Assertions

Introduction

Playwright includes test assertions in the form of `expect` function. To make an assertion, call `expect(value)` and choose a matcher that reflects the expectation. There are many generic matchers like `toEqual`, `toContain`, `toBeTruthy` that can be used to assert any conditions.

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

Playwright also includes web-specific async matchers that will wait until the expected condition is met. Consider the following example:

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

Playwright will be re-testing the element with the test id of `status` until the fetched element has the `"Submitted"` text. It will re-fetch the element and check it over and over, until the condition is met or until the timeout is reached. You can either pass this timeout or configure it once via the testConfig.expect value in the test config.

By default, the timeout for assertions is set to 5 seconds. Learn more about various timeouts.

Auto-retrying assertions

The following assertions will retry until the assertion passes, or the assertion timeout is reached. Note that retrying assertions are async, so you must `await` them.

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
await expect(locator).toBeFocused()	Element is focused
await expect(locator).toBeHidden()	Element is not visible
await expect(locator).toBeInViewport()	Element intersects viewport
await expect(locator).toBeVisible()	Element is visible

Assertion	Description
await expect(locator).toContainText()	Element contains text
await expect(locator).toHaveAccessibleDescription()	Element has a matching accessible description
await expect(locator).toHaveAccessibleName()	Element has a matching accessible name
await expect(locator).toHaveAttribute()	Element has a DOM attribute
await expect(locator).toHaveClass()	Element has a class property
await expect(locator).toHaveCount()	List has exact number of children
await expect(locator).toHaveCSS()	Element has CSS property
await expect(locator).toHaveId()	Element has an ID
await expect(locator).toHaveJSProperty()	Element has a JavaScript property
await expect(locator).toHaveRole()	Element has a specific ARIA role
await expect(locator).toHaveScreenshot()	Element has a screenshot
await expect(locator).toHaveText()	Element matches text
await expect(locator).toHaveValue()	Input has a value
await expect(locator).toHaveValues()	Select has options selected
await expect(page).toHaveScreenshot()	Page has a screenshot
await expect(page).toHaveTitle()	Page has a title
await expect(page).toHaveURL()	Page has a URL
await expect(response).toBeOK()	Response has an OK status

Non-retrying assertions

These assertions allow to test any conditions, but do not auto-retry. Most of the time, web pages show information asynchronously, and using non-retrying assertions can lead to a flaky test.

Prefer auto-retrying assertions whenever possible. For more complex assertions that need to be retried, use `expect.poll` or `expect.toPass`.

Assertion	Description
expect(value).toBe()	Value is the same
expect(value).toBeCloseTo()	Number is approximately equal
expect(value).toBeDefined()	Value is not `undefined`
expect(value).toBeFalsy()	Value is falsy, e.g. `false`, `0`, `null`, etc.
expect(value).toBeGreaterThan()	Number is more than
expect(value).toBeGreaterThanOrEqual()	Number is more than or equal
expect(value).toBeInstanceOf()	Object is an instance of a class
expect(value).toBeLessThan()	Number is less than
expect(value).toBeLessThanOrEqual()	Number is less than or equal
expect(value).toBeNaN()	Value is `NaN`
expect(value).toBeNull()	Value is `null`
expect(value).toBeTruthy()	Value is truthy, i.e. not `false`, `0`, `null`, etc.
expect(value).toBeUndefined()	Value is `undefined`
expect(value).toContain()	String contains a substring

Assertion	Description
expect(value).toContain()	Array or set contains an element
expect(value).toContainEqual()	Array or set contains a similar element
expect(value).toEqual()	Value is similar - deep equality and pattern matching
expect(value).toHaveLength()	Array or string has length
expect(value).toHaveProperty()	Object has a property
expect(value).toMatch()	String matches a regular expression
expect(value).toMatchObject()	Object contains specified properties
expect(value).toStrictEqual()	Value is similar, including property types
expect(value).toThrow()	Function throws an error
expect(value).any()	Matches any instance of a class/primitive
expect(value).anything()	Matches anything
expect(value).arrayContaining()	Array contains specific elements
expect(value).closeTo()	Number is approximately equal
expect(value).objectContaining()	Object contains specific properties
expect(value).stringContaining()	String contains a substring
expect(value).stringMatching()	String matches a regular expression

Negating matchers

In general, we can expect the opposite to be true by adding a `.not` to the front of the matchers:

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

Soft assertions

By default, failed assertion will terminate test execution. Playwright also supports *soft assertions*: failed soft assertions **do not** terminate test execution, but mark the test as failed.

```
// Make a few checks that will not stop the test when failed...
await expect.soft(page.getByTestId('status')).toHaveText('Success');
await expect.soft(page.getByTestId('eta')).toHaveText('1 day');

// ... and continue the test to check more things.
await page.getByRole('link', { name: 'next page' }).click();
await expect.soft(page.getByRole('heading', { name: 'Make another order'
})).toBeVisible();
```

At any point during test execution, you can check whether there were any soft assertion failures:

```
// Make a few checks that will not stop the test when failed...
await expect.soft(page.getByTestId('status')).toHaveText('Success');
await expect.soft(page.getByTestId('eta')).toHaveText('1 day');

// Avoid running further if there were soft assertion failures.
expect(test.info().errors).toHaveLength(0);
```

Note that soft assertions only work with Playwright test runner.

Custom expect message

You can specify a custom expect message as a second argument to the `expect` function, for example:

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

This message will be shown in reporters, both for passing and failing expects, providing more context about the assertion.

