UnCodemy

Playwright Locators Guide PDF

Playwright locators find elements and re-check the page every time you use them. Prefer user-facing locators such as getByRole, getByLabel and getByText, use getByTestId when no accessible name exists, chain and filter to narrow a match, and keep CSS or XPath as a last resort because they break when markup changes.

Key takeaways

- Prefer getByRole, getByLabel and getByText, which mirror how users and assistive tools see the page.
- Use getByTestId with a stable data attribute when no good accessible name exists.
- Locators are lazy and re-queried, which avoids stale element errors.
- Strict mode raises an error if a locator matches more than one element for an action.
- Avoid long CSS or XPath chains tied to layout.

What is Playwright locators?

A Playwright locator is an object that describes how to find an element on the page, is evaluated lazily each time it is used, and works with auto-waiting actions and assertions.

What is a locator in Playwright?

A locator represents a way to find an element. Calling `page.getByRole('button', { name: 'Sign in' })` does not search yet; the search happens when you click, fill or assert. Because the lookup runs again at that moment, a locator keeps working after the page re-renders, which removes a whole class of stale element problems known from older tools.

Which built-in locators should I prefer?

Locator	Use for
`getByRole`	Buttons, links, headings, checkboxes, with accessible name
`getByLabel`	Form fields linked to a label
`getByPlaceholder`	Inputs identified by placeholder text
`getByText`	Visible text in elements
`getByAltText`	Images with alt text
`getByTitle`	Elements with a title attribute

Locator	Use for
`getByTestId`	Elements with a dedicated test attribute

The first choices match what a user perceives, so tests double as a light accessibility check.

How do I write a clear example?

```
import { test, expect } from '@playwright/test';

test('login form', async ({ page }) => {
  await page.goto('https://example.com/login');
  await page.getByLabel('Email').fill('user@example.com');
  await page.getByLabel('Password').fill('secret');
  await page.getByRole('button', { name: 'Sign in' }).click();
  await expect(page.getByRole('heading', { name: 'Dashboard' })).toBeVisible();
});
```

Each line reads like a user story, and no manual wait is needed.

When should I use test ids?

Use `getByTestId('order-total')` when text changes with language or when elements have no accessible name. Ask developers to add a `data-testid` attribute and keep it stable. By default Playwright reads `data-testid`, and you can change the attribute name in configuration. Test ids should be a fallback, not the first tool.

How do I narrow a match?

Chain locators to search inside a parent: `page.getByRole('listitem').filter({ hasText: 'Laptop' }).getByRole('button', { name: 'Add' })`. Use `.filter({ has: locator })` to match containers that hold a child, `.first()`, `.last()` or `.nth(i)` when order is truly meaningful, and `locator.locator('...')` for nested searches. Prefer filtering by content over index, because lists change order.

What is strict mode?

If a locator matches several elements and you call an action such as click, Playwright throws an error. This is deliberate, because it prevents silent clicks on the wrong element. Fix it by making the locator more specific or by using `all()` when you really want a list, for example to count rows with `await expect(rows).toHaveCount(5)`.

What about CSS and XPath?

`page.locator('css=...')` and XPath still work for legacy markup. Keep them short, avoid depending on class names used for styling, and avoid absolute paths. Text selectors inside CSS such as `:has-text()` exist, but built-in locators are clearer. If a CSS locator is the only choice, add a comment saying why.

How can I debug a locator?

Run `npx playwright codegen` to suggest locators while you click, use the inspector with `--debug` to highlight matches, and view traces to see what the page looked like at each step. Treat generated locators as a starting point and improve them for readability and stability.

What are common mistakes?

Using index-based locators on dynamic lists, matching on text that changes by language, depending on layout classes, calling `elementHandle` APIs when locators suffice, and copying long selectors from browser tools. Review locators during code review, the same way you review other code.

How do locators help accessibility?

Role-based locators only work when the page exposes proper roles and names, so a failing getByRole call can reveal a missing label or a clickable div that should be a button. Treat such failures as useful feedback for developers rather than a reason to switch to a fragile selector. Over time, a suite written with user-facing locators encourages better markup, improves screen reader support and keeps tests readable for people who do not know the code base.

How should a team agree on locator rules?

Write a short guideline: use role or label first, text second, test id third and CSS last; never use index unless order is the point; never use classes created for styling. Add a lint or review checklist item, and keep a shared helper for repeated patterns such as table rows or modal dialogs. Revisit the rules after each release so the suite stays consistent as new people join and the application changes.

What is a quick locator checklist?

Before committing a locator, ask whether a user would recognise it, whether it survives a language change, whether it matches exactly one element and whether it reads clearly in a report.

Frequently asked questions

Which Playwright locator should I use first?

Start with getByRole, then getByLabel and getByText, because they match how users see the page and tend to survive markup changes.

What is a lazy locator?

A lazy locator does not find the element when created; it searches each time you act or assert, which keeps it valid after re-renders.

Why do I get a strict mode violation?

The locator matched more than one element for an action, so make it more specific by filtering, chaining or using a role and name.

When should I use getByTestId?

Use it when elements lack a good accessible name or visible text is unstable, and agree a stable data-testid attribute with developers.

Can I still use XPath in Playwright?

Yes, but keep it short and rare, because XPath tied to layout breaks easily and built-in locators are clearer.

Sources and further reading

- Playwright documentation
- Playwright API reference
- Playwright locators guide

About this PDF

These PDFs are study material for learning and revision. They are not a certificate, a degree or a promise of a job, and technology changes, so check official documentation for the latest details. Spot an error? Write to info@uncodemy.com and the trainer team will review it.

More free resources and courses: uncodemy.com | info@uncodemy.com | +91 98183 66550