

TÌM HIỂU NÂNG CAO VỀ PLAYWRIGHT

Contents

I. LÝ THUYẾT	3
1. TÌM HIỂU CÁC HÀM CƠ BẢN CỦA PLAYWRIGHT.....	3
1.1. Goto	3
1.2. Fill	3
1.3. Click	3
1.4. Screenshot.....	3
1.5. Expect	4
1.6. Title.....	4
1.7. Locator	4
1.8. Check & uncheck	4
1.9. Hover	4
1.10. Press	5
1.11. Close	5
1.12. setViewportSize	5
2. TÌM HIỂU VỀ LOCATOR	5
2.1. Giới thiệu về locator	5
2.2. Các cách get locator	6
2.3. Locator động và tĩnh	11
2.4. Các kỹ thuật định vị locator phổ biến	12
3. TÌM HIỂU VỀ CÁCH CONFIG PROJECT	17
3.1. Các configuration cơ bản của playwright.config.ts.....	17
3.2. Cách config multi-env	22
3.3. Cách dùng trace viewer, debug mode, video record.	25
4. TÌM HIỂU VỀ DATA TEST	27
4.1. Đọc và sử dụng dữ liệu từ JSON.....	27

4.2.	Đọc và sử dụng dữ liệu từ CSV	28
5.	REPORT	29
5.1.	HTML	29
5.2.	ALLURE.....	30

I. LÝ THUYẾT

1. TÌM HIỂU CÁC HÀM CƠ BẢN CỦA PLAYWRIGHT

1.1. Goto

- Hàm goto(url, option) dùng để điều hướng trình duyệt tới một URL cụ thể, trả về response chính của trang sau khi điều hướng.
- Nếu gặp lỗi như SSL, URL không hợp lệ hoặc timeout, hàm sẽ báo lỗi

VD: `await page.goto('https://example.com');`

1.2. Fill

- Hàm fill(value, option) dùng để điền dữ liệu vào trường nhập liệu như <input>. Hàm sẽ tự động chờ phần tử sẵn sàng rồi mới điền và kích hoạt sự kiện input.
- Nếu phần tử không tồn tại hoặc không phải trường nhập liệu, hàm sẽ báo lỗi

VD: `await page.locator('input[name="username"]').fill('admin');`

1.3. Click

- Hàm click(options) thực hiện thao tác click chuột lên phần tử đã được chọn. Hàm này sẽ chờ phần tử hiển thị, có thể tương tác, cuộn phần tử vào view rồi mới click.
- Hỗ trợ click trái, phải, giữa, đúp, click kèm phím sửa đổi.

VD: `await page.locator('button#submit').click();`

`await page.locator('button#right-click').click({ button: 'right' });`

`await page.locator('button#middle-click').click({ button: 'middle' });`

`await page.locator('button#double-click').dblclick();`

`await page.locator('button#ctrl-click').click({ modifiers: ['Control'] });`

`await page.locator('button#shift-click').click({ modifiers: ['Shift'] });`

1.4. Screenshot

- Hàm screenshot(options) dùng để chụp ảnh màn hình trang hoặc phần tử, vùng cụ thể. Có thể lưu ảnh ra file hoặc lấy ảnh dưới dạng buffer.

VD: `await page.screenshot({ path: 'screen.png', fullPage: true });`

// Chụp ảnh một vùng cụ thể (clipped screenshot)

`await page.screenshot({`

```
path: 'region-screenshot.png',  
clip: { x: 0, y: 0, width: 500, height: 300 } // tọa độ top-left và kích thước vùng chụp  
});  
  
// Chụp ảnh một phần tử cụ thể  
  
const element = page.locator('header');  
  
await element.screenshot({ path: 'element-screenshot.png' });
```

1.5. Expect

- Hàm expect(value) dùng để so sánh, kiểm tra các điều kiện trong test.
- Có nhiều matcher để so sánh như toBe, toEqual, toContainText, toBeVisible, toHaveAttribute, ...

VD:

```
await expect(page).toHaveURL(/example\.com/);  
  
await expect(page.locator('h1')).toHaveText('Welcome');  
  
await expect(page.locator('#agree')).toBeChecked();
```

1.6. Title

- Lấy tiêu đề của trang hiện tại và trả về kiểu string.

VD: const title = await page.title();

1.7. Locator

- Tạo locator để thao tác với element trong trang, từ đó gọi các hành động như click, fill, check, expect.

VD: await page.locator('button.login').click()

1.8. Check & uncheck

- Check hoặc bỏ check của checkbox hoặc radio.

VD:

```
await page.locator('#agree').check();  
  
await page.locator('#agree').unchecked();
```

1.9. Hover

- Di chuột đến phần tử, hữu ích để kiểm thử các tooltip, menu xuất hiện khi hover.

VD: `await page.locator('.menu-item').hover();`

1.10. Press

- `Press(key, option)` gửi phím bấm keyboard vào phần tử.

VD: `await page.locator('input').press('Enter');`

1.11. Close

- `Close()` đóng page hoặc browser.

VD:

`await page.close();`

`await browser.close();`

1.12. setViewportSize

- Thiết lập kích thước viewport của trình duyệt.

VD: `await page.setViewportSize({ width: 1280, height: 720 });`

2. TÌM HIỂU VỀ LOCATOR

2.1. Giới thiệu về locator

- Định nghĩa: Locator là cách để tìm một hoặc nhiều phần tử trên trang web tại thời điểm sử dụng.
- Vai trò: Trung tâm của tính năng tự động chờ (auto-waiting) và thử lại (retry-ability) trong Playwright.
- Điểm nổi bật:
 - Auto-waiting (tự động chờ): Locator sẽ tự động chờ phần tử xuất hiện, sẵn sàng tương tác trước khi thao tác.
 - Retry-ability (tự động thử lại): Nếu phần tử chưa có sẵn hoặc thay đổi DOM, Locator sẽ thử lại nhiều lần cho đến khi thành công hoặc timeout.
 - Strict mode (chế độ nghiêm ngặt): Mọi thao tác Locator yêu cầu phải chỉ định đúng một phần tử duy nhất, tránh thao tác nhầm khi có nhiều phần tử giống nhau.

Strictness

Locators are strict. This means that all operations on locators that imply some target DOM element will throw an exception if more than one element matches. For example, the following call throws if there are several buttons in the DOM:

Throws an error if more than one

```
await page.getByRole('button').click();
```

On the other hand, Playwright understands when you perform a multiple-element operation, so the following call works perfectly fine when the locator resolves to multiple elements.

Works fine with multiple elements

```
await page.getByRole('button').count();
```

You can explicitly opt-out from strictness check by telling Playwright which element to use when multiple elements match, through `locator.first()`, `locator.last()`, and `locator.nth()`. These methods are **not recommended** because when your page changes, Playwright may click on an element you did not intend. Instead, follow best practices above to create a locator that uniquely identifies the target element.

- Lazy evaluation (đánh giá trễ): Locator không tìm phần tử ngay khi khởi tạo, mà chỉ khi hành động thực sự được gọi mới tìm phần tử cập nhật mới nhất.
- Mỗi lần locator được sử dụng sẽ lấy phần tử cập nhật mới nhất từ DOM. Ở ví dụ dưới, DOM element sẽ được định vị 2 lần, trước mỗi hành động (hover, click) thay vì lấy phần tử cũ. Vì thế, nếu DOM thay đổi giữa các lần gọi do re-render, phần tử mới nhất sẽ được sử dụng. Điều này hữu ích với giao diện web động.

