ep-13 Time for Test

- Writing Test cases for our application

↳ Importance of writing Test cases

- Whenever we are changing something in our application, it can introduced bug in other part of application also.

- There is Test cases, means code that test application automatically after writing it once

- Developer no need to test manually every single feature

↳ Types of Testing

- 3 types of testing we can do as an react developer.
  1] Unit Testing
  2] Integration Testing
  3] End to End Testing

1] Unit Testing -
Test react component in isolation.

Testing small unit of app, not the worrying about whole app

eg Header component.

apsara

Date: / /

2) Integration Testing -
Testing the integration of the component

Multiple components talking to each other for specific feature, the action flow is getting created by them, Testing it is known Integration testing

3) End-to-End Testing -

Testing all the action flow from user lands to website to the user leaves the website.

Selenium like tools needs to test E2E testing.

↳ Libraries for Testing

1) React Testing Library

- React Testing Library built on top of 'DOM Testing Library'.
- create-react-app already have React Testing Library.
- React Testing Library uses something known as 'jest'.
- Jest is delightful Javascript Testing framework

Page No. / /

- Jest is different Testing Library

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

- Jest Using Babel needs other dependancies as well

  npm i --save-dev babel-jest @babel/core @babel/preset-env.

- Configure babel

  Create babel.config.js

- parcel bundler behind the scense uses 'babel' and it have its own babel configuration. But we also add root level babel config for jest. So its creates conflict for parcel for babel configuration. (1)

- To solve issue ~~the~~ create .parcelrc configuration. to disable default babel transpilation.

- To run test: (2)

  npm run test

↳ Jest Configuration

  npx jest --init

- jsdom : It is like browser but not actual browser and will give the features of browsers.

Install Jsdom Library :
npm install --save-dev jest-environment-jsdom

→ Sample Testing Code

- Create '__tests__' folder anywhere in folder structure. (js/ts) extensions files in that folder will consider as testing files.

e.g files name
Header.test.js
Header.test.ts
Header.spec.js
Header.spec.ts

① Testing (sum.js) code

```
export const sum = (a,b) => {
    return a+b;
}
```

② Create (sum.test.js) in __tests__ folder

- To create 'test' there is test() function.
It takes 2 args (string, callback functn)

eg import {sum} from "../sum";

test("sum function should calculate the sum of two numbers", () => {
  const result = sum(3,4);

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

- npm run test

↳ Writing Unit Testing for React

- Whenever testing UI component inside react, have to render component on to jsdom.

- To enable jsx for testing add library
  @babel/preset-react
  OR
  To make JSX works in test cases

  Include Library inside babel config

- npm @testing-library/jest-dom

eg test("should load 2 input boxes on the component", () => {
  render(<Contact />);

  const inputBoxes = screen.getAllByRole("textbox");

console.log(inputBoxes); → Returns JSX (React Element in the form of Object)
});

↳ Describe Block

- Group multiple test cases into a single block that block is known as describe

eg describe("Contact Us Page Test Case", () => {

test("test 1", () => {});

test("test 2", () => {});

test("test 3", () => {});

});

- There is multiple describe() blocks and also nested describe() as well.

- We can write test() as it() also.

↳ Advanced Unit Testing

eg① it("should load Header with login button",
() => {
render(<Header />); → throw an error
}

We are using react-redux library in Header

- jsdom only knows JS, React, JSX. It does not understand React-Redux

- We need to proxide redux store to Header component. Because testing we are doing in isolation.

- ```
  it (" ", () => {
    render (
      <Provider store = {appStore}>
        <Header />
      </Provider>
    );
  ```

```
const loginBtn = Screen. getByRole ("button");
expect (loginBtn) .toBeInTheDocument ();
};
```

↪ Integration Testing

- checking features like fetch data

- test cases running on browser like enviorment (jsdom)

- Integration Testing is used for to check implemention of the component

- OR to check accuracy between the components how they communicate with each other
```