Fixtures

Gleiche Grundlage für jeden Test

Einführung

Playwright Test basiert auf dem Konzept der Testfixtures.

Testfixtures werden verwendet, um die Umgebung für jeden Test einzurichten, wobei dem Test alles mitgegeben wird, was er braucht, und nichts anderes.

Testfixtures **`sind zwischen Tests isoliert`**. Mit Fixtures kann man **`Tests auf der Grundlage ihrer Bedeutung gruppieren`** und nicht auf der Grundlage ihres gemeinsamen Aufbaus.

Built-in Fixtures

Hier ein erstes Beispiel:

```
1 // 14_fixtures/tests/01_intro/intro.test.ts
2 import { test, expect } from '@playwright/test';
3
4 test('basic test', async ({ page }) => {
5   await page.goto('https://playwright.dev/');
6
7   await expect(page).toHaveTitle(/Playwright/);
8 });
```

Built-in Fixtures

Das Argument { page } weist Playwright Test an, das Seiten-Fixture einzurichten und an die Testfunktion zu übergeben.

Built-in Fixtures

Fixture	Typ	Beschreibung
page	Page	Isolierte Seite für diesen Testlauf.
context	BrowserContext	Isolierter Kontext für diesen Testlauf. Die Seitenbefestigung gehört ebenfalls zu diesem Kontext. Erfahren Sie, wie Sie den Kontext konfigurieren.
browser	Browser	Die Browser werden für alle Tests gemeinsam genutzt, um die Ressourcen zu optimieren. Erfahren Sie, wie Sie den Browser konfigurieren.
browserName	string	Der Name des Browsers, in dem der Test ausgeführt wird. Entweder Chromium, Firefox oder Webkit.
request	APIRequestContext	Isolated APIRequestContext instance for this test run.

Ohne / Mit Fixtures

```
const { test } = require('@playwright/test');
const { TodoPage } = require('./todo-page');

test.describe('todo tests', () => {
  let todoPage;

  test.beforeEach(async ({ page }) => {
    todoPage = new TodoPage(page);
    await todoPage.goto();
    await todoPage.addToDo('item1');
    await todoPage.addToDo('item2');
  });

  test.afterEach(async () => {
    await todoPage.removeAll();
  });

  test('should add an item', async () => {
    await todoPage.addToDo('my item');
    // ...
  });

  test('should remove an item', async () => {
    await todoPage.remove('item1');
    // ...
  });
});
```

```
import { test as base } from '@playwright/test';
import { TodoPage } from './todo-page';

// Extend basic test by providing a "todoPage" fixture.
const test = base.extend<{ todoPage: TodoPage }>({
  todoPage: async ({ page }, use) => {
    const todoPage = new TodoPage(page);
    await todoPage.goto();
    await todoPage.addToDo('item1');
    await todoPage.addToDo('item2');
    await use(todoPage);
    await todoPage.removeAll();
  },
});

test('should add an item', async ({ todoPage }) => {
  await todoPage.addToDo('my item');
  // ...
});

test('should remove an item', async ({ todoPage }) => {
  await todoPage.remove('item1');
  // ...
});
```

Ohne / Mit Fixtures

Fixtures haben eine Reihe von Vorteilen gegenüber Vorher/Nachher-Hooks:

- Fixtures kapseln Setup und Teardown an der gleichen Stelle, so dass sie einfacher zu schreiben sind.
- Fixtures sind zwischen Testdateien wiederverwendbar - Sie können sie einmal definieren und in allen Ihren Tests verwenden. So funktioniert auch die in Playwright integrierte Seitenfixture.
- Fixtures sind bedarfsorientiert - Sie können beliebig viele Fixtures definieren, und Playwright Test richtet nur die Fixtures ein, die für Ihren Test benötigt werden, und sonst nichts.

Ohne / Mit Fixtures

- Fixtures sind zusammensetzbar - sie können voneinander abhängen, um komplexe Verhaltensweisen zu ermöglichen.
- Fixtures sind flexibel. Tests können beliebige Kombinationen von Fixtures verwenden, um die von ihnen benötigte Umgebung genau anzupassen, ohne andere Tests zu beeinträchtigen.
- Fixtures vereinfachen die Gruppierung. Sie müssen Tests nicht mehr in Beschreibungen einwickeln, die die Umgebung festlegen, sondern können Ihre Tests stattdessen nach ihrer Bedeutung gruppieren.

Eigenes Fixture

Um Ihre eigene Fixture zu erstellen, verwendet man test.extend(), um ein neues Testobjekt zu erstellen, das die Fixture enthält.

Eigenes Fixture

```ts
// 14_fixtures/tests/02_own_fixture/fix.ts
import { test as base } from '@playwright/test';
import { TodoPage } from './todo-page';
import { SettingsPage } from './settings-page';

// Declare the types of your fixtures.
type MyFixtures = {
  todoPage: TodoPage;
  settingsPage: SettingsPage;
};

// Extend base test by providing "todoPage" and "settingsPage".
// This new "test" can be used in multiple test files, and each of them will get the fixtures.
export const test = base.extend<MyFixtures>({
  todoPage: async ({ page }, use) => {
    // Set up the fixture.
    const todoPage = new TodoPage(page);
    await todoPage.goto();
    await todoPage.addToDo('item1');
    await todoPage.addToDo('item2');

    // Use the fixture value in the test.
    await use(todoPage);

    // Clean up the fixture.
    await todoPage.removeAll();
  },

  settingsPage: async ({ page }, use) => {
    await use(new SettingsPage(page));
  },
});
export { expect } from '@playwright/test';
```

Eigenes Fixture

Nun können wir ganz einfach das Fixture verwenden todoPage und settingsPage.

```
1  // 14_fixtures/tests/02_own_fixture/fix.test.ts
2  import { test, expect } from './fix.ts';
3
4  test.beforeEach(async ({ settingsPage }) => {
5    await settingsPage.switchToDarkMode();
6  });
7
8  test('basic test', async ({ todoPage, page }) => {
9    await todoPage.addToDo('something nice');
10   await expect(page.getByTestId('todo-title')).toContainText(['something nice']);
11 });
```

Ende

Das war alles für dieses Kapitel