Recap: Unit testing

Overview

How do you guarantee that the app you code is working as planned? You could test all the pieces of functionality manually by opening up the app in the browser and going through all the possible interactions. However, this would be time-consuming and error-prone.

Besides this manual testing approach, other approaches are usually better suited for the task. These approaches are unit testing, automated testing and integration testing. In this reading, you'll revisit the concept of unit testing.

Unit testing: A quick refresher

Unit testing is testing that focuses on units of code. In practice, when testing React apps, you write unit tests for specific react components. Let's explore this using a simple code example.

The following React app consists of a single component, App component. This app is just a button that increments a number value on click. The app's code is as follows:

```
import React from "react";
export default function App() {
  const [number, setNumber] = React.useState(1);
  function increment() {
    setNumber((prevNumber) => prevNumber + 1);
  }
  return (
    <>
      <h1 data-test-id="currentNumber"> {number} </h1>
      <button data-testid="add-one" onClick={increment}>
          Add one
      </button>
    </>
  );
}
```

Now that you have the full app's code, you can write the unit test for it.

Writing unit tests

Here is the test code of the test file named App.test.js:

```
import { render, fireEvent, screen } from "@testing-library/react";
import App from "./App";

test("Adds one", () => {
  // render the App component
  render(<App />);

  // save the heading in a variable
  const heading = screen.getByTestId("currentNumber");

  // save the button in a variable
  const btn = screen.getByTestId("addOne");

  // click the btn
  fireEvent.click(btn);

  // test assumption
  expect(heading).toHaveTextContent("2");
```

```
19  });
20
```

After you've written the test, you can run it using the command `npm test` or `npm run test`, which are essentially the same command. Once you run the test command, the output will look as follows:

```
1   PASS  src/App.test.js
2     √ Adds one (29 ms)
3
4   Test Suites: 1 passed, 1 total
5   Tests:       1 passed, 1 total
6   Snapshots:   0 total
7   Time:        3.278 s
8   Ran all test suites.
9
10  Watch Usage: Press w to show more.
```

Testing best practices

In addition to recapping what unit tests are and how to run them, it's worth summarizing some best practices to keep in mind when writing tests. These include:

- Avoid including implementation details.
- Work with DOM nodes.
- Resemble software usage.
- Keep maintainability in mind.

Conclusion

In this reading, you revisited the concept of unit testing and how developers use it to ensure that their code works as expected. Next up, you'll have the opportunity to work with unit tests first-hand by adding unit tests to your project. To explore the topics recapped in this reading in greater depth, please refer to the following lesson items from the **Advanced React** course:

- Why React Testing Libary
- Writing the first test for your form