FE UNIT TESTING

It’s a real fun time

Why Unit test?

- Definition:
  - *A **unit test** is a type of automated test that focuses on verifying that a small, isolated piece of code—usually a single function, method, or component—behaves as expected.*

- Catch bugs early
- Easy to write
- Makes refactoring safer
- Documents intended behavior
- Encourages modular, testable code

Writing unit tests

Pros:	**Cons:**
-they'll improve the quality of my code **-it'll take like 10 mins max** **-literally everyone says that I should**	**-i don't wanna**

CONCLUSION: i will not write unit tests

VS Code Extensions I use

- Code Coverage
- Jest
- And then copilot of course

REMINDER!!!

- There are different types of tests:
  - *Unit tests, integration tests, end-to-end tests, regression tests, performance tests, smoke tests, security tests etc.*
- For our repositories, SE are expected to do *unit* tests
- To keep things as clean as possible, ensure there is one test file per code file

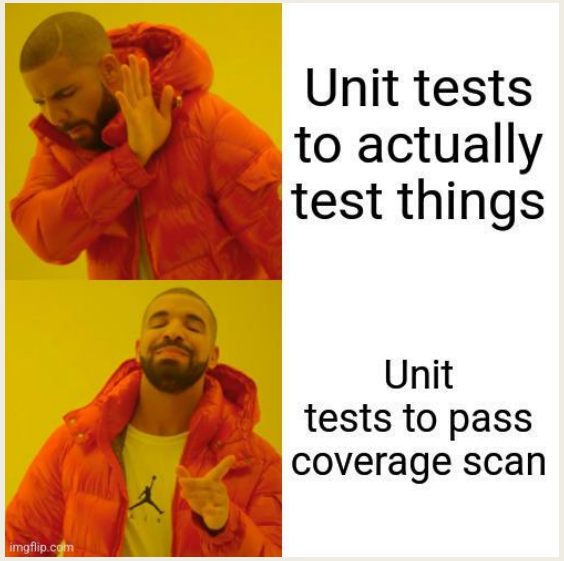
Aspect	Unit Tests	Integration Tests	E2E Tests
Scope	Single unit of code	Interaction between modules	Entire application flow
Dependencies	Mocked	Real or mocked	Real
Speed	Fast	Moderate	Slow
Confidence Level	Low	Medium	High
Tools	Jest, Mocha	Jest, Testing Library	Cypress, Playwright

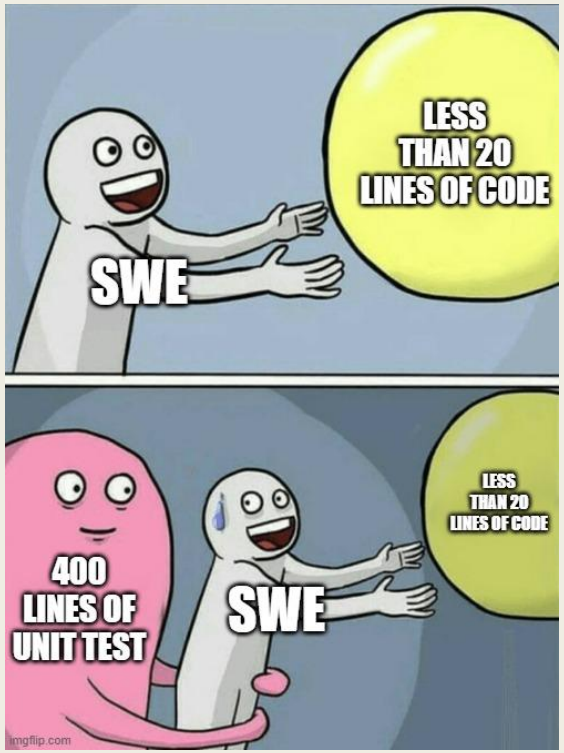
Code Coverage

Writing *unit* tests vs passing a coverage gate

```
----------|---------|----------|---------|---------|-------------------
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------|---------|----------|---------|---------|-------------------
All files |       0 |      100 |       0 |       0 |
 App.js   |       0 |      100 |       0 |       0 | 1
 Day.js   |       0 |      100 |       0 |       0 | 1-5
----------|---------|----------|---------|---------|-------------------
```

- Statements
  - *Measures individual statements executed*
- Branch
  - *Measures what lines of logic have been executed. If, else if, else*
- Functions
  - *Measures what functions have been executed*
- Lines
  - *Measures lines executed (similar to statements but different)*
- Uncovered Line #s
  - *What lines are uncovered, great for guiding your tests*





