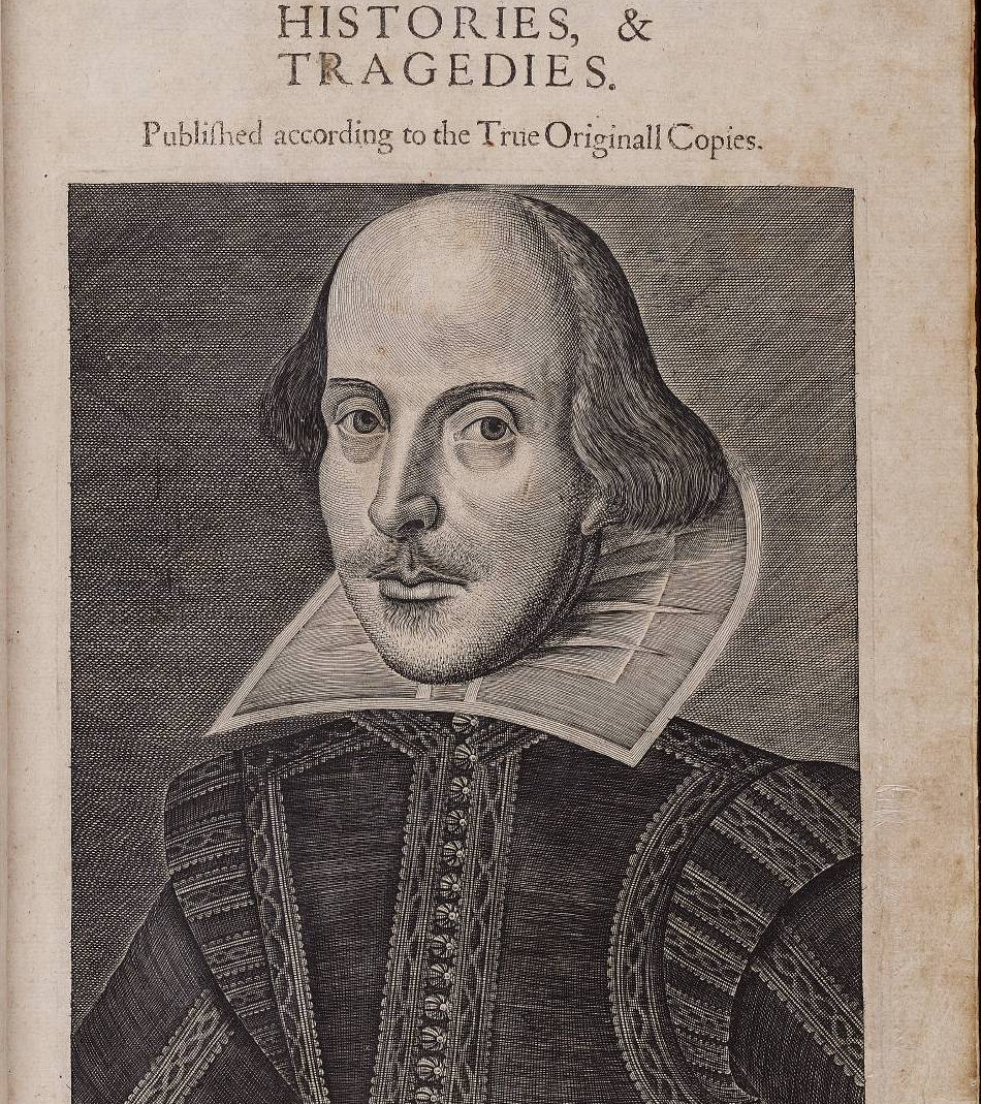
Cleaner Playwright tests

Using page objects, composition, and fixtures

Contents



Question…

> What's in a ~~name~~ page?

–William Shakespeare, at some point, maybe…

A simple page object:

```
// pages/playwright-docs-page.js

import { expect } from "@playwright/test";

export class SomePage {
  constructor(page) {
    this.page = page;
    this.getStarted = page.getByRole(' ... ');
    this.header = page.locator('h1');
    // ...
  }

  async goto() {
    await this.page.goto('https://playwright.dev');
  }

  async getStarted() {
    await this.getStarted.click();
    await expect(this.header).toBeVisible();
  }
  // ...
}
```

Using our page object in a test:

```
// tests/some.test.js

import { test } from "@playwright/test";
import { SomePage } from "...js";

test("getting started", async ({ page }) => {
  const docsPage = new SomePage(page);

  await docsPage.goto();
  await docsPage.getStarted();
  // ...
});
```

Question… So, what are page objects good for?

Answer: Encapsulation & DRY

The page-object model allows us to encapsulate the logic for interacting with a page in a reusable way.

Question… What’s the catch?

