2.Parameterize

**Introduction**

Parameterization in tests is a powerful way to run the same test logic with different input data. Playwright supports parameterizing tests either at a **test level** or at a **project level**.

**Parameterized Tests**

One way to parameterize tests in Playwright is by iterating over an array of inputs. Each iteration runs the test with different values.

**Example**

ts

Copy code

[

{ name: 'Alice', expected: 'Hello, Alice!' },

{ name: 'Bob', expected: 'Hello, Bob!' },

{ name: 'Charlie', expected: 'Hello, Charlie!' },

].forEach(({ name, expected }) => {

test(`testing with ${name}`, async ({ page }) => {

await page.goto(`https://example.com/greet?name=${name}`);

await expect(page.getByRole('heading')).toHaveText(expected);

});

});

**Explanation**

In the above example:

* An array of objects containing name and expected values is iterated over.
* For each object, a test is created dynamically with the input values.
* The test function runs each test with a unique name (like testing with Alice) and verifies the text on the page using the expected value.

**Before and After Hooks**

**Efficient Use of Hooks**

Most of the time, it's better to place beforeEach, afterEach, beforeAll, and afterAll hooks **outside** the loop that iterates over test data. This way, the hooks are executed only once, instead of running for each test iteration.

**Example**

ts

Copy code

test.beforeEach(async ({ page }) => {

// Setup code before each test.

});

test.afterEach(async ({ page }) => {

// Cleanup code after each test.

});

[

{ name: 'Alice', expected: 'Hello, Alice!' },

{ name: 'Bob', expected: 'Hello, Bob!' },

{ name: 'Charlie', expected: 'Hello, Charlie!' },

].forEach(({ name, expected }) => {

test(`testing with ${name}`, async ({ page }) => {

await page.goto(`https://example.com/greet?name=${name}`);

await expect(page.getByRole('heading')).toHaveText(expected);

});

});

**Placing Hooks Inside describe()**

If you need hooks to run for each test individually, you can place them inside test.describe(). This ensures the hooks are executed for each iteration.

**Example**

ts

Copy code

[

{ name: 'Alice', expected: 'Hello, Alice!' },

{ name: 'Bob', expected: 'Hello, Bob!' },

{ name: 'Charlie', expected: 'Hello, Charlie!' },

].forEach(({ name, expected }) => {

test.describe(() => {

test.beforeEach(async ({ page }) => {

await page.goto(`https://example.com/greet?name=${name}`);

});

test(`testing with ${expected}`, async ({ page }) => {

await expect(page.getByRole('heading')).toHaveText(expected);

});

});

});

**Parameterized Projects**

Playwright also supports **running multiple test projects** simultaneously, each with different configurations.

**Example**

In the following example, we declare the option person and set different values for two projects (Alice and Bob).

**Configuration**

js

Copy code

const base = require('@playwright/test');

exports.test = base.test.extend({

person: ['John', { option: true }],

});

We use this option in the test and configure multiple projects.

ts

Copy code

import { test } from './my-test';

test('test 1', async ({ page, person }) => {

await page.goto(`/index.html`);

await expect(page.locator('#node')).toContainText(person);

});

**Running Multiple Projects**

Define the test projects in the Playwright config:

ts

Copy code

module.exports = defineConfig({

projects: [

{

name: 'alice',

use: { person: 'Alice' },

},

{

name: 'bob',

use: { person: 'Bob' },

},

]

});

Each project runs the same test with a different person value.

**Passing Environment Variables**

Playwright allows configuring tests through environment variables, which is useful for passing secrets or changing settings based on the environment.

**Example**

ts

Copy code

test(`example test`, async ({ page }) => {

await page.getByLabel('User Name').fill(process.env.USER\_NAME);

await page.getByLabel('Password').fill(process.env.PASSWORD);

});

Run the test using environment variables:

bash

Copy code

USER\_NAME=me PASSWORD=secret npx playwright test

**Reading Environment Variables in Config**

ts

Copy code

import { defineConfig } from '@playwright/test';

export default defineConfig({

use: {

baseURL: process.env.STAGING === '1' ? 'http://staging.example.test/' : 'http://example.test/',

}

});

Run tests in different environments:

bash

Copy code

STAGING=1 npx playwright test

**Using .env Files**

To manage environment variables more easily, use .env files in combination with the dotenv package.

**Example**

ts

Copy code

import dotenv from 'dotenv';

import path from 'path';

dotenv.config({ path: path.resolve(\_\_dirname, '.env') });

export default defineConfig({

use: {

baseURL: process.env.BASE\_URL,

}

});

The .env file:

makefile

Copy code

STAGING=0

USER\_NAME=me

PASSWORD=secret

**Creating Tests via CSV**

Playwright supports reading CSV files and generating tests dynamically. For example, given the following CSV file input.csv:

arduino

Copy code

"test\_case","some\_value","some\_other\_value"

"value 1","value 11","foobar1"

"value 2","value 22","foobar21"

"value 3","value 33","foobar321"

**Example**

ts

Copy code

import fs from 'fs';

import path from 'path';

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

import { parse } from 'csv-parse/sync';

const records = parse(fs.readFileSync(path.join(\_\_dirname, 'input.csv')), {

columns: true,

skip\_empty\_lines: true

});

for (const record of records) {

test(`foo: ${record.test\_case}`, async ({ page }) => {

console.log(record.test\_case, record.some\_value, record.some\_other\_value);

});

}