Core Techniques

Terms

Boundary testing
False negatives
False positives
Loose assertions
Matchers
Negative testing
Parameterized tests
Positive testing
Robust tests
Setup
Tear down
Tight assertions
Trustworthy tests

Summary

- It's better not to write tests than to write bad tests; good tests should be easy to maintain, reliable, and trustworthy.
- To make your tests easier to maintain, give them clear names, focus on testing one thing at a time, and keep them short and well-organized.
- Effective tests are *robust* and don't break easily when you make small changes to your code, as long as the main functionality remains the same. To achieve this, test what the code should do, not how it does it. In other words, test the behavior, not the implementation.
- *Tight assertions* can result in fragile tests that frequently break, while *loose assertions* can lead to tests that consistently pass (false positives) and fail to provide us with enough confidence that our software works. Striking the right balance between loose and tight assertions is crucial for effective testing.

- Tests should not be dependent on random values, current date/time, or any global state, as this can cause them to give different results each time they are run.

- A *matcher* is a function (or method) that allows us to make assertions about the values produced during testing. Examples are toBe, toEqual, toBeNull, etc.

- *Positive tests* make sure that our application works under normal conditions, while negative tests check how it handles unexpected or invalid input.

- *Boundary testing* is when we test how our software behaves in extreme situations or at the edges of what it can handle.

- *Parameterized* tests, also known as *data-driven tests*, are a testing technique where a single test case is executed multiple times with different sets of input data or parameters.

- *Setup* and *teardown* are phases in testing where setup prepares the initial environment, while teardown cleans up the environment after the test, ensuring test isolation and consistency. Vitest (and Jest) provide four hooks for these phases: beforeEach, beforeAll, afterEach, and afterAll.

Common Matchers

Equality

`toBe()`
`toEqual()`

Truthiness

`toBeTruthy()`
`toBeFalsy()`
`toBeNull()`
`toBeUndefined()`
`toBeDefined()`

Numbers

`toBeGreaterThan()`
`toBeGreaterThanOrEqualTo()`
`toBeLessThan()`
`toBeLessThanOrEqualTo()`
`toBeCloseTo()`

Strings

`toMatch()`

Objects

`toMatchObject()`
`toHaveProperty()`

Arrays

`toContain()`
`toHaveLength()`

