# Test Annotation in Playwright (JavaScript/TypeScript)

In Playwright for JavaScript/TypeScript, the concept of an `@test` annotation, commonly found in Java frameworks like JUnit or TestNG, does not exist. Playwright utilizes a different approach for structuring and running tests without the need for explicit `@test` annotations. Instead, Playwright uses the `test` function to define and execute test cases. In this section, we will explore how Playwright achieves what traditional annotations do in other frameworks.

## 1. The Purpose of @test Annotations in General Frameworks

In traditional test frameworks like JUnit or TestNG, `@test` annotations are used to mark methods as test cases. These annotations indicate that a particular method should be executed as a test during the test run. Annotations make it easy to organize and manage tests, automatically identify test methods, and apply additional behaviors such as skipping tests, grouping them, or marking them as expected failures.

## 2. How Playwright Achieves Similar Functionality

While Playwright for JavaScript/TypeScript does not use `@test` annotations, it offers an intuitive approach through the `test` function to organize and execute test cases. The `test` function is the core of Playwright's test runner, providing a cleaner and simpler way to structure tests compared to annotation-based systems.

Here's an example of how a test is defined in Playwright:

import { test, expect } from '@playwright/test';  
  
test('should load homepage and check title', async ({ page }) => {  
 await page.goto('https://example.com');  
 const title = await page.title();  
 expect(title).toBe('Example Domain');  
});

In this example, the `test` function itself is used to define the test case without requiring any explicit `@test` annotation. This approach simplifies the test structure, and Playwright's test runner automatically recognizes it as a test case. The use of anonymous functions ensures each test is self-contained and has access to relevant fixtures like the `page` object.

## 3. Additional Test Annotations in Playwright

Although Playwright doesn't have a direct equivalent to `@test`, it offers annotations like `test.skip`, `test.only`, `test.fixme`, and `test.slow` for controlling test behavior. These annotations provide additional flexibility in deciding how and when tests are run.

For example, you can skip a test or mark it as expected to fail using these annotations:

test.skip('skip this test for now', async ({ page }) => {  
 await page.goto('https://example.com');  
});  
  
test.fixme('this test is under development', async ({ page }) => {  
 await page.goto('https://example.com');  
});

## Conclusion

In summary, although Playwright for JavaScript/TypeScript doesn't rely on `@test` annotations, it provides a highly effective structure for defining, organizing, and executing test cases through the use of the `test` function. Playwright's test runner also offers flexible annotations like `test.skip` and `test.only` to control test execution, providing similar advantages without the need for explicit `@test` declarations.