2.2. Các cách get locator

a. Locate by role

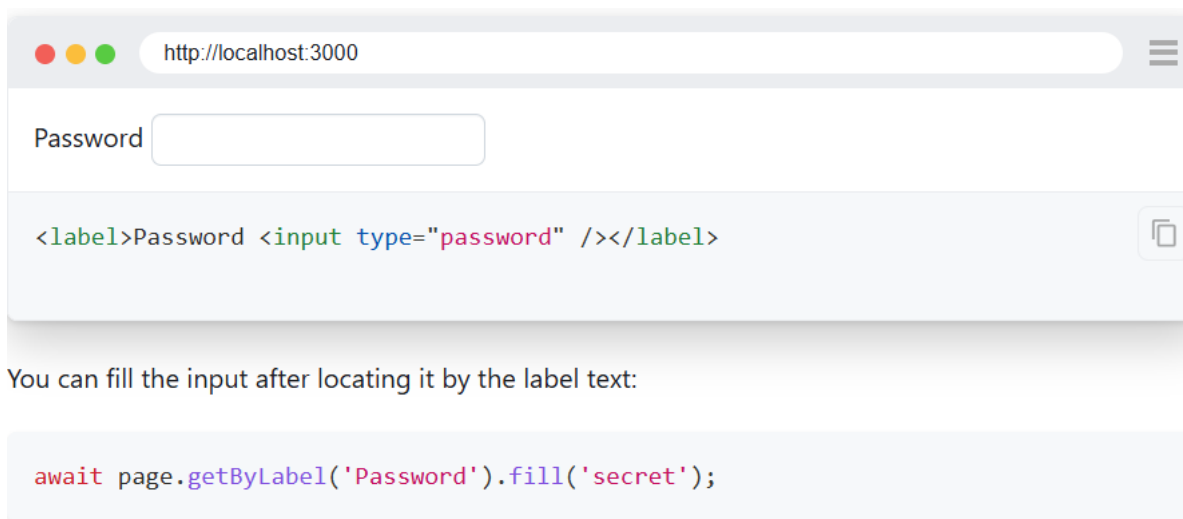
- `page.getByRole()` phản ánh cách người dùng hoặc công nghệ nhận thức trang. Tìm phần tử theo vai trò (button, checkbox, heading...) và tên truy cập.



The screenshot shows a web browser window with the address bar displaying 'http://localhost:3000'. The page content includes a heading 'Sign up', a checkbox labeled 'Subscribe', and a 'Submit' button. Below the form, the HTML code is displayed in a light blue box:

```
<h3>Sign up</h3>
<label>
  <input type="checkbox" /> Subscribe
</label>
<br/>
<button>Submit</button>
```

- `await expect(page.getByRole('heading', { name: 'Sign up' })).toBeVisible();`
 - `await page.getByRole('checkbox', { name: 'Subscribe' }).check();`
 - `await page.getByRole('button', { name: /submit/i }).click();`
- b. Locate by label
- `page.getByLabel(label)`: Tìm phần tử form theo label.



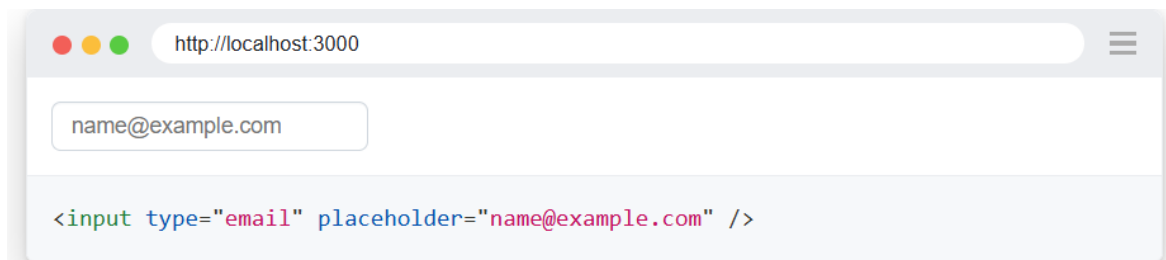
The screenshot shows a web browser window with the address bar displaying 'http://localhost:3000'. The page content includes a label 'Password' followed by an input field. Below the form, the HTML code is displayed in a light blue box:

```
<label>Password <input type="password" /></label>
```

You can fill the input after locating it by the label text:

```
await page.getByLabel('Password').fill('secret');
```

- c. Locate by placeholder
- `page.getByPlaceholder(text)`: Tìm input có placeholder tương ứng.

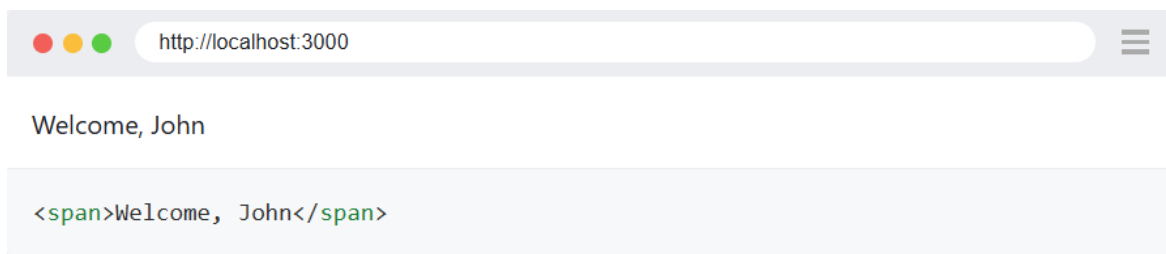


You can fill the input after locating it by the placeholder text:

```
await page
  .getByPlaceholder('name@example.com')
  .fill('playwright@microsoft.com');
```

d. Locate by text:

- `Page.getByText(text)`: Tìm phần tử chứa văn bản cụ thể, hỗ trợ chuỗi hoặc regex.



You can locate the element by the text it contains:

```
await expect(page.getByText('Welcome, John')).toBeVisible();
```

Set an exact match:

```
await expect(page.getByText('Welcome, John', { exact: true })).toBeVisible();
```

Match with a regular expression:

```
await expect(page.getByText(/welcome, [A-Za-z]+$/i)).toBeVisible();
```

e. Locate by alt text

- `Page.getByAltText(altText)`: tìm ảnh dựa trên thuộc tính alt (để mô tả ảnh).



You can click on the image after locating it by the text alternative:

```
await page.getByAltText('playwright logo').click();
```

f. Locate by title

- `page.getByTitle(title)`: Tìm phần tử có thuộc tính title.

