <Route>
        <InventoryListItem {...item} />
      </Route>
    </BrowserRouter>
  )
  await expect(component).toBeVisible()
  await expect(component.locator('.inventory_item_name')).toHaveText(item.name)
})
```


PLAYWRIGHT
Filter (e.g. text, @tag)
Status: all Projects: chromium
0/1 passed (0%)
pw
example.spec.ts
src
components
Button.spec.jsx
renders a button on gr...
InventoryListItem.spe...
InventoryLi...
50ms
100ms
150ms
200ms
250ms
300ms
350ms
400ms
450ms
500ms
550ms
Actions
Metadata
Passed 530ms
Before Hooks 425ms
expect.toBeVisible locator('#root').locator('internal:control=compon... 28ms
expect.toHaveText locator('#root').locator('internal:control=compone... 4ms
After Hooks 29ms
Action
Before
After
http://localhost:3600/
Test.allTheThings() T-Shirt (Red)
This classic Sauce Labs t-shirt is perfect to wear when cozying up to your keyboard to automate a few tests. Super-soft and comfy ringspun combed cotton.
$15.99
ADD TO CART
Locator
Source
Call
Log
Errors
Console
Network 9
Attachments
Annotations

Todo: use helper function to mount inside a Router

```
// Helper function to mount the component inside a router
// to avoid repeating the router setup in each test
// Note: in the real application, you probably would use Playwright Test Fixtures
// https://playwright.dev/docs/test-fixtures
function mountWithRouter(mountFn, component) {}

test('InventoryListItem loads one item (mount helper)', async ({ mount }) => {
  // rewrite the above test to use the "mountWithRouter" helper function
  // to avoid repeating the router setup
  const item = InventoryData[3]
  // confirm the component is visible
  // and has the correct item name
})
```

```
function mountWithRouter(mountFn, component) {
  return mountFn(
    <BrowserRouter initialEntries={[]}>
      <Route>{component}</Route>
    </BrowserRouter>
  )
}

test('InventoryListItem loads one item (mount helper)', async ({ mount }) => {
  const item = InventoryData[3]
  const component = await mountWithRouter(
    mount,
    <InventoryListItem {...item} />
  )
  await expect(component).toBeVisible()
  await expect(component.locator('.inventory_item_name')).toHaveText(item.name)
```

Tip: for a better way of doing this see Pw Test fixtures

COMMUNICATION BETWEEN THE COMPONENT AND THE TEST

- clone repo `https://github.com/bahmutov/taste-the-sauce-vite`
- check out branch `g8`
- `npm install`
- `npx playwright install`

PRICE FORMAT COMPONENT

```
// src/components/InputPrice.jsx
const InputPrice = ({ priceFormatter }) => {
  const formatter = priceFormatter || defaultPriceFormatter
  const [price, setPrice] = useState()

  return (
    <div className="input-price">
      <input
        type="text"
        value={price}
        onChange={(e) => setPrice(e.target.value)}
        placeholder="Enter price (cents)"
      />{' '}
      {!isNaN(price) && <span className="price">{formatter(price)}</span>}
    </div>
  )
```

CYPRESS: TEST THE DEFAULT FORMAT FUNCTION

