TESTING WITH VITEST

NUXT + VITEST 🤝

AGENDA

- Unit testing concepts (~5min)
- Mocking foundations (~15min)
- Demo: Tooling (~5min)
- Demo: Nuxt testing (~10min)
- Mocking cheat sheets (~10min)

UNIT TESTING CONCEPTS

STRUCTURING TESTS

- **System Under Test (SUT)** represents the module that is being tested
- **Test Candidate** synonym for the SUT
- **Actors** entities that interact with the SUT during the test, such as other modules

ORGANIZING TESTS

- **Arrange** set up the conditions for the test
- **Act** invoke the SUT
- **Assert** verify that the SUT behaved as expected
- **Teardown** optional clean up, e.g., resetting test doubles

TEST CANDIDATE

```ts
// calculate.ts
import { logger } from "./logger";

export function calculate(num1, num2) {
  const result = num1 + num2;
  logger.log(`The result is ${result}`);
  return result;
}
```

TEST STRUCTURE

```
1  // calculate.spec.ts
2  import { describe, expect, it, vi } from
   "vitest";
3  // Import the test candidate / SUT
4  import { calculate } from "./calculate";
5  // import the module to substitute (used
   inside of SUT)
6  import { logger } from "./logger";
7
8  // create test suite explicitly with
   describe
9  describe("calculate function", () => {
10   // create actual test (alias for test)
11   it("should log the result of the
   calculation", () => {
12     // ...
13   });
14 });
```

ARRANGE

```
1  // mock the actor to prevent actual log
   output
2  vi.mock("./logger");
3  describe("calculate function", () => {
4    it("should log the result of the
   calculation", () => {
5      // Arrange: Set up the test candidate
   and the actors
6      const num1 = 5;
7      const num2 = 10;
8      const expectedResult = 15;
9      // gain access to the mock to assert on
   it later
10     const loggerMock = vi.mocked(logger);
11     // ...
12   });
13 });
```

ACT AND ASSERT

```
it("should log the result of the calculation", () => {
  // ...

  // Act: Call the SUT with the arranged args
  const result = calculate(num1, num2);

  // Assert: Check that the SUT behaved as expected
  // perform state verification
  expect(result).toBe(expectedResult);
  // perform behavior verification

expect(loggerMock.log).toHaveBeenCalledWith(
    `The result is ${expectedResult}`
  );
});
```

CLEANUP / TEARDOWN

```
describe("calculate function", () => {
  it("should log the result of the calculation", () => {
    // ...
  });

  afterEach(() => {
    // Clear all mocks to ensure a clean slate
    vi.clearAllMocks();
  });
});
```

MOCKING CONCEPTS

MOCKING CONCEPTS (I)

- **Test doubles** umbrella term for various types of objects used to replace real components
- **Dummies** placeholders to satisfy function signature, not involved in the actual logic being tested
- **Fakes** Lightweight working implementations making them unsuitable for prod (e.g., in-memory DB)

MOCKING CONCEPTS (II)

- **Stubs** functions providing predefined responses without recording usage information
- **Spies** stubs that also record information about their usage helping in the verification of expected behavior
- **Mocks** functions designed to anticipate and verify specific interactions, replacing the original implementation

TEST DOUBLES WITH VITEST API

- **vi.fn** creates standalone mock function, ideal for creating stubs used as arguments of the SUT
- **vi.spyOn** spies on an existing function with optional implementation override
- **vi.mock** mocks an entire module or specific functions, isolating the SUT from its dependencies

BLURRING LINES

- distinction between fakes, stubs, spies, and mocks is not as clear-cut
- all main APIs (vi.fn, vi.spyOn, vi.mock)
- record usage information to verify interactions
- can replace parts or the whole implementation
- vi.fn can act as a stub, spy, or mock

TEST CANDIDATE

```ts
// myModule.ts
export function myFunction(arg: string): string {
  return `Original value: ${arg}`;
}

export function anotherFunction(): string {
  return "Another original value";
}
```

