Annotations in Playwright

What is an Annotation?

An **annotation** is a special instruction in programming used to provide extra information about code. It helps tools and frameworks understand how to handle specific parts of the code.

In **Playwright**, annotations help describe test cases — like when they should run, whether to skip them or highlight them for exclusive execution.
They make your test suite easier to manage and organize.

Commonly Used Annotations in Playwright

`test.only()`

- Use this when you have **multiple tests** in a file but want to run **only one**.
- Helps you focus on a specific test during debugging.

Example:

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

test('testOne', async ({ page }) => {
  console.log('Test One');
});

test.only('testTwo', async ({ page }) => {
  console.log('Test Two');
});
```

In the above code, **only `testTwo` will run**, because we used `.only()`.

`test.skip()`

- Use this when you want to **skip a test** from execution.
- Can be used **with or without a condition**.

`test.skip()` (with and without condition)

With Condition:

You can skip a test **only when a certain condition is met**, like skipping non-Chromium browsers:

```
const { test, expect } = require('@playwright/test');

test('testOne', async ({ page, browserName }) => {
  if (browserName !== 'chromium') {
    test.skip();
  }
  console.log('Executed because the browser is Chromium');
});
```

Without Condition:

You can directly skip a test with `.skip()`:

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

test.skip('testOne', async ({ page }) => {
  console.log('Skipped — this will not print');
});

test('testTwo', async ({ page }) => {
  console.log('It will print because this test is not skipped');
});
```

`test.fixme()`

- Use this when a test case is **still under development**.
- Or if a test is **known to fail** and you want to **temporarily disable it**.
- Similar to `.skip()`, but better for tracking incomplete or unstable tests.

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

test('testOne', async ({ page }) => {
  test.fixme();
  console.log('This will not be printed');
});
```

`test.fail()`

- Use this when you want to **intentionally mark a test as expected to fail**.
- This helps when testing **negative scenarios** or tracking known issues.

📝 **Important Note:**
Always place the annotation (e.g., `test.fail()`) **before any test logic**.

Example 1: Annotation fails but Assertion passes

```
test('fail annotation', async ({ page }) => {
  test.fail(); // Expectation to fail the test
  console.log('Testing fail annotation behaviour');
  expect(1).toBe(1); // This assertion will pass
});
```

In the report (`npx playwright show-report`), you'll see:

- **Expected:** Fail
- **Actual:** Pass
  👉 **Result:** Test fails because expectations didn't match actual result.

Example 2: Annotation fails and Assertion fails

```
test('fail annotation test', async ({ page }) => {
  test.fail(); // Expectation is to fail
  console.log('Testing fail annotation behaviour');
  expect(1).toBe(2); // This assertion fails
});
```

Now the test **passes** because:

- **Expected:** Fail
- **Actual:** Fail
  The expectation matched the result.

Negative Test Scenarios with Playwright Annotations

What is Negative Testing?

Negative testing checks how the system behaves when **something goes wrong** — like:

- Invalid user inputs
- Missing required fields
- System errors or failures

It's about ensuring the app fails **gracefully** and handles errors **properly**.

When to Use `test.fail()` in Negative Testing

You can use `test.fail()` to mark tests where **failure is expected** — useful when:

Example: Login Form with Invalid Credentials

We're testing a login form where the user enters **invalid credentials**.
The expected behavior is that the app **shows an error message**.

Example DOM

```
<input id="username" type="text" placeholder="Enter your username" />
<input id="password" type="password" placeholder="Enter your password" />
<button id="login-button">Login</button>
<div id="error-message" style="display:none;">Invalid credentials</div>
```

Example Code

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

test('login with invalid credentials', async ({ page }) => {
  // Navigate to the login page
  await page.goto('https://example.com/login');

  // Fill the form with invalid data
  await page.fill('#username', 'invalidUser');
  await page.fill('#password', 'wrongPassword');

  // Click the login button
  await page.click('#login-button');

  // Check that the error message is visible
  const errorMessage = await page.isVisible('#error-message');
  expect(errorMessage).toBe(true); // Error should be shown
});
```

Example with Playwright Annotations in Negative Scenarios

Sometimes, you may **expect a test to fail**, especially when:

- The feature is **incomplete**
- The issue is **known**
- You're testing how the app handles **errors or invalid flows**

In such cases, you can use the `test.fail()` annotation to mark the failure as **intentional**.

Example: Negative Test with `test.fail()`

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

test('login with invalid credentials (negative test)', async ({ page }) =>
{
  // Mark the test as expected to fail
  test.fail(); // Useful if the feature is under development or has a known
issue

  // Navigate to the login page
  await page.goto('https://example.com/login');

  // Fill in invalid login credentials
  await page.fill('#username', 'invalidUser');
  await page.fill('#password', 'wrongPassword');

  // Attempt login
  await page.click('#login-button');

  // Check if the error message appears
  const errorMessage = await page.isVisible('#error-message');
  expect(errorMessage).toBe(true); // This is expected, but will fail if
the feature isn't working yet
});
```

Why Use `test.fail()` in Negative Tests?

- Marks the test as **"expected to fail"** — it won't mark your CI/CD pipeline red.
- Helps track **known bugs** or **incomplete features**.
- Useful for **temporary conditions** (like while dev work is ongoing).
- If both `test.fail()` and your assertion fail — the test **passes** (because the failure was expected).

Key Takeaways for Negative Scenarios with Playwright Annotations

Common Annotations:

- **`test.fail()`**
  → Use when a test is *expected to fail*, e.g., due to a known bug or an incomplete feature.
- **`expect()` assertions**
  → Use to validate negative outcomes (e.g., invalid inputs, error messages).
- **`test.skip()`**
  → Use to *temporarily disable* tests — e.g., for unsupported browsers, unfinished features, or unstable environments.

---

test.fail() with Conditions

You can apply `test.fail()` **conditionally**, based on the browser or environment.

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

test('test fail annotation with condition', async ({ page, browserName })
=> {
  console.log('Running conditional fail test');

  if (browserName === 'chromium') {
    test.fail(); // Only mark as fail when running in Chromium
  }

  await page.goto('https://example.com');
  // Further steps...
});
```

Behavior Explanation:

Condition	Expectation	Result
`browserName === 'chromium'`	Test fails	✅ Passes (expected failure)
`browserName !== 'chromium'`	Test fails	❌ Fails (unexpected failure)

---

test.slow() – Increase Timeout for a Test

Playwright tests **fail after 30 seconds** by default. You can slow a test down using `test.slow()` — which multiplies the timeout by **3x** (so 90s by default).

Configuration File: `playwright.config.js`

```
use: {
  // other settings...
},
timeout: 30000, // default timeout is 30 seconds
```

Usage Example:

```
test('test with slow annotation', async ({ page }) => {
  test.slow(); // Waits up to 90s if default timeout is 30s

  await page.goto('https://example.com/slow-page');
  console.log('Slow test running...');
});
```

---

test.setTimeout() – Set Timeout for a Specific Test

If you want **only one test** to wait longer or shorter than the global timeout, use `test.setTimeout()`.

```
test('test with custom timeout', async ({ page }) => {
  test.setTimeout(9000); // This test will fail after 9 seconds if not
complete

  await page.goto('https://example.com');
  console.log('Custom timeout test running...');
});
```

Summary Table of Annotations

Annotation	Purpose
`test.fail()`	Mark test as expected to fail (e.g., known bug, dev in progress)
`test.skip()`	Skip test execution temporarily
`test.slow()`	Increase test timeout (default ×3)
`test.setTimeout()`	Set a custom timeout for an individual test
