Grouping, Hooks, Annotations & Tags

Grouping Tests

test.describe()

- **Purpose**: Group related tests together.
- **Use Case**: Organize tests by feature, page, or functionality.

grouping.spec.ts:

```
test.describe('Group1', () => {
        test('test1', async ({ page }) => {
          // test logic
        });
        test(' test2', async ({ page }) => {
          // test logic
        });
       test(' test3', async ({ page }) => {
          // test logic
        });
 });
```

To run all the tests from all the groups.

```
npx playwright test tests/grouping.spec.ts
```

To run specific group of tests.

```
npx playwright test tests/grouping.spec.ts --grep Group1
```

Playwright Hooks

Playwright provides several **hooks** to manage how and when tests are executed.

These help in **setting up test environments**, **cleaning up after tests**, and **organizing or skipping specific tests**.

test.beforeAll()

- **Purpose**: Runs **once before all tests** in a file or a describe block.
- **Use Case**: Initialize shared resources like launching a browser or setting up test data.

```
beforeAll(async () => {
  console.log('Runs once before all tests');
});
```

test.afterAll()

- **Purpose**: Runs **once after all tests** in a file or a describe block.
- **Use Case**: Clean up resources like closing the browser or deleting test data.

```
afterAll(async () => {
  console.log('Runs once after all tests');
});
```

test.beforeEach()

- **Purpose**: Runs **before each individual test**.
- **Use Case**: Set up a fresh state like opening a new page or logging into an app.

```
beforeEach(async ({ page }) => {
  await page.goto('https://example.com');
});
```

test.afterEach()

- **Purpose**: Runs **after each individual test**.
- **Use Case**: Clean up after each test like closing the page or clearing local storage.

```
afterEach(async ({ page }) => {
  await page.close();
});
```

Annotations

Playwright provides **annotations** to control how and when tests run. These are helpful during development, debugging, and when working with flaky or incomplete tests.

Below are commonly used annotations:

test.only(...)

Purpose:
Runs **only** this test and skips all others. Useful for debugging a specific test.

Syntax:

```
test.only('My focused test', async ({ page }) => {
  // test steps
});
```

Use case:
When you want to **focus** on just one test temporarily without running the entire suite.

test.skip(...)

Purpose:
Skips the test entirely. The test will not run, and Playwright will show it as **skipped**.

Syntax:

```
test.skip('This test is not ready yet', async ({ page }) => {
  // this won't be executed
});
```

Use case:
When a test is incomplete or not relevant for the current test run.

test.skip()

Purpose:
This test is skipped intentionally.

Syntax:

```
Test.skip('My test', async ({ page }) => {
    // test steps
});
```

Use case:
To avoid failures in environments where a feature isn't supported.

test.fail(...)

Purpose:
Marks a test as **expected to fail**. If it passes, Playwright will highlight it as unexpected success.

Syntax:

```
test.fail('This is a known failing test', async ({ page }) => {
  // failing test steps
});
```

Use case:
When you're tracking a **known issue**, but don't want it to break the test suite.

test.fixme(...)

Purpose:
Marks a test that needs to be **fixed**. The test is automatically skipped.

Syntax:

```
test.fixme('Needs investigation for failure on Safari', async ({ page }) => {
  // test steps
});
```

Use case:
Great for developers to mark broken or incomplete tests for future fixing.

test.slow(...)

Purpose:
Increases the timeout for this test. Useful when you know a test takes longer than usual.

Syntax:

```
test('Long-running test', async ({ page }) => {
test.slow();
  // slow actions
});
```

Use case:
For tests with heavy computation, slow network requests, or large file downloads.

Summary:

- **beforeAll** – Use this for expensive setup done once (e.g., DB setup, API auth, etc.).
- **afterAll** – Clean up any global setup (e.g., close DB, reset state).
- **beforeEach** – Setup before every test (e.g., navigate to the test page).
- **afterEach** – Cleanup after every test (e.g., clear cookies, logout).
- **only** - Focus mode. Runs only this test and Skips all other tests.
- **skip** - Skip this test (Does not run).
- **describe**- Group related tests.
- **fail** - Marks test as expected to fail
- **fixme** - Marks test to fix; automatically skips
- **slow** - Extends timeout triple.

Tags

You can tag your tests (e.g., @sanity, @smoke, @regression) using comments and run them selectively using grep or grepInvert.

Sample Test File: example.spec.ts

```
test('@sanity @smoke Check title of the home page', async ({ page }) => {
  await page.goto('https://www.google.com/');
  await expect(page).toHaveTitle('Google');
});
```

Another way to add tags to the test:

```
test('Check title of the home page',{tag:['@sanity,@regression']}, async ({ page }) => {
  await page.goto('https://www.google.com/');
  await expect(page).toHaveTitle('Google');
});
```

Run Tests by Tag (Through terminal):

You can use the **--grep** option to run only specific tagged tests:

1. Run all sanity tests:

npx playwright test tests/tagging.spec.ts --grep "@sanity"

2. Run all regression tests:

npx playwright test tests/tagging.spec.ts --grep "@regression"

3. Run tests which are belongs to both sanity & regression

npx playwright test tests/tagging.spec.ts --grep "(?=.*@sanity)(?=.*@regression)"

(?=.*@sanity) ensures the tag @sanity is present.

(?=.*@regression) ensures the tag @regression is also present.

Combined, it matches tests that include both tags.

4. Run tests belongs to either sanity or regression.

npx playwright test tests/tagging.spec.ts --grep "@sanity|@regression"

5. Run only sanity tests which are not belongs to regression

npx playwright test tests/tagging.spec.ts --grep "@sanity" --grep-invert "@regression"

Configure tags in playwright.config.ts file:

```
export default defineConfig({
 grep: /@sanity/,
 grepInvert:/@regression/
});
```

Why Use Tags?

Tags help you:

- Organize test suites (e.g., smoke, regression, sanity).
- Save time during development or quick validations.