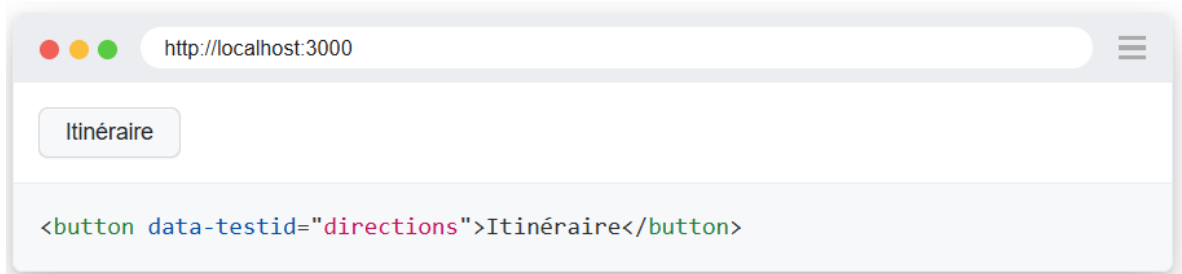


You can check the issues count after locating it by the title text:

```
await expect(page.getByTitle('Issues count')).toHaveText('25 issues');
```

g. Locate by test id

- `page.getByTestId(testId)`: Tìm phần tử dựa trên thuộc tính data-testid.
- Đây là cách kiểm thử linh hoạt nhất vì thậm chí text hoặc role của thuộc tính thay đổi, kiểm thử vẫn thành công. Tuy nhiên nếu nhìn nhận role hoặc giá trị text quan trọng, xem xét việc sử dụng locators hướng giao diện người dùng như role và text locators.



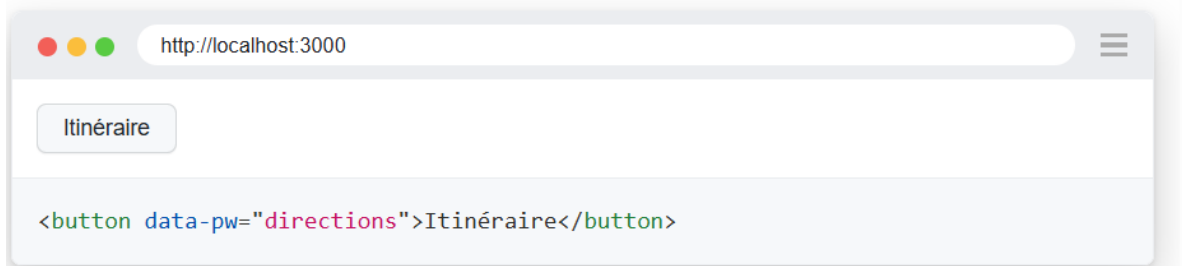
You can locate the element by its test id:

```
await page.getByTestId('directions').click();
```

- Mặc định, `page.getByTestId()` sẽ định vị phần tử dựa trên thuộc tính `data-testid`, nhưng có thể thay đổi cấu hình trong file `playwright.config.ts` thành thuộc tính khác.



In your html you can now use `data-pw` as your test id instead of the default `data-testid`.



h. Locate by CSS or Xpath

- `Page.locator()` để định vị bằng CSS và Xpath selectors.

VD:

```
await page.locator('button').click(); // CSS selector
```

```
await page.locator('//button').click(); // Xpath selector
```

2.3. Locator động và tĩnh

- a. Locator tĩnh (static locator)
 - Locator tĩnh (Static locator) là loại locator xác định phần tử dựa trên các thuộc tính cố định như ID, class, hoặc cấu trúc DOM không thay đổi theo thời gian. Nó không thay đổi ngay cả khi trang web tải lại hoặc có các tương tác khác xảy ra. Ví dụ, một selector CSS cố định như #submit-button hoặc XPath cố định /html/body/div/button[1] là locator tĩnh. Tuy nhiên, locator tĩnh có nhược điểm là dễ bị hỏng nếu DOM thay đổi hoặc thuộc tính phần tử bị thay đổi.
- b. Locator động (dynamic locator)
 - Locator động (Dynamic locator) là loại locator xác định phần tử một cách linh hoạt, thường dựa trên các thuộc tính thay đổi, vị trí, hoặc kết hợp nhiều điều kiện để tìm đúng phần tử ngay cả khi cấu trúc DOM có biến đổi. Playwright locator theo API cung cấp các phương pháp như locator('button:has-text("Submit")') hoặc locators kết hợp nhiều điều kiện, lọc theo trạng thái hiện tại của DOM, giúp locator tự động chờ phần tử sẵn sàng trước khi tương tác. Locator động thường có độ bền và tính ổn định cao hơn locator tĩnh trong test automation hiện đại.
- c. So sánh

Tiêu chí	Locator tĩnh	Locator động
Đặc điểm	Xác định phần tử dựa trên thuộc tính cố định hoặc vị trí không đổi	Xác định phần tử dựa trên điều kiện, trạng thái hoặc kết hợp linh hoạt
Độ ổn định	Dễ bị hỏng nếu DOM hoặc thuộc tính thay đổi	Thích ứng tốt với thay đổi DOM và trạng thái

Tiêu chí	Locator tĩnh	Locator động
Tính linh hoạt	Thấp, cần thay đổi locator nếu UI thay đổi	Cao, locator tự điều chỉnh khi DOM thay đổi
Tính tự động chờ	Thường không có, cần dùng thêm wait	Tự động wait phần tử xuất hiện và sẵn sàng
Độ dễ viết và duy trì	Dễ viết nhưng khó duy trì khi UI thay đổi	Khó viết ban đầu nhưng dễ bảo trì và mở rộng
Ứng dụng phổ biến	Được dùng trong automation cũ hoặc các phần tử ổn định	Dùng phổ biến trong Playwright để giảm lỗi flaky

2.4. Các kỹ thuật định vị locator phổ biến

a. Dựa vào thuộc tính của thẻ HTML

- ID. VD: `driver.findElement(By.id("email"));`
- Name. VD: `driver.findElement(By.name("password"));`
- LinkText. VD: `driver.findElement(By.linkText("Forgot Password?"));`
- PartialLinkText. VD: `driver.findElement(By.partialLinkText("Forgot"));`
- TagName. VD: `driver.findElements(By.tagName("a"))`
- ClassName. VD: `driver.findElement(By.className("btn btn-primary btn-block"));`

b. Xpath

- Định nghĩa: XPath là một ngôn ngữ truy vấn dùng để định vị các phần tử trên trang web bằng cách sử dụng cấu trúc DOM của tài liệu HTML hoặc XML. Nó cho phép xác định chính xác vị trí của một phần tử dựa trên tên thẻ, các thuộc tính hoặc quan hệ giữa các nút trong cây DOM. Trong Selenium, XPath được sử dụng khi các trình định vị cơ bản như id, class hay name không thể xác định duy nhất phần tử. XPath cho phép viết các biểu thức linh hoạt để tìm phần tử dựa trên thuộc tính, văn bản hiển thị, hay vị trí tương đối so với các phần tử khác.