MOCK ENTIRE MODULE

```
import { describe, it, expect, vi } from 'vitest';
import * as myModule from "./myModule";
vi.mock("./myModule");
describe("Automatic mocking with vi.mock", () => {
  it("should mock all functions in the module", () => {
expect(vi.isMockFunction(myModule.anotherFunction))
          .toBe(true);
    // Call a mocked function
    myModule.myFunction("test");
    // Verify the interaction
    expect(myModule.myFunction)
        .toHaveBeenCalledWith("test");
  });
});
```

SPY ON MODULE

```
import { describe, it, expect, vi } from 'vitest';
import * as myModule from './myModule';
describe('Spy on myModule', () => {
  it('without changing its implementation', () => {
    const spy = vi.spyOn(myModule, 'anotherFunction');

    const result = myModule.anotherFunction();


    expect(vi.isMockFunction(myModule.anotherFunction))
        .toBe(true);
    expect(result).toBe('Another original value');
    expect(spy).toHaveBeenCalledTimes(1);
  });
});
```

CALLBACK STUB

```
1  import { describe, it, expect, vi } from
   'vitest';
2  function processCallback(
3      callback: (arg: string) => string, arg:
   string) {
4          return callback(arg);
5  }
6
7  describe("Standalone mock (as stub)", () =>
   {
8    it("should return a predefined value", ()
   => {
9      const stub =
   vi.fn().mockReturnValue("mocked value");
10     const result = processCallback(stub,
   "test");
11     expect(result).toBe("mocked value");
12
   expect(stub).toHaveBeenCalledWith("test");
13   });
14 });
```

DEMO

CHEAT SHEETS

TESTING TIMERS

```
const fetchWithPolling = async
(refetchMillis: number) => {
  const poll = async () => {
    await fetch("https://dummyjson.com");

    setTimeout(poll, refetchMillis);
  };

  poll();
};
```

FAKE TIMERS

```
describe("fetchWithPolling", async () => {
  beforeAll(() => {
    vi.useFakeTimers();
  });

  afterAll(() => {
    vi.useRealTimers();
  });

  beforeEach(() => {
    vi.clearAllMocks();
  });
});
```

ADVANCE TIMERS

```
 test("fetches at specified intervals", async () => {
    globalThis.fetch = vi.fn();
    const fetchMock = globalThis.fetch;
    await fetchWithPolling(1000);

    await new Promise(process.nextTick);

  expect(fetchMock).toHaveBeenCalledTimes(1);
    await vi.advanceTimersByTimeAsync(999);

  expect(fetchMock).toHaveBeenCalledTimes(1);
    await vi.advanceTimersByTimeAsync(2);

  expect(fetchMock).toHaveBeenCalledTimes(2);
    await vi.advanceTimersByTimeAsync(2000);

  expect(fetchMock).toHaveBeenCalledTimes(4);
  });
```

PARTIALLY MOCK A MODULE (I)

```
import { describe, it, expect, vi } from 'vitest';
import * as myModule from './myModule';

vi.mock('./myModule', async () => {
  const originalModule = await vi.importActual
    <typeof myModule>('./myModule');
  return {
    ...originalModule,
    myFunction: vi.fn().mockReturnValue('mocked value'),
  };
});
it('Partially mock module with vi.mock', () => {
    // ...
});
```

PARTIALLY MOCK A MODULE (II)

```
1  it('Partially mock module with vi.mock', () => {
2      const result1 = myModule.myFunction('test');
3      const result2 = myModule.anotherFunction();
4
5
   expect(vi.isMockFunction(myModule.myFunction))
6          .toBe(true);
7
   expect(vi.isMockFunction(myModule.anotherFunction))
8          .toBe(false);
9
10     expect(result1).toBe('mocked value');
11     expect(result2).toBe('Another original value');
12 });
```

STUB GLOBALS (I)

```
test("stub window", () => {
  vi.stubGlobal("window", {
    innerWidth: 1024,
    innerHeight: 768,
  });

  expect(window.innerWidth).toBe(1024);
  expect(window.innerHeight).toBe(768);
});
```

