PlayWright Grouping

Organizing your Playwright tests effectively is the difference between a suite that's easy to maintain and one that becomes a "spaghetti" mess of code. In Playwright (Node.js), grouping is primarily handled via the test.describe block.

---

1. The Basics of test.describe

The test.describe function allows you to group related tests into a logical block. This is helpful for organizing by feature, page, or user flow.

- **Structure:** It wraps multiple test() blocks.
- **Reporting:** Tools like the HTML reporter will nest these tests under the group title, making it much easier to read results.

```
import { test, expect } from '@playwright/test';

test.describe('Login Functionality', () => {
  test('successful login with valid credentials', async ({ page }) => {
    // ...
  });

  test('error message with invalid credentials', async ({ page }) => {
    // ...
  });
});
```

---

2. Shared Hooks (Scope)

Grouping is most powerful when combined with **Hooks**. When you place a hook inside a describe block, it only applies to the tests within that group.

Hook	Execution Frequency	Use Case
beforeEach	Runs before **every** test in the group.	Navigating to a URL, logging in.
afterEach	Runs after **every** test in the group.	Cleaning up data, logging out.
beforeAll	Runs **once** before the group starts.	Setting up a database, starting a server.
afterAll	Runs **once** after the group finishes.	Tearing down environments.

---

3. Advanced Grouping Patterns

Nested Groups

You can nest describe blocks to create a hierarchy (e.g., Feature > Sub-feature > Scenario).

JavaScript

```
test.describe('Admin Dashboard', () => {
  test.describe('User Management', () => {
    test('create new user', async ({ page }) => { /* ... */ });
  });

  test.describe('Settings', () => {
    test('update profile', async ({ page }) => { /* ... */ });
  });
});
```

Serial vs. Parallel Execution

By default, tests in a file run in parallel if configured. However, you can force a group to run **serially** (one after another) if they depend on the state of the previous test.

- **test.describe.configure({ mode: 'serial' });**: If one test fails, all subsequent tests in that group are skipped.
- **test.describe.parallel()**: Explicitly marks the group to run in parallel.

Conditional Grouping

You can use standard JavaScript logic to decide whether to run a group based on the environment.

JavaScript

```
const isProduction = process.env.NODE_ENV === 'prod';

test.describe('Destructive Cleanup Tests', () => {
  test.skip(isProduction, 'Do not run cleanup in Production!');
  // ... tests here
});
```

---

4. Best Practices

- **Keep it Flat:** Avoid nesting more than 2–3 levels deep; it makes the code hard to read.
- **Logical Naming:** Name your groups after **User Goals** (e.g., "Shopping Cart Checkout") rather than technical components.
- **Atomic Tests:** Even within a group, try to keep tests independent so they can be retried or run in isolation if needed.