- Có 2 loại Xpath chính:

- Xpath tuyệt đối (Absolute Xpath): bắt đầu từ nút gốc của tài liệu và đi theo một đường cố định đến phần tử cần tìm, dễ bị lỗi khi cấu trúc trang thay đổi. Bắt đầu bằng một dấu '/'.
VD: /html/body/div[1]/form/input[2]

- Xpath tương đối (Relative Xpath): bắt đầu tìm kiếm ở bất kỳ vị trí nào trong DOM, linh hoạt và ít bị ảnh hưởng bởi thay đổi giao diện. Bắt đầu bằng '//'. Cho phép sử dụng các thuộc tính, text và partial matches.
VD: //input[@type='email']

- Cách viết Xpath trong Selenium:

- Syntax: //tagname[@attribute='value']
VD: //div[@aria-labelledby='sign_up_with_google_label']

- Contains(): hữu dụng khi giá trị thuộc tính chỉ biết được 1 phần hoặc thay đổi linh động. Syntax: //tagname[@attribute='value']
VD: //button[contains(@class,'justify-center')]

- Logical Operators (and/or): cho phép kết hợp nhiều điều kiện trong một XPath. Syntax: //tagname[@attr1='value1' or/and @attr2='value2']
VD: //input[@name="email" or contains(@placeholder,'abc')]

- Text(): tìm các phần tử dựa trên những text hiển thị giữa các tags. Syntax: //tagname[text()='Exact Text']
VD: //span[text()='Sign up with Google']

- Starts-with(): sử dụng khi giá trị của thuộc tính bắt đầu với một tiền tố cố định. Syntax: //tagname[starts-with(@attribute,'startValue')]
VD: //input[starts-with(@placeholder,'Desired')]

- Index: sử dụng index khi nhiều phần tử khớp với một Xpath, muốn chọn một phần tử cụ thể. Syntax:
(//tagname[contains(@attribute,'value')])[index]
VD: (//input[contains(@class,'customPlaceholder')])[2]

- Chained XPath in Selenium: giúp di chuyển qua các phần tử lồng nhau.
Syntax: `//parentTag[@attribute='value']//childTag[@attribute='value']`

VD: `//div[contains(@class,'signUpWithEmail')]/input[@id='email']`

- Xpath axes: là các phương thức trong XPath dùng để định vị các phần tử dựa trên mối quan hệ cây DOM giữa phần tử hiện tại và các phần tử khác trong tài liệu XML hoặc HTML. Các axes định nghĩa một hướng cụ thể để duyệt qua các phần tử như tổ tiên, cha mẹ, con, anh chị em, các phần tử trước hoặc sau phần tử hiện tại. Cách viết Xpath axes phổ biến:

- Following: tìm ra tất cả các phần tử theo sau node hiện tại. Syntax:
`//tagname[@attribute='value']//following::tagname`

VD: `//form[contains(@class,'form')]/following::div/input[@id='userpassword']`

- Following-sibling: siblings là những node cùng chung 1 node cha hoặc ở cùng cấp. Do đó, cách này sẽ trả về node phía sau node hiện tại nhưng ở cùng cấp. VD: `//tagname[@attribute='value']//following-sibling::tagname`

VD: `(//div[contains(@class,'custom__border')]/following-sibling::div)[1]`

- Preceding: định vị các phần tử trước node hiện tại bất kể cấp bậc. Syntax:
`//tagname[@attribute='value']//preceding::tagname.`

VD: `//button[@data-testid='signup-button']//preceding::input[@type='password']`

- Child: được sử dụng để định vị các phần tử con của một node cụ thể. Một trường hợp sử dụng phổ biến cho các tiếp cận này là để lặp qua dữ liệu trong table bằng cách di chuyển qua các hàng. Syntax:
`//tagname[@attribute='value']//child::tagname`

VD: `//div[@aria-labelledby='sign_up_with_google_label']/child::span`

- Parent: được sử dụng để chọn node cha của node hiện tại trong XML hoặc HTML document. Syntax: `//div[@aria-labelledby='sign_up_with_google_label']/parent::div`

VD: `//button[@data-testid='signup-button']/parent::div/preceding-sibling::div/input[@type='password'].`

- Decendant: được sử dụng để chọn tất cả phần tử hậu duệ trong XML và HTML document. Gồm các con, cháu từ nhiều cấp khác nhau trong cây DOM. Syntax: `//tagname[@attribute='value']//descendant::tagname`

VD: `//div[contains(@class,'overflow-hidden')]/descendant::span`

- Ancestor: được sử dụng để chọn các tổ tiên của node hiện tại (từ nhiều cấp khác nhau). Syntax: `//tagname[@attribute='value']/ancestors::tagname`

VD: `//input[@type='password']/ancestor::div//input[@id='email']`

- Best practices:

- Ưu tiên dùng Relative XPath hơn Absolute XPath: Sử dụng XPath tương đối (bắt đầu bằng `//`) thay vì đường dẫn tuyệt đối (`/html/body/...`) để giúp các locator vững chắc hơn khi giao diện thay đổi.
- Sử dụng thuộc tính độc nhất: Hướng tới các thuộc tính như id, name hoặc các thuộc tính tùy chỉnh data-* khi có thể để tạo locator ổn định và chính xác.
- Dùng hàm `contains()` để khớp một phần: Khi xử lý các thuộc tính động, dùng hàm `contains()` để tìm kiếm dựa trên giá trị thuộc tính chứa một phần chuỗi.
- Dùng hàm `text()` để khớp văn bản hiển thị: Nếu văn bản hiển thị của phần tử là duy nhất và nhất quán, dùng hàm `text()` để xác định phần tử dựa trên nội dung văn bản.
- Kết hợp nhiều điều kiện: Làm XPath cụ thể hơn bằng cách kết hợp các điều kiện với toán tử `and` hoặc `or`.
- Dùng `starts-with()` cho tiền tố động: Với các thuộc tính bắt đầu bằng một tiền tố cố định, dùng hàm `starts-with()` để tìm kiếm.
- Tránh dùng XPath dựa trên chỉ số khi có thể: Việc dùng chỉ số như `(//div[@class='item'])` dễ bị lỗi khi cấu trúc DOM thay đổi.
- Sử dụng XPath Axes khi cần thiết: Các trục như `following-sibling::`, `parent::`, `ancestor::` giúp tìm phần tử trong các cấu trúc DOM phức tạp.
- Giữ XPath ngắn gọn và dễ đọc: Viết biểu thức ngắn, dễ hiểu và dễ bảo trì. Tránh các đường dẫn quá sâu hoặc phức tạp.
- Kiểm tra XPath bằng DevTools của trình duyệt: Dùng các công cụ như XPath tester hoặc Chrome DevTools với lệnh `$x("your_xpath")` để kiểm tra và xác nhận biểu thức XPath trước khi dùng.

- Tận dụng thuộc tính dữ liệu tùy chỉnh: Ưu tiên dùng các thuộc tính như data-testid, data-role hoặc các thuộc tính data-* khác vì chúng được thiết kế cho việc tự động hóa.
- Viết locator theo ngữ cảnh: Sử dụng các phần tử ổn định gần khu vực, như nhãn (label), làm điểm neo giúp XPath chính xác và đáng tin cậy hơn.

c. CSS selector

- Định nghĩa: CSS selector là một phương pháp được sử dụng trong Selenium để xác định các phần tử web dựa trên id, class, name, thuộc tính và các đặc điểm khác của phần tử HTML. CSS selector được ưa chuộng hơn vì cú pháp đơn giản, dễ viết và tốc độ xử lý nhanh hơn Xpath.
- Phương thức trong Selenium để sử dụng CSS selector là By.cssSelector(String cssSelector) nhận vào một chuỗi CSS Selector để xác định phần tử cần thao tác. Có 5 loại CSS Selectors trong test Selenium: ID, class, attribute, sub-string, inner string.