```
// src/components/InputPrice.cy.jsx
it('renders the InputPrice component', () => {
  // mount the component using the default price formatter
  cy.mount(<InputPrice />)
  // initially the formatted price is not displayed
  cy.get('.price').should('not.exist')
  // type the "9" and confirm the formatted price is "$0.09"
  cy.get('[placeholder="Enter price (cents)"]').type('9')
  cy.get('.price').should('be.visible').and('have.text', '$0.09')
  // type another "9" and confirm the formatted price is "$0.99"
  cy.get('[placeholder="Enter price (cents)"]').type('9')
  cy.get('.price').should('have.text', '$0.99')
})
```

The test is working.

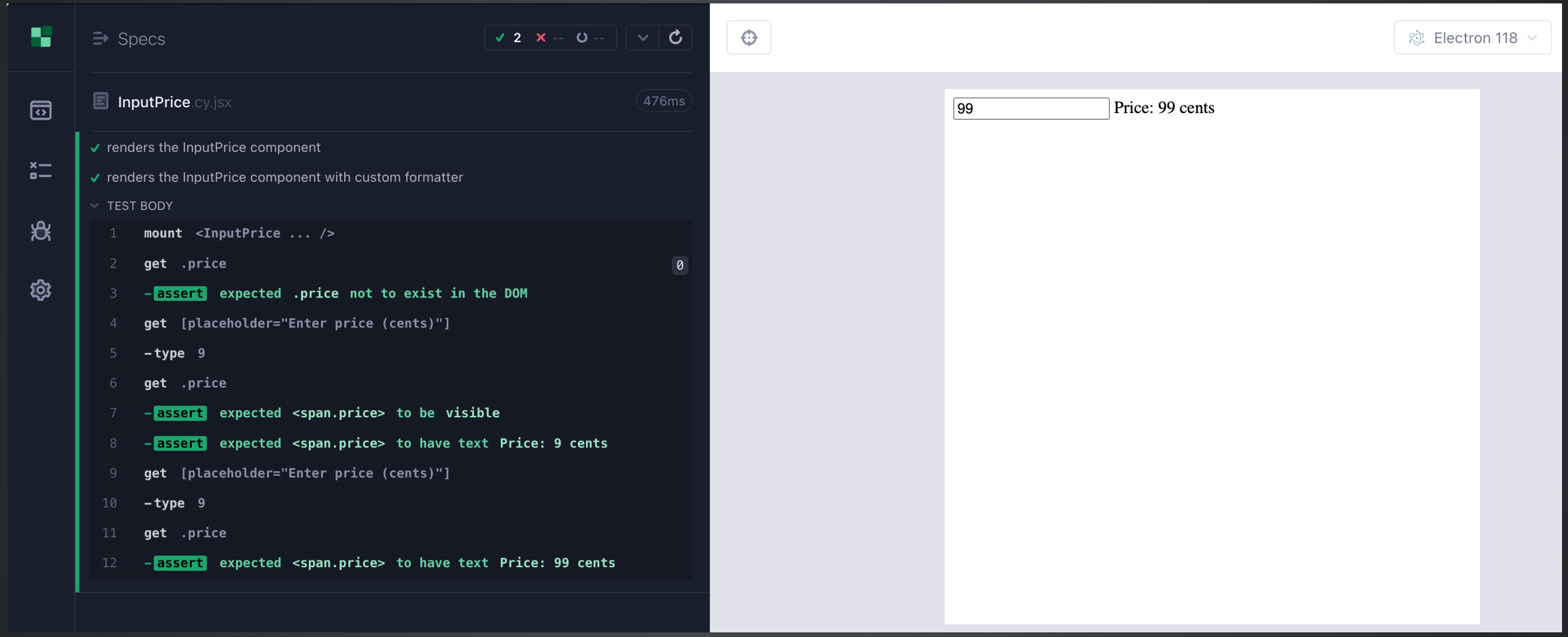
FINISH THE SECOND TEST

Test the custom format function

```
it('renders the InputPrice component with custom formatter', () => {
  const customFormatter = (price) => `Price: ${price} cents`
  // mount the component with a custom format function above
  // follow the test above and check if the formatted price is displayed
})
```

CYPRESS SOLUTION

