UnCodemy

Playwright Fixtures Guide PDF

Fixtures give each Playwright test the objects and state it needs, such as page, context, request and your own helpers. Create custom fixtures with test.extend, put setup before the use call and teardown after it, choose test or worker scope, and use them to share login state and page objects without repeating code.

Key takeaways

- Built-in fixtures include page, context, browser, request and browserName.
- Custom fixtures are defined with test.extend and run setup before use and teardown after.
- Fixtures are created only when a test asks for them, which saves time.
- Worker-scoped fixtures run once per worker and suit expensive setup.
- Fixtures replace beforeEach hooks for most shared setup.

What is Playwright fixtures?

A Playwright fixture is a named piece of setup that the test runner creates on demand, passes into tests as an argument and tears down afterwards, keeping tests short and isolated.

What is a fixture?

A fixture is setup code that the runner provides to a test by name. When a test declares `async ({ page, request }) => { ... }`, Playwright builds a page and a request context, runs the test and cleans them up. Fixtures are isolated per test by default, so one test cannot leak cookies or state into another.

Which built-in fixtures exist?

Fixture	What it gives you
`page`	A new page in an isolated context
`context`	The browser context for the test
`browser`	A shared browser instance per worker
`browserName`	Name of the current browser
`request`	An API request context
`baseURL`	The configured base URL
Use only the ones a test needs; unused fixtures are not created.	

How do I write a custom fixture?

```
import { test as base, expect } from '@playwright/test';
import { InventoryPage } from './pages/inventory-page';

type Fixtures = { inventory: InventoryPage };

export const test = base.extend<Fixtures>({
  inventory: async ({ page }, use) => {
    const inventory = new InventoryPage(page);
    await inventory.goto();
    await use(inventory);
    // teardown code goes here, after use()
  },
});
export { expect };
```

Import `test` from this file in specs. Code before `use` is setup; code after it is teardown, and it runs even if the test fails.

When should I use fixtures instead of hooks?

Hooks such as beforeEach run for every test in a file whether needed or not. A fixture runs only when requested, can depend on other fixtures, can be shared across files and keeps setup and cleanup together in one place. Prefer fixtures for page objects, test data and authenticated sessions; keep hooks for small, file-specific steps.

What are scopes?

The default scope is **test**: a fresh instance for every test. A **worker** scope creates the fixture once per worker process, which suits costly work such as starting a mock server or seeding a database. Define it with `[async ({}, use) => { ... }, { scope: 'worker' }]`. Worker fixtures must not hold per-test state.

How do I reuse a login?

Create a setup project or a global step that logs in once, saves the state with `context.storageState({ path: 'auth.json' })` and then set `storageState: 'auth.json'` in the use block. Each test starts already signed in without repeating the form. For several roles, save one file per role and select it through a fixture or project option.

What are auto fixtures and option fixtures?

An **auto fixture** with `{ auto: true }` runs for every test even if not requested, useful for attaching logs or checking console errors. An **option fixture** with `{ option: true }` can be configured from the project settings, such as a default user name or an API URL, making the same tests run against different setups.

How do I combine fixtures?

Fixtures can depend on each other by listing them as arguments. A `loggedInPage` fixture can depend on `page` and a `testUser` fixture. Playwright works out the order. Merge fixture sets from several files with `mergeTests` so teams can own their own pieces.

What mistakes appear most often?

Putting heavy setup in test scope, forgetting the `use` call, doing teardown without error handling, sharing mutable data in a worker fixture and creating a fixture that does too many things. Keep each fixture small, name it for what it provides and document it.

How do I test my fixtures?

Because fixtures are code, give them the same care as tests. Keep them small, add a short comment describing what they provide, and write one tiny test that uses each new fixture to confirm setup and teardown work. If a fixture talks to an API, make failures loud with clear error messages so that a broken environment is easy to tell apart from a broken feature.

How do fixtures fit with page objects?

A common pattern is to create one fixture per page object, so tests receive ready-to-use classes such as loginPage or cartPage. This removes repeated constructors from specs and lets you change how a page is created in one place. Combine them with a fixture for authenticated state, and your tests read like short user stories with almost no setup noise, which makes reviews quicker and onboarding easier for new team members.

What is a simple exercise to learn fixtures?

Write three tests that each log in and open the dashboard, then move the repeated lines into one fixture and see how much code disappears. Add teardown that deletes any data created and run the tests in parallel to confirm isolation. This exercise shows the benefit in about thirty minutes.

Frequently asked questions

What is a fixture in Playwright Test?

It is named setup that the runner creates for a test, passes in as an argument and tears down afterwards, such as page or request.

How do I create a custom fixture?

Use test.extend, put setup before the use call and teardown after it, then import the extended test object in your spec files.

What is the difference between test and worker scope?

Test scope creates the fixture for every test, while worker scope creates it once per worker process for expensive shared setup.

Are fixtures better than beforeEach hooks?

Often yes, because fixtures run only when requested, keep setup and cleanup together and can be shared across files.

How can fixtures help with login?

A fixture or setup project can log in once, save storage state and let tests start already authenticated, saving time on every test.

Sources and further reading

- Playwright documentation
- Playwright API reference
- Playwright fixtures

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