VD: `driver.findElement(By.cssSelector("#offers"));`

`driver.findElement(By.cssSelector("a[id='offers']"));`

`driver.findElement(By.cssSelector("a[href='/favourites']"));`

- ID selector: dùng dấu #
 - `driver.findElement(By.cssSelector("tagname[id = 'id_value']"))`
 - `driver.findElement(By.cssSelector("#id"))`
- Class selector: dùng dấu .
 - `driver.findElement(By.cssSelector("tagname[class = 'class_value']"))`
 - `driver.findElement(By.cssSelector(".class"))`
- Attribute
 - `driver.findElement(By.cssSelector("<tagname>[href='<href value>']"));`
- Combining attributes:
 - `driver.findElement(By.cssSelector("<tagname>#<id value>[href='<href value>']"));`
 - `driver.findElement(By.cssSelector("<tagname>.<class value>[href='<href value>']"));`
- Substring: CSS Selector cho phép khớp một phần của giá trị thuộc tính với các ký hiệu
 - ^= bắt đầu với (starts with): `a[class^='Navbar_logo_']`
 - \$= kết thúc với (ends with): `a[class$='26S5Y']`
 - *= chứa chuỗi con (contains): `a[class*='logo_']`

d. So sánh Xpath và CSS Selector

Tiêu chí	CSS Selector	XPath
Cú pháp	Đơn giản, ngắn gọn, dễ đọc	Cú pháp dài và phức tạp hơn
Hiệu suất	Nhanh hơn, hoạt động tốt trên hầu hết trình duyệt	Chậm hơn so với CSS, đặc biệt trên tài liệu lớn
Khả năng truy cập DOM	Giới hạn truy cập, không hỗ trợ truy cập các phần tử cha hoặc tổ tiên	Có thể truy cập bất kỳ nút nào trong cây DOM, bao gồm cha, tổ tiên, anh chị em
Hỗ trợ truy vấn phức tạp	Hạn chế, không thể thực hiện các điều kiện phức tạp hoặc tìm phần tử theo vị trí tương đối	Có thể sử dụng các biểu thức phức tạp, điều kiện logic và vị trí tương đối
Truy cập phần tử bằng text	Không hỗ trợ chọn phần tử dựa trên text	Hỗ trợ trực tiếp với hàm text() để chọn phần tử dựa trên nội dung văn bản
Ứng dụng trong Automation Testing	Tốt cho các phần tử có id, class hoặc thuộc tính cố định	Tốt cho việc truy cập các phần tử động, phức tạp hoặc sâu trong DOM
Áp dụng thực tế	Ưu tiên dùng khi cần hiệu suất nhanh, cú pháp đơn giản, và phần tử được xác định rõ ràng bởi id, class hoặc các thuộc tính.	Phù hợp với các tình huống cần truy cập cấu trúc DOM phức tạp, các phần tử không có định danh duy nhất, hoặc cần dùng các biểu thức logic/phức tạp để định vị

3. TÌM HIỂU VỀ CÁCH CONFIG PROJECT

3.1. Các configuration cơ bản của playwright.config.ts

Option	Description
<u>testConfig.forbidOnly</u>	Khi forbidOnly được bật (true), nếu trong mã nguồn còn test nào sử dụng test.only (có nghĩa là chỉ chạy test đó mà bỏ qua test khác), Playwright sẽ báo lỗi và thoát khi chạy test, giúp tránh việc

Option	Description
	vô tình chỉ chạy một vài test mà bỏ sót phần lớn các test khác.
<u>testConfig.fullyParallel</u>	Điều khiển cách chạy test song song. Mặc định, các file test chạy song song, nhưng các test trong cùng 1 file chạy lần lượt. Nếu bật fullyParallel : true thì mọi test trong tất cả các file chạy đồng thời, tăng tốc độ nhưng cần đảm bảo test không phụ thuộc nhau.
<u>testConfig.projects</u>	Cho phép định nghĩa nhiều project để test cùng lúc nhiều môi trường, trình duyệt hoặc cấu hình khác nhau. Ví dụ: 1 project cho Chrome Desktop, 1 cho FireFox mobile
<u>testConfig.reporter</u>	Cấu hình định dạng báo cáo kết quả test. Có thể dùng các reporter tích hợp sẵn như list, line, json, html ...
<u>testConfig.retries</u>	Số lần thử lại các failed test, hữu ích để giảm ảnh hưởng của test flaky, đặc biệt trên CI.
<u>testConfig.testDir</u>	Đường dẫn tới thư mục chứa các file test, mặc định là ./tests, giúp playwright biết nơi tìm test để chạy.

Option	Description
testConfig.use	<p>Cấu hình chung cho các test, chia sẻ cho tất cả projects hoặc test files, bao gồm các setting như trình duyệt, viewport, baseURL, headless, timeout, video, trace ...</p> <p>VD: use: {</p> <pre> browserName: 'chromium', headless: true, baseURL: 'https://example.com', trace: 'on-first-retry', video: 'retain-on-failure' } </pre>
testConfig.webServer	<p>Cấu hình để tự động start một web server trước khi test chạy. Ví dụ khi cần chạy app local server:</p> <ul style="list-style-type: none"> - Command: lệnh start server như npm run start - Port: cổng server chạy. <p>Playwright sẽ chờ server sẵn sàng rồi mới chạy test.</p>
testConfig.workers	<p>Số lượng tiến trình chạy song song. Mặc định playwright sẽ tự đặt dựa trên số lõi CPU. Có thể giới hạn để tránh quá tải tài nguyên hoặc khi chạy debug.</p>

Option	Description
<u>testConfig.testIgnore</u>	<p>Đây là cấu hình dùng để loại trừ các file hoặc thư mục khỏi quá trình tìm kiếm và chạy test</p> <p>VD: testIgnore: ['**/helpers/**', '**/*.skip.ts'],</p>
<u>testConfig.testMatch</u>	<p>Ngược lại với testIgnore, thuộc tính này dùng để chỉ định cụ thể các file test cần chạy bằng pattern.</p> <p>VD: testMatch: ['**/*.e2e.ts']</p>
<u>testConfig.globalSetup</u>	<p>Đường dẫn tới file global setup. File này bắt buộc và được chạy trước tất cả các test. Thường dùng để chuẩn bị môi trường, cần phải export một hàm asynce nhận đối tượng cấu hình FullConfig.</p>
<u>testConfig.globalTeardown</u>	<p>Đường dẫn tới file chứa hàm chạy sau khi chạy hết tất cả test, có tác dụng dọn dẹp tài nguyên như đóng kết nối DB, xóa dữ liệu tạm.</p>
<u>testConfig.outputDir</u>	<p>Là thư mục để playwright lưu tất cả các file tạo ra trong quá trình chạy test, bao gồm các file log, báo cáo, trace file, screenshot, video ... Mặc định nó nằm ở ./test-results. Thư mục này sẽ được xóa sạch mỗi lần chạy test mới.</p>