```
it('renders the InputPrice component with custom formatter', () => {
  const customFormatter = (price) => `Price: ${price} cents`
  // mount the component with a custom format function above
  // follow the test above and check if the formatted price is displayed
  cy.mount(<InputPrice priceFormatter={customFormatter} />)
  cy.get('.price').should('not.exist')
  cy.get('[placeholder="Enter price (cents)"]').type('9')
  cy.get('.price').should('be.visible').and('have.text', 'Price: 9 cents')
  cy.get('[placeholder="Enter price (cents)"]').type('9')
  cy.get('.price').should('have.text', 'Price: 99 cents')
})
```


Specs
2
InputPrice.cy.jsx
476ms
renders the InputPrice component
renders the InputPrice component with custom formatter
TEST BODY
1 mount <InputPrice ... />
2 get .price 0
3 -assert expected .price not to exist in the DOM
4 get [placeholder="Enter price (cents)"]
5 -type 9
6 get .price
7 -assert expected <span.price> to be visible
8 -assert expected <span.price> to have text Price: 9 cents
9 get [placeholder="Enter price (cents)"]
10 -type 9
11 get .price
12 -assert expected <span.price> to have text Price: 99 cents
Electron 118
99
Price: 99 cents

PLAYWRIGHT: TEST THE DEFAULT FORMAT FUNCTION

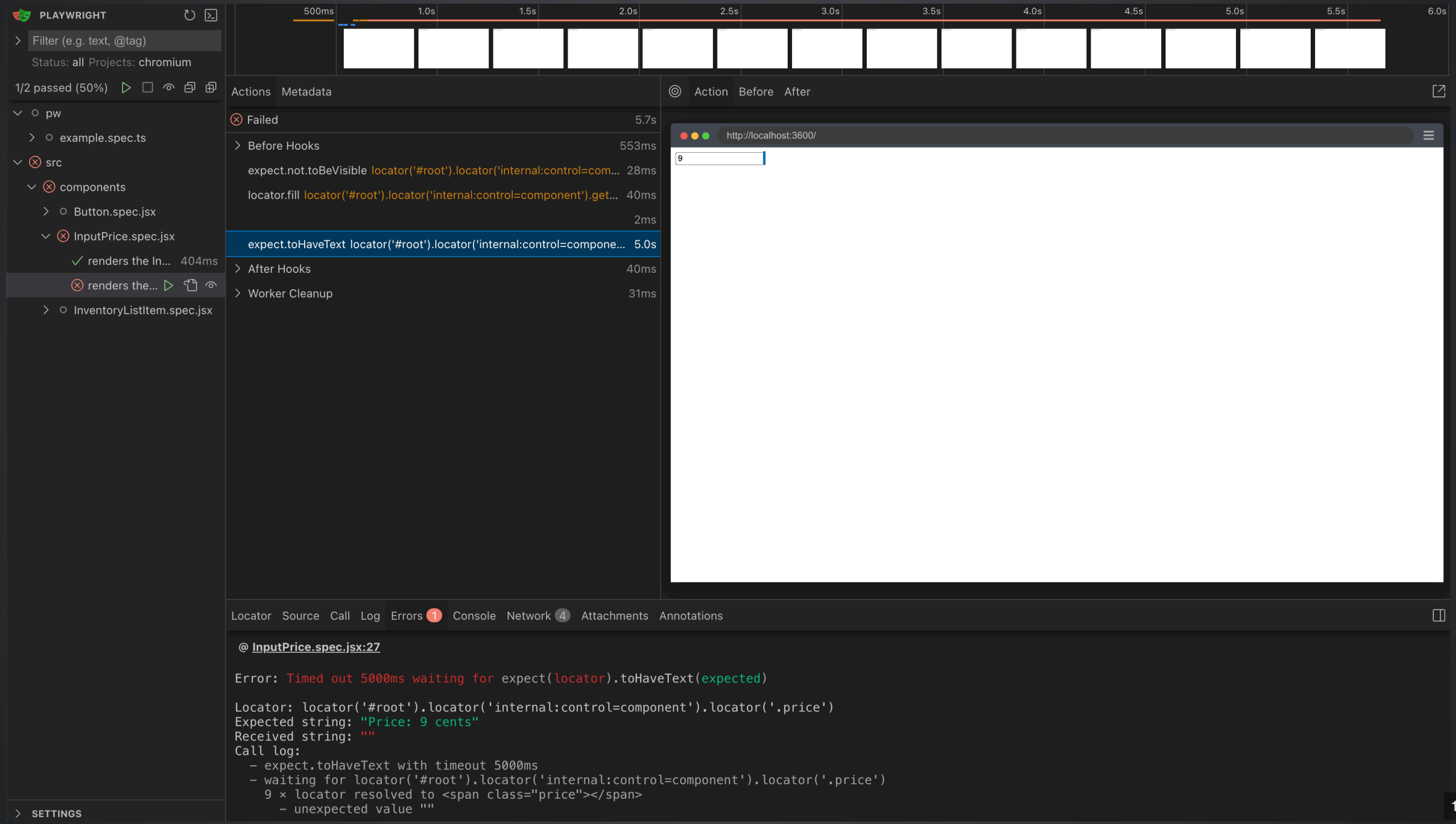
```
// src/components/InputPrice.spec.jsx
test('renders the InputPrice component', async ({ mount }) => {
  // mount the component using the default price formatter
  const component = await mount(<InputPrice />)
  // initially the formatted price is not displayed
  await expect(component.locator('.price')).not.toBeVisible()
  // type the "9" and confirm the formatted price is "$0.09"
  await component.getByPlaceholder('Enter price (cents)').fill('9')
  await expect(component.locator('.price')).toHaveText('$0.09')
  // type another "9" and confirm the formatted price is "$0.99"
  await component.getByPlaceholder('Enter price (cents)').fill('99')
  await expect(component.locator('.price')).toHaveText('$0.99')
})
```

TODO: FINISH TESTING THE CUSTOM FORMAT FUNCTION

