1. Manual Testing
2. Write the test cases that can test the code automatically.

There are three type of testing as per the developer

1. **Unit Testing: testing components independently**
2. **Integration Testing: Testing the flows**
3. E2E Testing (end to end testing) : front landing to ending.

Libraries

-> ReactTesting Library

Jest Delightdul js testing framework (jestjs.io)

React testing Library use Jest behind the scenes

> Npm i -D @testing-library/react

> npm i -D jest

install babel dependencies

Configure babel

Now as we know we are using parcel

And it have its own configuration for Babel

And here we are adding it manually

There may be some conflicts

So we gave ti changes in parcel to accommodate the babel to run with jest

**npm install --save-dev @testing-library/react @testing-library/dom**

**npm install --save-dev jest**

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

https://jestjs.io/docs/getting-started

<https://parceljs.org/languages/javascript/#usage-with-other-tools>

> **npm run test**

Output:

namaste-react@1.0.0 test

> jest

No tests found, exiting with code 1

— if you got above output means you have installed jest successfully

Now we have to do jest configuuration

> **npx jest –init**

N

Jsdom

Yes

Babel

Yes

If the jest version is above 28 we have to install jsdom separately

> **npm install --save-dev jest-environment-jsdom**

**Where it will test the file**

When you write command npm run test it show you

> namaste-react@1.0.0 test

> jest

No tests found, exiting with code 1

Run with `--passWithNoTests` to exit with code 0

In C:\git\Namaste-React

38 files checked.

testMatch: \*\*/\_\_tests\_\_/\*\*/\*.[jt]s?(x), \*\*/?(\*.)+(spec|test).[tj]s?(x) - 0 matches

testPathIgnorePatterns: \\node\_modules\\ - 38 matches

testRegex: - 0 matches

Pattern: - 0 matches

It show where it wills earch for test case

> mkdir \_\_tests\_\_

Or FileName.test.js or FileName.spec.js

2 underscores (\_\_someName\_\_) is called DUNDER

**Lets start testing**

First we are creating one method to testing lets see

export const Sum = (a,b)=>{

return a+b;

}

Now we wil write test case

sun.test.js

import { Sum } from "../src/components/Sum"

test("Perform Addition of two numbers via Sum Component", ()=>{

const res =Sum(4,5);

//Assertion

expect(res).toBe(9);

})

Output:

> npm run test

> namaste-react@1.0.0 test

> jest

PASS \_\_tests\_\_/sum.test.js

√ Perform Addition of two numbers via Sum Component (2 ms)

----------|---------|----------|---------|---------|-------------------

File | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s

----------|---------|----------|---------|---------|-------------------

All files | 100 | 100 | 100 | 100 |

Sum.js | 100 | 100 | 100 | 100 |

----------|---------|----------|---------|---------|-------------------

Test Suites: 1 passed, 1 total

Tests: 1 passed, 1 total

Snapshots: 0 total

Time: 1.453 s

Ran all test suites.

All green code

Its tested successfully.

**Lets make wrong output**

import { Sum } from "../src/components/Sum"

test("Perform Addition of two numbers via Sum Component", ()=>{

const res =Sum(4,5);

//Assertion

expect(res).toBe(10);

})

Output

npm run test

> namaste-react@1.0.0 test

> jest

FAIL \_\_tests\_\_/sum.test.js

× Perform Addition of two numbers via Sum Component (5 ms)

● Perform Addition of two numbers via Sum Component

expect(received).toBe(expected) // Object.is equality

**Expected: 10**

**Received: 9**

5 |

6 | //Assertion

> 7 | expect(res).toBe(10);

| ^

8 | })

at Object.toBe (\_\_tests\_\_/sum.test.js:7:17)

----------|---------|----------|---------|---------|-------------------

File | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s

----------|---------|----------|---------|---------|-------------------

All files | 100 | 100 | 100 | 100 |

Sum.js | 100 | 100 | 100 | 100 |

----------|---------|----------|---------|---------|-------------------

Test Suites: 1 failed, 1 total

Tests: 1 failed, 1 total

Snapshots: 0 total

Time: 1.476 s

Ran all test suites.

