Playwright the new selenium killer

Tamir Reiss

Agenda

Topic one

Topic two

Topic three

Topic four

Topic five




Introduction

- Playwright is a powerful testing tool that provides reliable end-to-end testing and cross browser testing for modern web applications.

Lets compare it to selenium

SELENIUM

- sends each command as a separate HTTP request and receives JSON responses.
- So every interaction, such as opening the browser, clicking an element, or sending keys in Selenium, is sent as a separate HTTP request.
- As you can imagine, this results in slower execution as we wait for responses and introduces a layer of potential flakiness

PLAYWRIGHT

- communicates all requests through a single WebSocket connection
- which stays in place until test execution is completed.
- This reduces the points of failure and allows commands to be sent quickly on a single connection.
- Does not have an equivalent to selenium grid

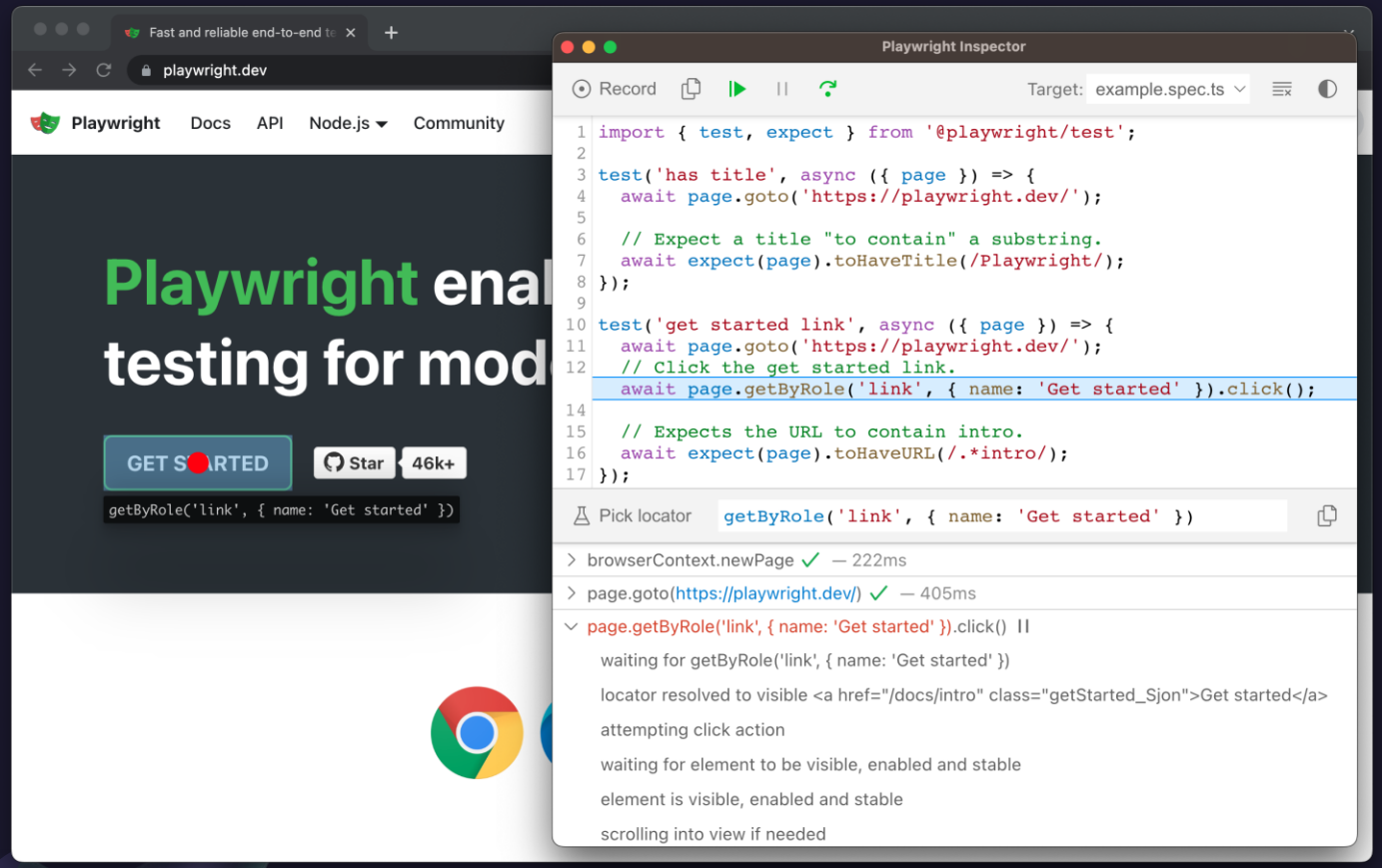

pytest
Se
pytest
Playwright

Playwright Vs selenium Main differences

	Playwright	Selenium
Compatible Languages	Java, Python, .NET, C#, TypeScript, JavaScript	Java, Python, C#, Ruby, Perl, PHP, JavaScript, Kotlin
Browser Support	Chromium, WebKit, Firefox	Chrome, Safari, Firefox, Opera, Edge, IE
Operating System Support	Windows, Linux, macOS	Windows, Linux, macOS, Solaris
Real Device Support	Native mobile emulation and experimental real Android support	Real device clouds and remote servers
Community	A small but growing community	Established a collection of documentation along with a huge community

Playwright Inspector

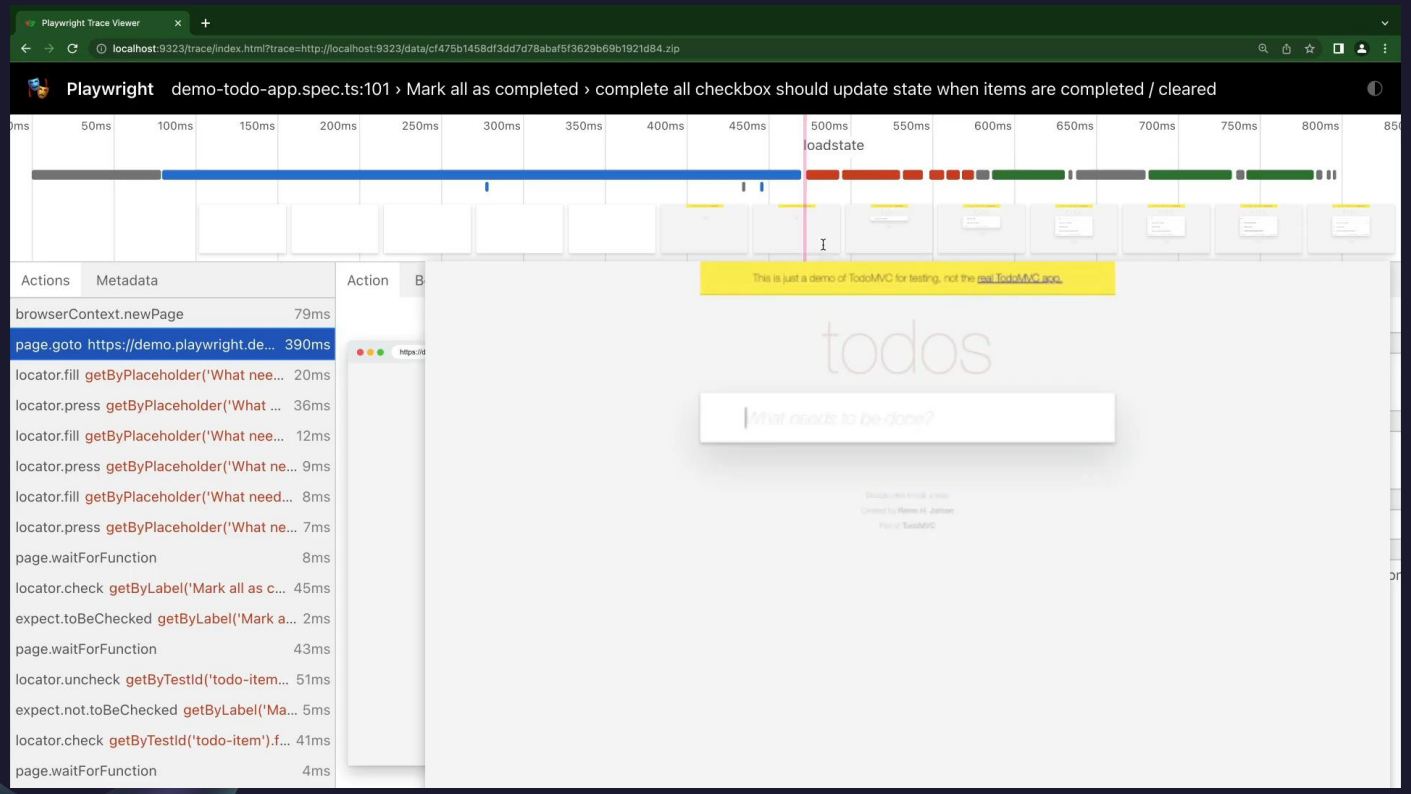
The Playwright Inspector is a GUI tool to help you debug your Playwright tests. It allows you to step through your tests, live edit locators, pick locators and see actionability logs.