Best Practices

- Unit test off use cases, not test coverage
  - *Think of your edge cases!!*
- Start your testing by writing what tests you are going to do, then implement them
  - *Use test coverage to gauge if you're missing anything*

EXAMPLE1

Code Coverage

TOOLS

findBy, getBy, queryBy

Type of Query	0 Matches	1 Match	>1 Matches	Retry (Async/Await)
Single Element				
`getBy...`	Throw error	Return element	Throw error	No
`queryBy...`	Return `null`	Return element	Throw error	No
`findBy...`	Throw error	Return element	Throw error	Yes
Multiple Elements				
`getAllBy...`	Throw error	Return array	Return array	No
`queryAllBy...`	Return `[]`	Return array	Return array	No
`findAllBy...`	Throw error	Return array	Return array	Yes

Best practices: query priority

- “Your test should resemble how users interact with your code (component, page, etc.) as much as possible. With this in mind, we recommend this order of priority”
  1. **ByRole*
  2. **ByLabelText*
  3. **ByPlaceholderText*
  4. **ByText*
  5. **ByDisplayValue*
  6. **ByAltText*
  7. **ByTitle*
  8. **ByTestId* ← This should be your last resort

render and renderHook

- Render
  - *Renders react components in a test*
  - *Allows you to interact with the rendered DOM*
  - *Used to test the UI*
- RenderHook
  - *Allows you to test the behavior of a hook in isolation*
  - *Used to unit test hook logic and behavior*
- Testing utils
  - *No need for Render or Render hook, just import and test the function!*

Act warnings and how to fix them

- The act warning from React is there to tell us that something happened to our component when we weren't expecting anything to happen. (This means your test is not testing everything that’s happening)
- Use cases for manually calling act()
  - *When using jest.useFakeTimers*
  - *When using custom hooks*

EXAMPLE2

How to test with render

EXAMPLE3

How to test with renderHook

fireEvent vs userEvent

- fireEvent
  - *Purpose: Simulates individual DOM events (e.g., change, click, etc.).*
  - *Behavior: Directly triggers the specified event on the target element without simulating intermediate events.*
  - *Use Case: Useful for simple interactions or when you need fine-grained control over specific events.*
- userEvent
  - *Purpose: Simulates real user interactions, including typing, clicking, and more.*
  - *Behavior: Mimics how a user interacts with the DOM, triggering all associated events (e.g., keydown, keypress, keyup, input for typing).*
  - *Use Case: Ideal for testing realistic user interactions.*
- When to use fireEvent
  - *Testing low-level event handlers (e.g., keydown, focus, scroll).*
  - *Simulating programmatic or non-user-triggered events.*
  - *When you need precise control over the event payload.*

FASTER

MORE REALISTIC

EXAMPLE4

fireEvent vs userEvent

MOCKING

Graphql and unit testing

- Jest.mock <- I prefer this in components, I like to ensure that where we actually use useQuery and useMutation is in its own separate file
- MockedProvider <- This is apollo's recommendation for best practices for testing React components that use Apollo Client

```
1  import "@testing-library/jest-dom";
2  import { render, screen } from "@testing-library/react";
3  import { MockedProvider } from "@apollo/client/testing";
4  import { GET_DOG_QUERY, Dog } from "./dog";
5
6  const mocks = []; // We'll fill this in next
7
8  it("renders without error", async () => {
9    render(
10     <MockedProvider mocks={mocks}>
11       <Dog name="Buck" />
12     </MockedProvider>
13   );
14   expect(await screen.findByText("Loading...")).toBeInTheDocument();
15 });
```

EXAMPLE5

jest.mock and <MockProvider />

DEBUGGING

Flaky tests and how to debug them

- Flaky
  - *Timeouts (we see this frequently when autocomplete is in the picture)*
  - *MSW issues (moving away from MSW and replacing with jest.mock)*
- Debugging
  - *Breakpoints*
  - *Console.log*
  - *Screen.debug*
  - *Throw it all out and start from scratch*

Now go off and test

Sources

- https://testing-library.com/docs/queries/about
- https://testing-library.com/docs/dom-testing-library/api-debugging/
- https://kentcdodds.com/blog/common-mistakes-with-react-testing-library
- https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Reference/Roles
- https://www.apollographql.com/docs/react/development-testing/testing
- https://kentcdodds.com/blog/fix-the-not-wrapped-in-act-warning