When expect passes, you might see a successful step like this:

```
✅ should be logged in    @example.spec.ts:18
```

When expect fails, the error would look like this:

```
    Error: should be logged in

    Call log:
      - expect.toBeVisible with timeout 5000ms
      - waiting for "getByText('Name')"


      2 |
      3 | test('example test', async({ page }) => {
    > 4 |   await expect(page.getByText('Name'), 'should be logged
in').toBeVisible();
        |
^
      5 | });
      6 |
```

Soft assertions also support custom message:

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

expect.configure

You can create your own pre-configured `expect` instance to have its own defaults such as `timeout` and `soft`.

```
const slowExpect = expect.configure({ timeout: 10000 });
await slowExpect(locator).toHaveText('Submit');

// Always do soft assertions.
const softExpect = expect.configure({ soft: true });
await softExpect(locator).toHaveText('Submit');
```

expect.poll

You can convert any synchronous `expect` to an asynchronous polling one using `expect.poll`.

The following method will poll given function until it returns HTTP status 200:

```
await expect.poll(async () => {
  const response = await page.request.get('https://api.example.com');
  return response.status();
}, {
  // Custom expect message for reporting, optional.
  message: 'make sure API eventually succeeds',
  // Poll for 10 seconds; defaults to 5 seconds. Pass 0 to disable timeout.
  timeout: 10000,
}).toBe(200);
```

You can also specify custom polling intervals:

```
await expect.poll(async () => {
  const response = await page.request.get('https://api.example.com');
  return response.status();
}, {
  // Probe, wait 1s, probe, wait 2s, probe, wait 10s, probe, wait 10s,
probe
  // ... Defaults to [100, 250, 500, 1000].
  intervals: [1_000, 2_000, 10_000],
  timeout: 60_000
}).toBe(200);
```

expect.toPass

You can retry blocks of code until they are passing successfully.

```
await expect(async () => {
  const response = await page.request.get('https://api.example.com');
  expect(response.status()).toBe(200);
}).toPass();
```

You can also specify custom timeout and retry intervals:

```
await expect(async () => {
  const response = await page.request.get('https://api.example.com');
  expect(response.status()).toBe(200);
}).toPass({
  // Probe, wait 1s, probe, wait 2s, probe, wait 10s, probe, wait 10s,
probe
  // ... Defaults to [100, 250, 500, 1000].
  intervals: [1_000, 2_000, 10_000],
  timeout: 60_000
});
```

Note that by default `toPass` has timeout 0 and does not respect custom expect timeout.

Add custom matchers using expect.extend

You can extend Playwright assertions by providing custom matchers. These matchers will be available on the `expect` object.

In this example we add a custom `toHaveAmount` function. Custom matcher should return a `message` callback and a `pass` flag indicating whether the assertion passed.

fixtures.ts

```
import { expect as baseExpect } from '@playwright/test';
import type { Page, Locator } from '@playwright/test';

export { test } from '@playwright/test';

export const expect = baseExpect.extend({
  async toHaveAmount(locator: Locator, expected: number, options?: {
timeout?: number }) {
    const assertionName = 'toHaveAmount';
    let pass: boolean;
    let matcherResult: any;
    try {
      await baseExpect(locator).toHaveAttribute('data-amount',
String(expected), options);
      pass = true;
    } catch (e: any) {
      matcherResult = e.matcherResult;
      pass = false;
    }

    const message = pass
      ? () => this.utils.matcherHint(assertionName, undefined, undefined, {
isNot: this.isNot }) +
          '\n\n' +
          `Locator: ${locator}\n` +
          `Expected: ${this.isNot ? 'not' :
''}${this.utils.printExpected(expected)}\n` +
          (matcherResult ? `Received:
${this.utils.printReceived(matcherResult.actual)}` : '')
      : () =>  this.utils.matcherHint(assertionName, undefined, undefined,
{ isNot: this.isNot }) +
          '\n\n' +
          `Locator: ${locator}\n` +
          `Expected: ${this.utils.printExpected(expected)}\n` +
          (matcherResult ? `Received:
${this.utils.printReceived(matcherResult.actual)}` : '');

    return {
      message,
      pass,
      name: assertionName,
      expected,
      actual: matcherResult?.actual,
    };
  },
});
```

Now we can use `toHaveAmount` in the test.

example.spec.ts

```
import { test, expect } from './fixtures';

test('amount', async () => {
  await expect(page.locator('.cart')).toHaveAmount(4);
});
```

Compatibility with expect library

NOTE

Do not confuse Playwright's `expect` with the `expect` library. The latter is not fully integrated with Playwright test runner, so make sure to use Playwright's own `expect`.

Combine custom matchers from multiple modules

You can combine custom matchers from multiple files or modules.

fixtures.ts

```
import { mergeTests, mergeExpects } from '@playwright/test';
import { test as dbTest, expect as dbExpect } from 'database-test-utils';
import { test as a11yTest, expect as a11yExpect } from 'a11y-test-utils';

export const expect = mergeExpects(dbExpect, a11yExpect);
export const test = mergeTests(dbTest, a11yTest);
```

test.spec.ts

```
import { test, expect } from './fixtures';

test('passes', async ({ database }) => {
  await expect(database).toHaveDatabaseUser('admin');
});
```

Previous
Actions

Next
API testing

Learn

- Getting started
- Playwright Training
- Learn Videos
- Feature Videos

Community

- Stack Overflow
- Discord
- Twitter
- LinkedIn

More

- GitHub
- YouTube
- Blog
- Ambassadors