Testing Our Application

Episode -13

① testing Manually
② Writting test case ← writing code that will test our application

Types of testing (developer)

1) Unit testing ← testing a single component
2) Integration testing ← testing multiple component integrated
3) End to End Testing - e2e testing ← testing the the flow from entering the site to leaving.

React testing Library.

↳ build ontop of DOM testing library

→ when we use Create React app the testing library is already included.

→ It uses Jest (Javascript testing framework)

So we will install both
(As we are learning react from Strach)

① → npm i -D @testing-library/react

② npm i -D jest

Now are are using jest with babel so we will need to install additional library.

https: jestjs.io/docs/getting-started

③ npm install --save-dev babel-jest @babel/core @babel/preset-env

Note
Here parcel has it's own babel and Jest also has its own so they should not conflict so we need to create a config

Also create babel.config.js from jes docs / getting Started

https: parcel.org/languages/javascript → usage with other tools

④ → create a new file .parcelrc
And paste that code from above site

Now to run test we have command in Pakage.json

```
"scripts" : {
    "test" : "jest"
```

Command → npm run test

test cases so this does not run on browser so they will need a simulated environment,

Jsdom is like a browser to test

⑤ Create jest config

→ npx jest --init

typescript → N
test environment → jsdom
Coverage report → Yess
Provider → babel
Mock calls etc → Yess

jest.config.js will be created

⑥ Install Jsdom library (Because if we are using higher version of jest then it's lib needed to be installed seprately)

(greater than 28)

npm i -D jest-environment-jsdom

So the test cases are written in folder or file with extension

all the files inside this tells files → __test__

xxx.test.js
xxx.test.ts
xxx.spec.js
xxx.spec.ts

this is called as Dunder (two under scores)

Also called as dunder test

How to write a test case in sum.test.js

→ import {sum} from "./sum";

↔ a normal sum code in js

test("sum function should calculate the sum of two number", () => {

const result = sum(3,4);

// Assertion
expect(result).toBe(7);

});

↑ description of our test

↑ call back function

To run test write → npm run test

① Making our contact us page look good.

Now we are testing the contact us Component

Testing A component weather it Rendered or not

① Contact.test.js create this file

to render in Is dom ↓ to get the elements from screen

```
import {render, screen} from "@testing-library/react";
import Contact from "../Contact";
import "@testing-library/jest-dom"

test("Should load contact us component", () => {
    render(<Contact />);
    const heading = screen.getByRole("heading");
    expect(heading).toBeInTheDocument();
});
```

different types role

→ here we are checking that if we rendered the contact us page then it should render all the components present inside it

So we use screen and render provided by testing library to check

② We need to install @babel/preset-react because the jsx format is not enabled in jest

in babel.config.js add a preset

this is helping to convert jsx to html babel is transpiler

important

```
["@babel/preset-react", {runtime: "automatic"}],
```

(3) • toBeInTheDocument() this function also comes from a library

npm i -D @testing-library/jest-dom

So install this and import it in your test document

→ import "@testing-library/jest-dom";

Beauty of jest is whenever it fails it also shows what was rendered on the page

---

Now to search the element on the screen their are many ways ex → ~~toget~~

screen.getAllByPlaceholderText(
getAllByAltText(
getAllByTitle(
getAllByLabelText(

• getBy	• getAllBy
expecting 1 result	expecting many result

→ If their are multiple and you are using getBy it will throw a error.

* When the no. of test case increases you can group your test cases by using a function

```
describe("Contact us test cases", () => {
    test(...) {
    test(...)
})
```

You can also have describe under describe

* You can use it or test same keyword

add the coverage in git ignore

*Adding testcases for header Component

→ So in header Component you used 3 new things

(1) You need to Provide you redux store to this dom

(2) You have used a <Link> tag this link tag is provided by "react-routed-dom";

Extra

→ So import appStore from "../../utils/appStore";
import {BrowserRouter} from "react-router-dom";

left to watch.