- **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**
- 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.

Playwright Trace Viewer

Playwright Trace Viewer is a GUI tool that helps you explore recorded Playwright traces after the script has ran.



- Action – On the left side you will see a list of actions
- Metadata -such as the time the action was performed, what browser engine was used, what the viewport was and if it was mobile and how many pages, actions and events were recorded.
- Screenshots -When tracing with the screenshots option turned on, each trace records a screencast and renders it as a film strip.
- Can also record :
  - Console
  - Network
  - Source

Trace example recording and opening

TRACE CODE EXAMPLE

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch()
    context = browser.new_context()
    context.tracing.start(screenshots=True, snapshots=True, sources=True)
    page = context.new_page()
    page.goto("http://playwright.dev")
    page.get_by_role("link", name="Docs").click()
    with page.expect_popup() as page1_info:
        page.get_by_role("link", name="GitHub repository").click()
    page1 = page1_info.value
    print(page.title())
    print(page1.title())
    context.tracing.stop(path="trace.zip")
    browser.close()
```

OPEN LOCALLY

playwright show-trace trace.zip

USING WEB

https://trace.playwright.dev/

Demo

Locators

LOCATORS ARE THE CENTRAL PIECE OF PLAYWRIGHT'S AUTO-WAITING AND RETRY-ABILITY. IN A NUTSHELL, LOCATORS REPRESENT A WAY TO FIND ELEMENT(S) ON THE PAGE AT ANY MOMENT.

These are the recommended built in locators.

- page.get_by_role() to locate by explicit and implicit accessibility attributes.
- page.get_by_text() to locate by text content.
- page.get_by_label() to locate a form control by associated label's text.
- page.get_by_placeholder() to locate an input by placeholder.
- page.get_by_alt_text() to locate an element, usually image, by its text alternative.
- page.get_by_title() to locate an element by its title attribute.
- page.get_by_test_id() to locate an element based on its data-testid attribute (other attributes can be configured).

Page.Locator() Locate by CSS or XPath

If you absolutely must use CSS or XPath locators,
you can use page.locator()
to create a locator that takes a selector describing how to find an element in the page.
Playwright supports CSS and XPath selectors,
and auto-detects them if you omit `css=` or `xpath=` prefix

```
<x-details role=button aria-expanded=true aria-controls=inner-details>
<div>Title</div>
#shadow-root
<div id=inner-details>Details</div>
</x-details>
```

```
page.get_by_text("Details").click()
page.locator("x-details", has_text="Details" ).click()
```

Filter elements

- FILTER BY TEXT
- FILTER BY CHILD/DESCENDANT
- FILTER BY MATCHING AN ADDITIONAL LOCATOR

```
<ul>
 <li>
   <h3>Product 1</h3>
   <button>Add to cart</button>
 </li>
 <li>
   <h3>Product 2</h3>
   <button>Add to cart</button>
 </li>
</ul>
```

```
page.get_by_role("listitem").filter(has_text="Product 2").get_by_role("button“ ,  name="Add to cart").click()
```

```
expect(page.get_by_role("listitem").filter(has_not_text="Out of stock")) .to_have_count(5)
```

Page object

- Playwright page object is slightly difference then the selenium Page object
- The main difference that you can initial the object before the page is open and no need to pass the driver from one page to another
- There is no need to init the object again if the page was reenter
- The locaters can be “search” in the constrictor

```
def test_first_play_wright(_page):
    login_page = loginPage(_page)
    main_page = mainPage(_page)
    main_page.navigate()
    main_page.Sign_in_btn.click()

login_page.login("test@sela.com","123456")
```

```
from playwright.sync_api import Page

class loginPage:

    def __init__(self,page : Page):
        self.page = page
        self.email_txt = page.locator("#email")
        self.password_txt = page.locator('//*[@id = "passwd"]')
        self.submit_btn = page.locator('//*[@name = "SubmitLogin"]')

    def login(self,user,password):
        self.email_txt.fill(user)
        self.password_txt.fill(password)
        self.submit_btn.click()
```

Demo