React Tutorial
Lesson 6

Today's lesson will cover

- Testing in React
- Test Suites, Test Cases, and Assertions
- Unit Testing: Functions
- Unit Testing: Components
- Integration Testing: Component
- Snapshot Testing
- React Project Structure

Intro to React

Testing in React

- Testing source code is an essential part of programming, and should be seen as a mandatory exercise for serious developers.
- The testing pyramid will serve as our guide which includes end-to-end tests, integration tests, and unit tests.
- **Unit tests** are for small, isolated blocks of code, such as a single function or component.
- **Integration tests** help us figure out how well these blocks of code work together.
- An **end-to-end test** simulates a real-life scenario, like a user logging into a web application. Unit tests are quick and easy to write and maintain; end-to-end tests are the opposite.
- Some of the syntax basics worth being familiar with:
    - `it/test` would execute passed function as a block of tests.
    - `describe` is optional method for grouping any number of it or test statements.
    - `expect` is the condition that the test needs to pass.
    - `mount` method renders the **full DOM** on which you are running the tests.
    - `shallow` renders only the **individual components** that you are testing

Test Suites, Test Cases, and Assertions

- Test suites and test cases are commonly used in JavaScript and many other programming languages.
- A **test suite** groups the individual test cases into one larger subject.
- The **"describe" block is test suite**, and the **"test" blocks are our test cases**. Note that test cases can be used without test suites.
- A "test" block can also be written as an "it" block. The blocks have the same purpose, except the "it" block may be more familiar to programmers from other programming languages
- An **assertion works by expecting value on the left side** `(expect)` **to match a value on the right side** `(toBe)`**.** The assertive function `toBe` is one of many available with Jest

Unit Testing: Functions

- A unit test is generally used to test components or functions in isolation.
- For functions, unit tests are for input and output; for components, we test props or the callback handlers communicating to the outside.
- Continue to make adjustments until the reducer test turns green, which is really testing a JavaScript function with a certain input and expecting a certain output.

Unit Testing: Components

- React Testing Library (RTL) can be used to test isolated React component with a unit test.
- Jest lets us pass a test-specific function to the Item component as prop. These test specific functions are called spy, stub, or mock; each is used for different test scenarios. The jest.fn() returns us a mock for the actual function, which lets us capture when it's called. As a result, we can use Jest assertions like toHaveBeenCalledTimes, which lets us assert a number of times the function has been called; and toHaveBeenCalledWith, to verify arguments that are passed to it.
- Every time we want to spy a JavaScript function, whether it has been called or whether it received certain arguments, we can use Jest's helper function to create a mocked function. Then, after invoking this function implicitly with RTL's fireEvent object's function, we can assert that the provided callback handler -- which is the mocked function -- has been called one time.

Integration Testing: Component

- React Testing Library adheres to a single core philosophy: instead of testing implementation details of React components, it tests how users interact with the application and if it works as expected. This becomes especially powerful for integration tests.
- There may be some confusion about when to use getBy or the queryBy search variants. As a rule of thumb, use getBy for single elements, and getAllBy for multiple elements. If you are checking for elements that aren't present, use queryBy (or queryAllBy).
- React Testing Library with Jest is the most popular library combination for React testing. RTL provides relevant testing tools, while Jest has a general testing framework for test suites, test cases, assertions, and mocking capabilities. If you need an alternative to RTL, consider trying Enzyme by Airbnb.

Snapshot Testing

- Facebook created snapshot tests as a more lightweight way to test React components and their structure.
- A snapshot test creates an instance of your rendered component's output as HTML elements and their structure. This snapshot is compared to the same point in the next text to give more output on how the rendered component changed and show why any tests failed in the difference.
- Jest stores snapshots in a folder so it can validate the difference against future snapshot tests. Users can share these snapshots across teams using version control platforms like git. This is how we make sure the DOM stays the same.
- Snapshot tests are useful for setting up tests quickly in React, though it's best to avoid using them exclusively. Instead, use snapshot tests for components that don't update often, are less complex, and where it's easier to compare component results.

React Project Structure

- There are many ways on how to structure your React project from small to large project: simple to complex folder structure; one-level nested to two-level nested folder nesting; dedicated folders for styling, types and testing next to implementation logic. There is no right way for folder/file structures.
- A project's requirements evolve over time and so should its structure. If keeping all assets in one file feels right, then there is no rule against it. Just try to keep the nesting level shallow, otherwise you could get lost deep in folders.
- There is no official opinionated suggestions for structuring your project. However, keeping the following points in mind can help:
    - Grouping by features or routes
    - Grouping by file type
    - Avoid too much nesting
    - Don’t overthink it

the shortcut

Self Study Assignments

To Dos

- Worth going through the Jest documentation on testing React apps: https://jestjs.io/docs/en/tutorial-react
- Unless you have your own project to work on to practice what we have learned, complete the Pokemon API app with React from here: https://learn.chrisoncode.io/webinars/javascript-to-react
- Start working on React or Fullstack JavaScript project, OR complete FCC Frontend Library Certification : Bootstrap, React, Redux, React & Redux, projects; APIs and Microservices Certification : https://www.freecodecamp.org/learn
- Refer to Reactjs official documentation as the best place to learn underlying main concepts of React with examples: https://reactjs.org/docs/hello-world.html