Answer: Boilerplate…

Let's say you're writing an E2E test for a complex user flow, requiring interaction with many pages. Pretty soon, it will look like this:

```
// tests/some.test.js

import { test } from "@playwright/test";
import { SomePage } from "...";
import { AnotherPage } from "...";
import { AndAnother } from "...";
import { EtcEtc } from "...";
// ...

test("getting started", async ({ page }) => {
  const somePage = new SomePage(page);
  const anotherPage = new AnotherPage(page);
  const andAnother = new AndAnother(page);
  const etcEtc = new EtcEtc(page);
  // ...
});
```

Enter: Fixtures

Question… First of all, what’s a fixture?

Answer:

> Test fixtures are used to establish environment for each test, giving the test everything it needs and nothing else.

–The Playwright Docs

In plain English, *s'il te plaît?*

Your tests use fixtures in order to have what they need to run.

We've actually already seen them before; remember this?

```
// tests/some.test.js

test("getting started", async ({ page }) => {
  // ...
});
```

The `page` object is a fixture that Playwright provides for us, free of charge.

Question… How does this help?

We can use custom fixtures to relocate much of the boilerplate from the previous example.

Let's create fixtures for our example that had too much boilerplate:

```
// fixtures.js

import { test as base } from "@playwright/test";

import { SomePage } from "...";
import { AnotherPage } from "...";
import { EtcEtc } from "...";
// ...

export const test = base.extend({
    somePage: async ({ page }, use) => {
        const somePage = new SomePage(page);
        await use(somePage);
    },
    anotherPage: async ({ page }, use) => {
        const anotherPage = new AnotherPage(page);
        await use(anotherPage);
    },
    etcEtc: async ({ page }, use) => {
        const etcEtc = new EtcEtc(page);
        await use(etcEtc);
    },
    // ...
});
```

Now, when we want to use multiple pages for a test, things are much cleaner. We turned this:

```
// tests/some.test.js

import { SomePage } from "...";
import { AnotherPage } from "...";
import { EtcEtc } from "...";
// ...

test("getting started", async ({ page }) => {
  const somePage = new SomePage(page);
  const anotherPage = new AnotherPage(page);
  const etcEtc = new EtcEtc(page);
  // ...
});
```

Into this:

```
import { test } from "../fixtures.js";

test("getting started", ({ somePage, anotherPage, etcEtc }) => {
  // ...
});
```

Now, you might be thinking…

"OK, but how is this an improvement? We've move boilerplate from one place to another…"

Well, sort of…

You're right, our `fixtures.js` now has all of the boilerplate that used to be present in our test…

However, this is still an overall improvement; instead of duplicating all of the instantiation code in every test, the nasty repetition is isolated to just one file!

As you can see, there's can still be a lot of boilerplate when we have a fixture for every page.

What if I told you we can do better yet?

How, you might ask? Well, we still have one more trick up our sleeves…

Answer:

Composition

Let's create another class. I like to call these `index` files, but I'm open to a better name if someone has suggestions… 🙂

```
// pages/index.js

import { test } from "@playwright/test";

import { SomePage } from "...";
import { AnotherPage } from "...";
import { AndAnother } from "...";
import { EtcEtc } from "...";
// ...

export class Application {
  constructor(page) {
    this.page = page;

    this.somePage = new SomePage(page)
    this.another = new AnotherPage(page)
    this.andAnother = new AndAnother(page)
    this.etcEtc = new EtcEtc(page)
    // ...
  }
  // ...
}
```

Now, we can dramatically simplify our fixture:

```
// fixtures.js

import { test as base } from "@playwright/test";
import { Application } from "pages/index.js";

export const test = base.extend({
    app: async ({ page }, use) => {
        const app = new Application(page);
        await use(app);
    },
});
```

AND our test!

```
1  // tests/some.test.js
2
3  import { test } from "../fixtures.js";
4
5  test("getting started", ({ app }) => {
6    await app.somePage.doSomething();
7    await app.another.doSomethingElse();
8    // ...
9  });
```

Bonus content!

Sadly, Playwright's fixture mechanism editor breaks completions / suggestions…

However, you can add JSDoc type annotations to fix this!

```
// tests/some.test.js

import { test } from "../fixtures.js";

test("getting started", ({ app: _app }) => {
/** @type {Application} */ const app = _app;

  await app.somePage.doSomething();
  await app.another.doSomethingElse();
  // ...
});

/** @typedef {import("../pages/index.js").Application} Application */
```

That’s a wrap! (Finally! 🤣)

Now, we have simple tests and simple fixtures.

Best of all, we did it by adding a single additional class that collects all of our pages under a single, coherent interface! 🎉 🚀

Credits

This presentation made possible by:

Slidev: an amazing presentation authoring tool

Playwright: some code and quotes were borrowed from the docs

@clmay | /in/chasemay/