STUB GLOBALS (II)

```
  test("stub console.log", () => {
    vi.stubGlobal("console", {
      log: vi.fn(),
    });

    console.log("Hello, World!");

    const log = vi.mocked(console.log);
    expect(log).toHaveBeenCalledWith("Hello,
World!");
    expect(vi.isMockFunction(log)).toBe(true);
  });
```

PARAMETERIZED TESTS

```
import { test, expect } from "vitest";

const inputs = ["Hello", "world", "!"];

test.each(inputs)("Testing string lengths of %s",
    (input) => {
  expect(input.length).toBeGreaterThan(0);
});
```

MOCK FETCH (I)

```
1  test("variant with globalThis.fetch", async
   () => {
2      const dummyData = { message: "hey" };
3      const stubResponse = {
4        ok: true, statusText: "OK",
5        json: async () => dummyData,
6      } as Response;
7
8      globalThis.fetch = vi.fn()
9          .mockResolvedValue(stubResponse);
10     const response =
11         await
   fetch("https://dummyjson.com/products");
12     const data = await response.json();
13     expect(data).toEqual(dummyData);
14 });
```

MOCK FETCH (II)

```
1  test("variant with vi.stubGlobal", async ()
   => {
2      const dummyData = { data: "hey" };
3      const dummyBlob = new Blob();
4      vi.stubGlobal("fetch",
5          vi.fn(() => Promise.resolve({
6              blob: async () => dummyBlob,
7              json: () => dummyData,
8          })));
9      const response = await
   fetch("dummyUrl");
10     const data = await response.json();
11     const blob = await response.blob();
12     expect(data).toEqual(dummyData);
13     expect(blob).toEqual(dummyBlob);
14 });
```

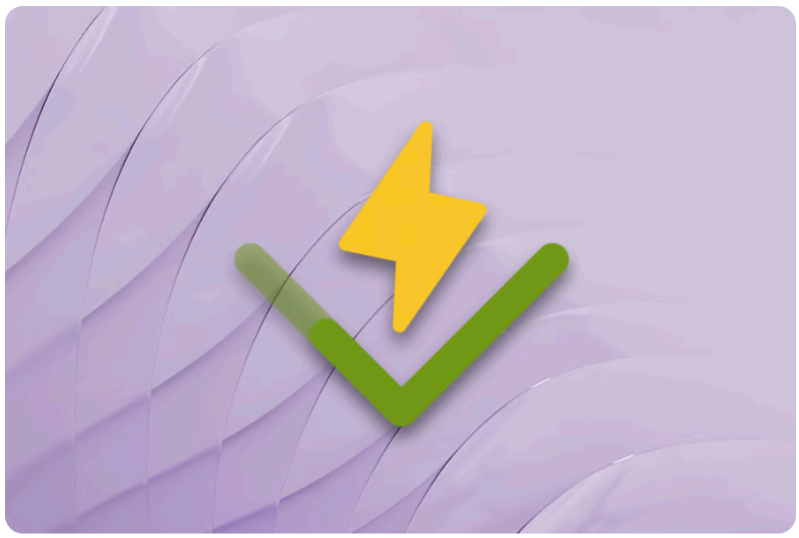
EXPECTING ERRORS (I)

```
1 it("should throw error", () => {
2     expect(() => {
3         throw new Error("Error message");
4     }).toThrow("Error message");
5 });
```

EXPECTING ERRORS (II)

```
1  it("should throw error aft. rejected fetch",
   async () => {
2      const errMsg = "Network error";
3      vi.stubGlobal(
4          "fetch",
5          vi.fn(() => Promise.reject(new
   Error(errMsg))),
6      );
7
8      await
   expect(fetch("https://api.example.com")).
9          rejects.toThrow(errMsg);
10 });
```

my blog post about Vitest



An advanced guide to Vitest testing and mocking

Dev

Use Vitest to write tests with practical examples and strategies, covering setting up workflows, mocking, and advanced testing techniques.

Sebastian Weber

Jun 6, 2024 · 24 min read

Speaker notes