Option	Description
<u>testConfig.timeout</u>	Thiết lập timeout mặc định tính bằng milisec cho mỗi test. Nếu test hoặc step nào chạy quá thời gian này sẽ tự động dừng. Mặc định là 30000ms (30s)
<u>testConfig.expect</u>	<p>Cấu hình tùy chọn cho thư viện expect dùng trong playwright để viết các câu lệnh assert trong test.</p> <p>Có thể cấu hình timeout mặc định cho các matcher async của expect, cấu hình cụ thể cho snapshot so sánh như toMatchSnapshot hoặc ảnh chụp màn hình toHaveScreenShot.</p> <pre> expect: { timeout: 10000, // 10 giây toMatchSnapshot: { maxDiffPixels: 10, // cho phép tối đa 10 điểm ảnh khác biệt threshold: 0.2, // độ nhạy màu sắc }, toHaveScreenshot: { animations: 'disabled', // tắt animation khi chụp caret: 'hide', // ẩn caret khi chụp } } </pre>

Option	Description
	maxDiffPixelRatio: 0.01 // tỷ lệ điểm ảnh khác tối đa }},

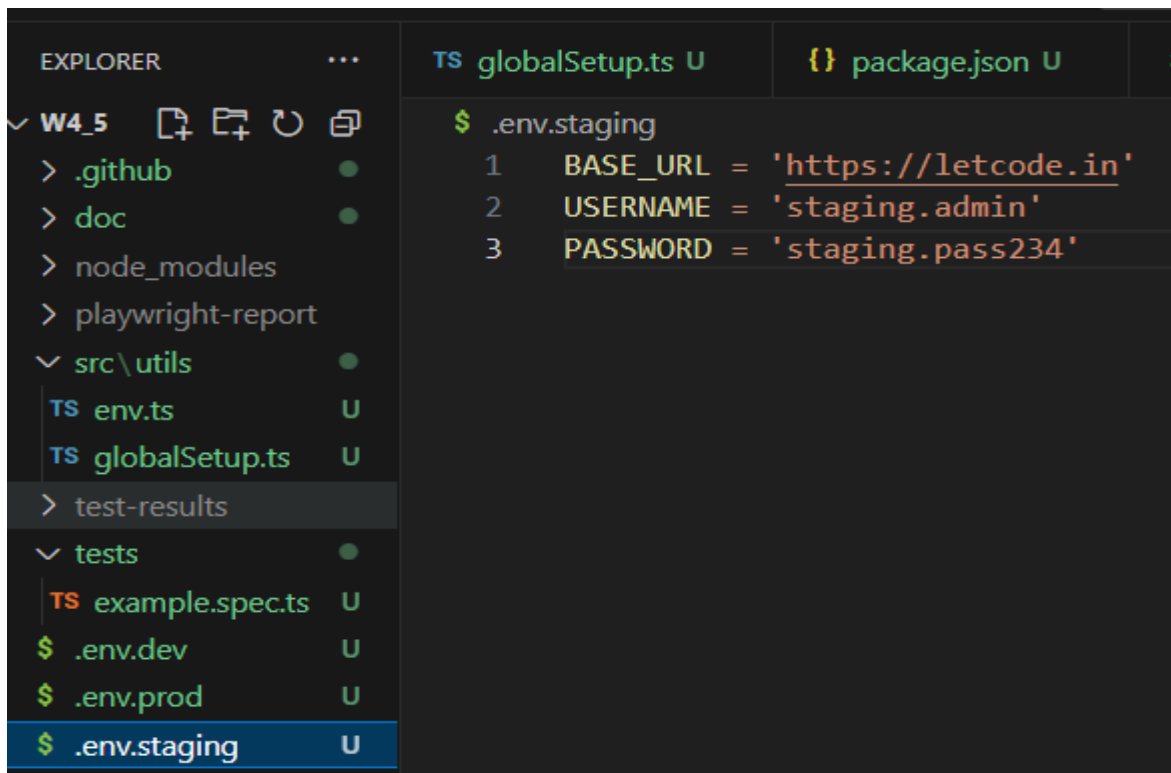
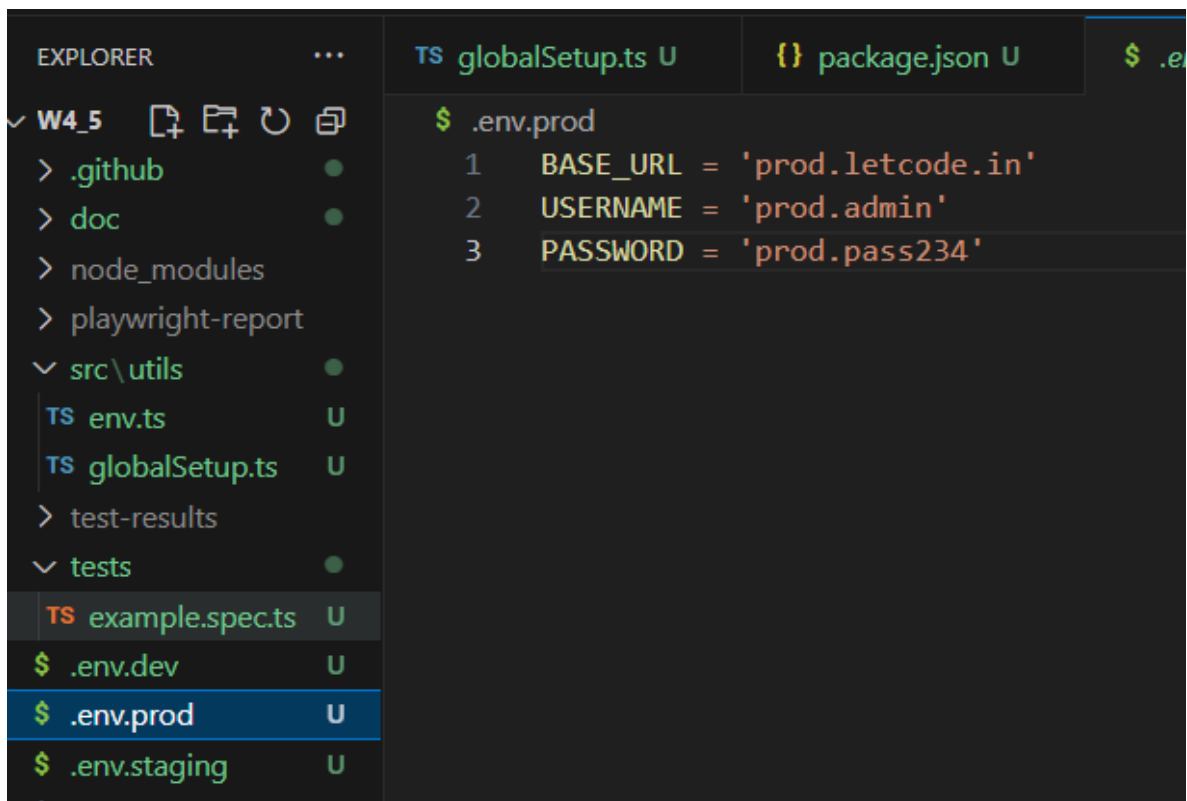
3.2. Cách config multi-env

