

 **Sonu Madheshiya**

Playwright: Mastering the Essentials - Part 6

Playwright Assertions : A Comprehensive Guide

Prepared By:

Sonu Madheshiya
Automation Tester



<https://www.linkedin.com/in/sonumadheshiya/>

Playwright : Mastering the Essentials - Part 6

Mastering Playwright Assertions: A Comprehensive Guide

Playwright offers powerful assertion capabilities through the expect function, making it easy to validate your tests. To make an assertion, call expect(value) and select a matcher that aligns with your expectation. Generic matchers like toEqual, toContain, and toBeTruthy allow you to assert various conditions effectively.

Example:

```
expect(success).toBeTruthy();
```

Web-Specific Async Matchers

Playwright also provides web-specific async matchers that wait for expected conditions to be met. Consider this example:

```
await expect(page.getByTestId('status')).toHaveText('Submitted');
```

In this case, Playwright will continuously re-fetch the element with the test ID of status and check its text until it matches "Submitted" or the timeout is reached.

By default, assertion timeouts are set to 5 seconds.

Auto-Retrying Assertions

The following assertions will retry until they pass or the timeout is reached. Since these assertions are async, always use await.

Assertion	Description
await expect(locator).toBeAttached()	Element is attached
await expect(locator).toBeChecked()	Checkbox is checked
await expect(locator).toBeDisabled()	Element is disabled
await expect(locator).toBeEditable()	Element is editable
await expect(locator).toBeEmpty()	Container is empty
await expect(locator).toBeEnabled()	Element is enabled



Assertion	Description
<code>await expect(locator).toBeFocused()</code>	Element is focused
<code>await expect(locator).toBeHidden()</code>	Element is not visible
<code>await expect(locator).toBeInViewPort()</code>	Element intersects viewport
<code>await expect(locator).toBeVisible()</code>	Element is visible
<code>await expect(locator).toContainText()</code>	Element contains text
<code>await expect(locator).toHaveAccessibleDescription()</code>	Element has a matching accessible description
<code>await expect(locator).toHaveAccessibleName()</code>	Element has a matching accessible name
<code>await expect(locator).toHaveAttribute()</code>	Element has a DOM attribute
<code>await expect(locator).toHaveClass()</code>	Element has a class property
<code>await expect(locator).toHaveCount()</code>	List has an exact number of children
<code>await expect(locator).toHaveCSS()</code>	Element has a CSS property
<code>await expect(locator).toHaveId()</code>	Element has an ID
<code>await expect(locator).toHaveJSProperty()</code>	Element has a JavaScript property
<code>await expect(locator).toHaveRole()</code>	Element has a specific ARIA role
<code>await expect(locator).toHaveScreenshot()</code>	Element has a screenshot
<code>await expect(locator).toHaveText()</code>	Element matches text
<code>await expect(locator).toHaveValue()</code>	Input has a value
<code>await expect(locator).toHaveValues()</code>	Select has options selected
<code>await expect(page).toHaveScreenshot()</code>	Page has a screenshot
<code>await expect(page).toHaveTitle()</code>	Page has a title
<code>await expect(page).toHaveURL()</code>	Page has a URL
<code>await expect(response).toBeOK()</code>	Response has an OK status

Non-Retrying Assertions

These assertions test conditions without retrying. Be cautious using them in web tests, as they can cause flaky results if the page loads asynchronously.



Prefer auto-retrying assertions, but for complex cases that need retrying, use `expect.poll` or `expect.toPass`.

Assertion	Description
<code>expect(value).toBe()</code>	Value is the same
<code>expect(value).toBeCloseTo()</code>	Number is approximately equal
<code>expect(value).toBeDefined()</code>	Value is not undefined
<code>expect(value).toBeFalsy()</code>	Value is falsy (e.g., false, 0, null)
<code>expect(value).toBeGreaterThan()</code>	Number is more than
<code>expect(value).toBeGreaterThanOrEqual()</code>	Number is more than or equal
<code>expect(value).toBeInstanceOf()</code>	Object is an instance of a class
<code>expect(value).toBeLessThan()</code>	Number is less than
<code>expect(value).toBeLessThanOrEqual()</code>	Number is less than or equal
<code>expect(value).toBeNaN()</code>	Value is NaN
<code>expect(value).toBeNull()</code>	Value is null
<code>expect(value).toBeTruthy()</code>	Value is truthy (e.g., not false, 0, null)
<code>expect(value).toBeUndefined()</code>	Value is undefined
<code>expect(value).toContain()</code>	String contains a substring
<code>expect(value).toContainEqual()</code>	Array or set contains a similar element
<code>expect(value).toEqual()</code>	Value is similar - deep equality and pattern matching
<code>expect(value).toHaveLength()</code>	Array or string has length
<code>expect(value).toHaveProperty()</code>	Object has a property
<code>expect(value).toMatch()</code>	String matches a regular expression
<code>expect(value).toMatchObject()</code>	Object contains specified properties
<code>expect(value).toStrictEqual()</code>	Value is similar, including property types
<code>expect(value).toThrow()</code>	Function throws an error
<code>expect(value).any()</code>	Matches any instance of a class/primitive
<code>expect(value).anything()</code>	Matches anything

Assertion	Description
<code>expect(value).arrayContaining()</code>	Array contains specific elements
<code>expect(value).closeTo()</code>	Number is approximately equal
<code>expect(value).objectContaining()</code>	Object contains specific properties
<code>expect(value).stringContaining()</code>	String contains a substring
<code>expect(value).stringMatching()</code>	String matches a regular expression

Negating Matchers

Invert expectations using `.not` to check the opposite:

```
expect(value).not.toEqual(0);
await expect(locator).not.toContainText('some text');
```

Soft Assertions

Soft assertions don't terminate test execution on failure but mark the test as failed. This allows you to continue testing even if some checks fail.

Example:

```
await expect.soft(page.getByTestId('status')).toHaveText('Success');
await expect.soft(page.getByTestId('eta')).toHaveText('1 day');
```

You can check for soft assertion failures at any point:

```
expect(test.info().errors).toHaveLength(0);
```

Note: Soft assertions only work with the Playwright test runner.

Custom Expect Messages

You can add custom messages to your assertions for better context in reports:

```
await expect(page.getByText('Name'), 'should be logged in').toBeVisible();
```

Custom messages are visible in both passing and failing asserts, making them invaluable for debugging.



<https://www.linkedin.com/in/sonumadheshiya/>

Soft assertions also support custom messages:

```
expect.soft(value, 'my soft assertion').toBe(56);
```

Stay tuned with [Sonu Madheshiya](#) on LinkedIn for more interesting content on automation testing.