```
test('renders the InputPrice component with custom formatter', async ({
  mount
}) => {
  const customFormatter = (price) => `Price: ${price} cents`
  // mount the component with a custom format function above
  // follow the test above and check if the formatted price is displayed
})
```

SOLUTION ATTEMPT

```
test('renders the InputPrice component with custom formatter', async ({
  mount
}) => {
  const customFormatter = (price) => `Price: ${price} cents`
  // mount the component with a custom format function above
  // follow the test above and check if the formatted price is displayed
  const component = await mount(<InputPrice priceFormatter={customFormatter} />)
  await expect(component.locator('.price')).not.toBeVisible()

  await component.getByPlaceholder('Enter price (cents)').fill('9')
  await expect(component.locator('.price')).toHaveText('Price: 9 cents')
  await component.getByPlaceholder('Enter price (cents)').fill('99')
  await expect(component.locator('.price')).toHaveText('Price: 99 cents')
})
```


PLAYWRIGHT
Filter (e.g. text, @tag)
Status: all Projects: chromium
1/2 passed (50%)
pw
example.spec.ts
src
components
Button.spec.jsx
InputPrice.spec.jsx
renders the In... 404ms
renders the...
InventoryListItem.spec.jsx
SETTINGS
500ms 1.0s 1.5s 2.0s 2.5s 3.0s 3.5s 4.0s 4.5s 5.0s 5.5s 6.0s
Actions Metadata
Failed 5.7s
Before Hooks 553ms
expect.not.toBeVisible locator('#root').locator('internal:control=com... 28ms
locator.fill locator('#root').locator('internal:control=component').get... 40ms
2ms
expect.toHaveText locator('#root').locator('internal:control=compone... 5.0s
After Hooks 40ms
Worker Cleanup 31ms
Action Before After
http://localhost:3600/
9
Locator Source Call Log Errors 1 Console Network 4 Attachments Annotations
@ InputPrice.spec.jsx:27
Error: Timed out 5000ms waiting for expect(locator).toHaveText(expected)
Locator: locator('#root').locator('internal:control=component').locator('.price')
Expected string: "Price: 9 cents"
Received string: ""
Call log:
- expect.toHaveText with timeout 5000ms
- waiting for locator('#root').locator('internal:control=component').locator('.price')
9 × locator resolved to
- unexpected value ""

PW COMPONENT <-> TEST COMMUNICATION

Pw has to switch every synchronous call to a Promise-returning call to communicate between the browser (component) and spec (Node)

```
<!-- normal component execution in the browser: formatter is in the browser -->
<span className="price">{formatter(price) // yields a string}</span>
<!-- playwright test execution: formatter is in the spec file -->
<span className="price">{formatter(price) // yields Promise<string>}</span>
```

🏁 CONCLUSIONS

- Cypress can run component tests the same way as its regular E2E tests
- Playwright creates a separate page with the component and "bridges" spec code to interact with the component
- Cypress has functional assertions
- Cypress interacts much more directly with the component

➡️ Go to the end chapter

Learn more: 📺 "Component Testing With Murat Ozcan: Cy vs Pw vs Vitest" https://youtu.be/SPSLeGUpdYA

WRITE E2E + COMPONENT TESTS

using Cypress and / or Playwright

```
# E2E candidates
Cart
  - items added, deleted
Cart page
  - zero items
  - several items
  - navigates to the item
Checkout flow
  - buy an item
Inventory page
  - has all the items
  - navigation
  - sorting
```

```
# Component test candidates
src/pages/InventoryItem.jsx
  - shows item details
  - adds an item to the cart
  - stores the cart in the local storage
  - handles non-existent item
src/components/InventoryListItem.jsx
  - adds an item to the cart
src/components/ErrorMessage.jsx
  - shows an error message
  - reacts to the user click
src/utils/Sorting.js
  - unit tests for the sorting functions
```