- Dùng lệnh : npm install --save-dev cross-env để cài đặt và sử dụng các biến môi trường xuyên suốt hệ điều hành
- Tạo các file env cho từng môi trường : dev, staging, production.

```

$ .env.dev
1  BASE_URL = 'dev.letcode.in'
2  USERNAME = 'dev.admin'
3  PASSWORD = 'dev.pass234'

```



- Trong thư mục src/Utils, tạo file globalSetup.ts để chọn ra từng file môi trường để thực thi

```

src > utils > TS globalSetup.ts > globalSetup
1  import { FullConfig } from "@playwright/test";
2
3  import dotenv from "dotenv"
4
5
6  async function globalSetup(config: FullConfig) {
7
8      if (process.env.test_env) {
9          dotenv.config({
10             path: `./.env.${process.env.test_env}`,
11             override: true
12          })
13      }
14  }
15  export default globalSetup;

```

- Trong file env.ts, định nghĩa ra class ENV để tập trung quyền truy cập tới các biến môi trường

```

EXPLORER
W4_5
> .github
> doc
> node_modules
> playwright-report
src\utils
TS env.ts

src > utils > TS env.ts > ENV
1  export default class ENV{
2      public static BASE_URL = process.env.BASE_URL
3      public static USERNAME = process.env.USERNAME
4      public static PASSWORD = process.env.PASSWORD
5  }

```

- Định nghĩa ra các script để chạy npm ngắn gọn hơn ở trong package.json.

```

scripts: {
  "env:staging": "cross-env test_env=staging npx playwright test --project chromium",
  "env:prod": "cross-env test_env=prod npx playwright test --project chromium",
  "env:dev": "cross-env test_env=dev npx playwright test --project chromium"
}

```


- Cấu hình lại file playwright.config.ts bằng cách thêm dòng: globalSetup: "src/utis/globalSetup.ts" tham chiếu đến file globalSetup.ts
- Trong file test, import ENV để tận dụng và truy cập các cấu hình biến môi trường

```

TS globalSetup.ts U    {} package.json U    TS example.spec.ts U X    $ .env.staging U
tests > TS example.spec.ts > test('test') callback
1  import { test, expect } from '@playwright/test';
2  import ENV from '../src/utis/env';
3  test('test', async ({ page }) => {
4      console.log(ENV.BASE_URL);
5      console.log(ENV.USERNAME);
6      console.log(ENV.PASSWORD);
7      await page.goto(ENV.BASE_URL || 'https://letcode.in/');
8  });

```

- Thực thi test bằng lệnh: npm run env:<environment>. Environment có thể là staging, dev hoặc prod.

3.3. Cách dùng trace viewer, debug mode, video record.

a. Trace viewer

- Playwright Trace Viewer là một công cụ giao diện người dùng (GUI) dùng để khám phá các bản ghi (trace) của Playwright sau khi kịch bản test đã chạy xong. Nó rất hữu ích để debug khi test bị lỗi, đặc biệt khi chạy trên CI.
- Tính năng chính:
 - o Hiển thị toàn bộ các hành động Playwright thực hiện trên trang web (click, fill, navigate...).
 - o Ghi lại thời gian thực hiện từng hành động.
 - o Cho phép xem trạng thái DOM trước, trong và sau mỗi hành động.
 - o Hiển thị log console, network requests kèm theo.
 - o Cung cấp ảnh chụp màn hình (screenshots) và đoạn video ngắn (screencast) từng bước để dễ dàng quan sát.
 - o Tích hợp công cụ inspect mã nguồn, selector, trạng thái, giúp phân tích lỗi nhanh chóng.
- Cách bật Trace khi chạy test:
 - o Cấu hình config:


```

use: {
  trace: 'on-first-retry'
}
                    
```

'on-first-retry' - Record a trace only when retrying a test for the first time.

'on-all-retries' - Record traces for all test retries.

'off' - Do not record a trace.

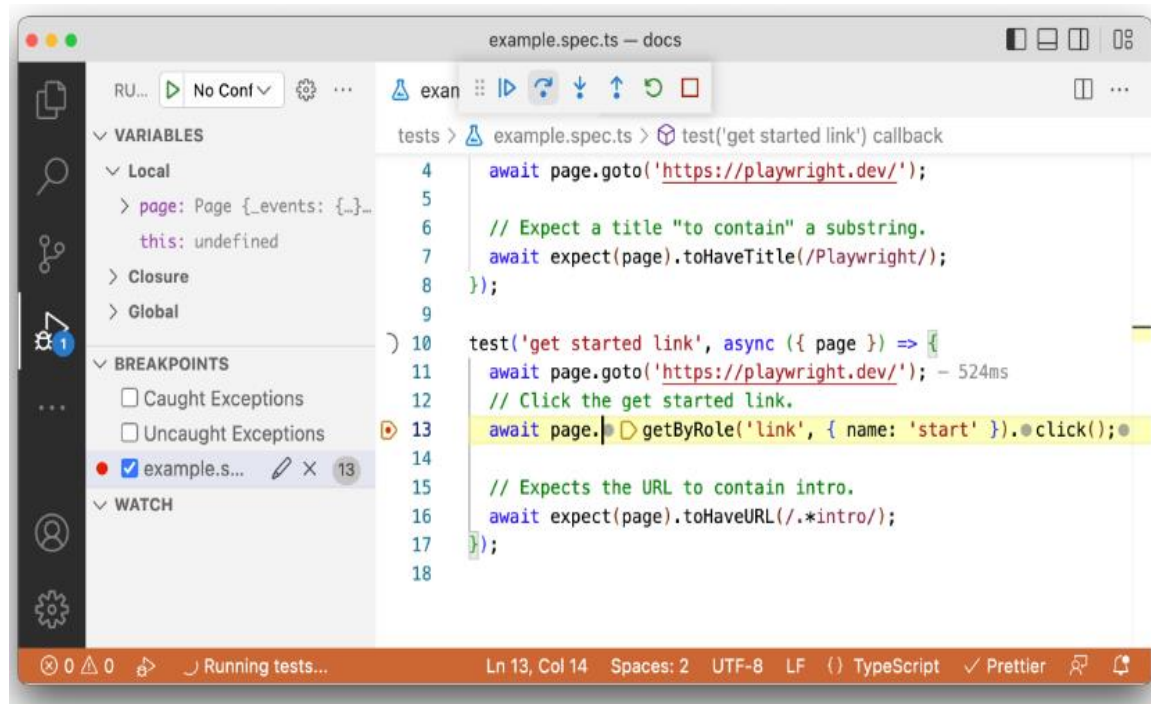
'on' - Record a trace for each test. (not recommended as it's performance heavy)

'retain-on-failure' - Record a trace for each test, but remove it from successful test runs.

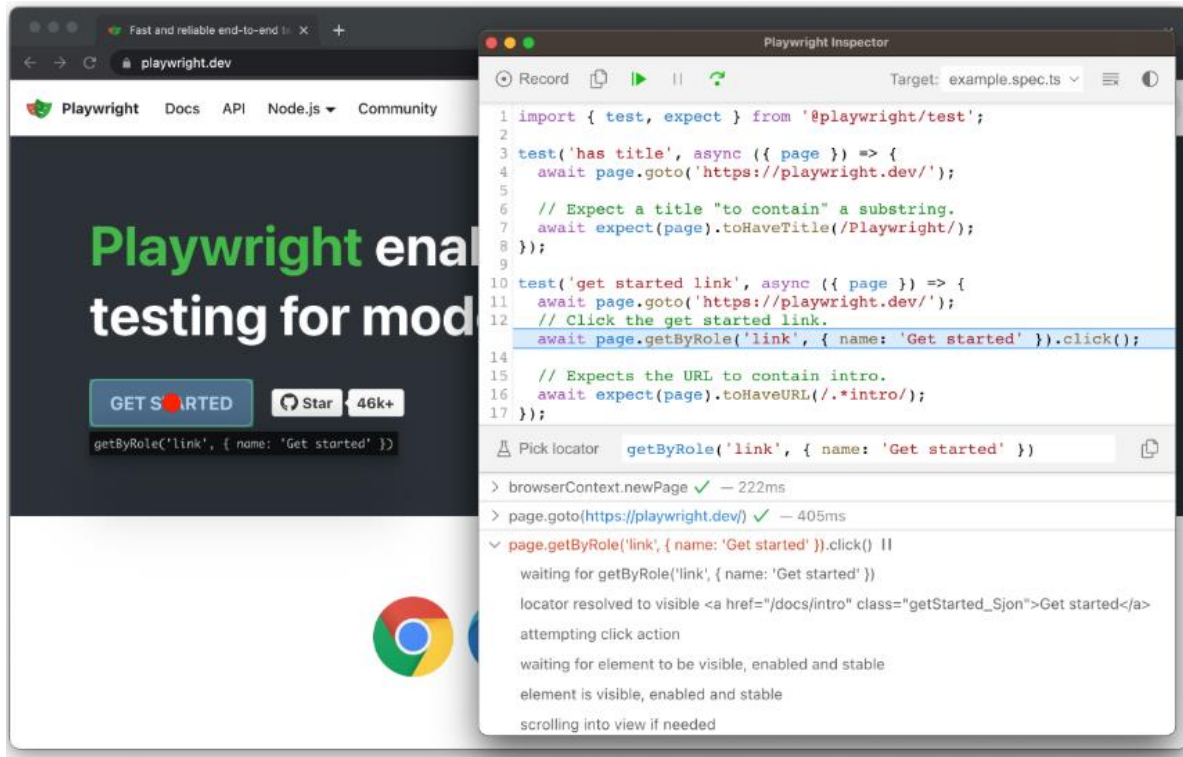
- Chạy test bật trace bằng CLI: `npx playwright test --trace=on`
- Xem trace viewer sau khi test chạy xong bằng lệnh `npx playwright show-trace path/to/trace.zip`

b. Debug mode

- VS code debugger:



- **Playwright inspector:** use `--debug` to open inspector. Or adding `await page.pause()` can also have the same result.



c. Video record

- Playwright có thể quay video cho tests, có thể cấu hình trong file playwright config. Video sẽ lưu lại ở test output directory, mặc định là test-results,

VD: use: {

video: {

mode: 'on-first-retry',

size: { width: 640, height: 480 }

}},

4. TÌM HIỂU VỀ DATA TEST

4.1. Đọc và sử dụng dữ liệu từ JSON

- Tạo file data.json có chứa dữ liệu test dạng mảng các đối tượng, mỗi đối tượng là 1 bộ giá trị.

```
json
[
  {"username": "user1", "password": "password123"},
  {"username": "user2", "password": "password456"}
]
```

- Trong Playwright, dùng Node.js fs để đọc file và JSON.parse để chuyển thành object JavaScript

```
ts
import * as fs from 'fs';

const data = JSON.parse(fs.readFileSync('data.json', 'utf-8'));

for (const user of data) {
  test(`login with ${user.username}`, async ({ page }) => {
    await page.goto('https://example.com/login');
    await page.fill('#username', user.username);
    await page.fill('#password', user.password);
    await page.click('button[type=submit]');
    // Kiểm tra kết quả
  });
}
```

4.2. Đọc và sử dụng dữ liệu từ CSV

- CSV là định dạng bảng rất nhẹ, dễ tạo từ excel hay Google Sheets, phù hợp với dữ liệu đơn giản và nhiều bản ghi
- File CSV mẫu: data.csv

```
text
username,password
user1,password123
user2,password456|
```

- Để đọc CSV trong Playwright (Node.js), sử dụng thư viện csv-parse hoặc papaparse:

```
ts
import * as fs from 'fs';
import { parse } from 'csv-parse/sync';

const csvData = fs.readFileSync('data.csv', 'utf-8');
const records = parse(csvData, { columns: true, skip_empty_lines: true });

for (const row of records) {
  test(`login with ${row.username}`, async ({ page }) => {
    await page.goto('https://example.com/login');
    await page.fill('#username', row.username);
    await page.fill('#password', row.password);
    await page.click('button[type=submit]');
    // Kiểm tra kết quả
  });
}
```

5. REPORT

5.1. HTML

- Playwright có sẵn HTML reporter để có thể tạo ra thư mục chứa báo cáo HTML chi tiết, có thể dùng lệnh CLI : `npx playwright test --reporter=html`
- Ngoài ra, có thể cài đặt cấu hình HTML reporter ở file playwright config như sau:

```
workers: process.env.CI ? 1 : undefined,
/* Reporter to use. See https://playwright.dev/docs/test-reporters */
reporter: [['html', { open: 'always' }],],
/* Shared settings for all the projects below. See https://playwright.d
```

- Để mở report, dùng lệnh CLI: `npx playwright show-report`, hoặc có thể cấu hình giá trị cho `open` : `always` / `on-failure` / `never` để playwright mở report theo nhu cầu

All 1
Passed 1
Failed 0
Flaky 0
Skipped 0

test

example.spec.ts:3 3.3s

chromium

✓ Run

Test Steps

> ✓ Before Hooks
2.0s

> ✓ Navigate to "/" — example.spec.ts:7
2.3s

> ✓ After Hooks
290ms

Screenshots

5.2. ALLURE

- Để cài đặt Allure, thực hiện các bước:
 - o Lệnh CLI: `npm i -D @playwright/test allure-playwright`
 - o Cài đặt Java 8+ và đặt biến môi trường `JAVA_HOME`
 - o Cài đặt allure commandline: `npm install -g allure-commandline`
 - o Cấu hình trong playwright config file:

```

/* Reporter to use. See https://playwright.dev/docs/test-reporters */
reporter: [
  ['html', { open: 'always', outputFolder: 'reports/html-report' }],
  ['allure-playwright', { open: 'always', resultsDir: 'reports/allure-results' }]
],
/* Shared settings for all the projects below. See https://playwright.dev/docs/api/class-test-

```

- Chạy và sinh kết quả bằng lệnh: `npx playwright test --reporter=allure-playwright`, lúc này kết quả của allure sẽ sinh ra ở thư mục root, bên trong thư mục allure-results.
- Để tạo báo cáo HTML từ dữ liệu này thì chạy tiếp lệnh: `allure generate allure-results -o reports/allure-report --clean`, lúc này file report sẽ nằm ở `./reports/allure-report`
- Mở báo cáo bằng lệnh: `allure open reports/allure-report`