Let's see if the component is rendering or not.

import { render, screen } from "@testing-library/react";

import Contact from "../src/components/Contact";

test("test loading of contact us component", ()=>{

render(<Contact />)

const heading = screen.getByRole("heading");

expect(heading).toBeInTheDocument();

});

> npm run test

It gives us Error.

Details:

SyntaxError: **C:\git\Namaste-React\\_\_tests\_\_\Contact.test.js: Support for the experimental syntax 'jsx' isn't currently enabled (5:12):**

3 |

4 | test("test loading of contact us component", ()=>{

> 5 | render(<Contact />)

| ^

6 | const heading = screen.getByRole("heading");

7 | expect(heading).toBeInTheDocument();

8 | });

**Add @babel/preset-react (https://github.com/babel/babel/tree/main/packages/babel-preset-react) to the 'presets' section of your Babel config to enable transformation.**

If you want to leave it as-is, add @babel/plugin-syntax-jsx (https://github.com/babel/babel/tree/main/packages/babel-plugin-syntax-jsx) to the 'plugins' section to enable parsing.

If you already added the plugin for this syntax to your config, it's possible that your config isn't being loaded.

You can re-run Babel with the BABEL\_SHOW\_CONFIG\_FOR environment variable to show the loaded configuration:

npx cross-env BABEL\_SHOW\_CONFIG\_FOR=C:\git\Namaste-React\\_\_tests\_\_\Contact.test.js <your build command>

See https://babeljs.io/docs/configuration#print-effective-configs for more info.

— It means yet jsx is not enabled for testing

To make jsx work in testing

**> npm i -D @babel/preset-react**

**After installing we have to include in babel config**

**babel.config.js**

module.exports = {

presets: [

['@babel/preset-env', {targets: {node: 'current'}}],

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

],

};

After this also we will get Error

**TypeError: expect(...).toBeInTheDocument is not a function**

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

import { render, screen } from "@testing-library/react";

import Contact from "../src/components/Contact";

import "@testing-library/jest-dom";

test("test loading of contact us component", ()=>{

render(<Contact />)

const heading = screen.getByRole("heading");

//Assertion

expect(heading).toBeInTheDocument();

});

getByRole(“button”)

getByText(“Submit”)

getByPlaceholderText ,,, there are many methods to search things on page.

To get all input or anything

getAllByRole(“textbox”) // to search input

getAllBy

If we print with help of conole log then it will give us the react dom element

To check inverse condition

expect(item).not.toBe(1) it will check not case

If we have multiple test cases a lot and want to make them group by

So we can use describe for that

describe(“Test caese list” , ()=>{

// list of all test casese we have written before.

})

We can have nested describe

We also can use “it” function in place of “test” function.

It is alias of test.

“It” message should start with “should”

To get function details just hover on the functions.

If suppose we are doing unit testing of a component that is using redux

Then this test function will not execute it will show Error

Error: Could not find react-redux context value;

/\*

const { render } = require("@testing-library/react");

import Header from "../src/components/Header";

test("Should load header component with login button", ()=>{

render(<Header />)

});

Error: could not find react-redux context value; please ensure the component is wrapped in a <Provider>

\*/

/\*

// strep 2

const { render } = require("@testing-library/react");

import { Provider } from "react-redux";

import Header from "../src/components/Header";

import appStore from '../src/utils/appStore';

test("Should load header component with login button", ()=>{

render(

<Provider store={appStore}>

<Header />

</Provider>

)

});

Error: The above error occurred in the <Link> component: \*/

//and link is coming from react-rounter-dom

// step 3

const { render, screen } = require("@testing-library/react");

import { Provider } from "react-redux";

import Header from "../src/components/Header";

import appStore from '../src/utils/appStore';

import { BrowserRouter } from "react-router-dom";

import '@testing-library/jest-dom'

test("Should render header component with login button", ()=>{

render(

<BrowserRouter>

<Provider store={appStore}>

<Header />

</Provider>

</BrowserRouter>

)

const loginButton = screen.getByRole("button");

expect(loginButton).toBeInTheDocument();

});

// done suggessfully

Button with name login

const loginButton = screen.getByRole("button", {name: "login"});

regEx

const cart = screen.getByText(/Cart/);

If we have to search with flexibility where cart is available or not we can use regex

Check login button clicked

test("Should have Logout button", ()=>{

render(

<BrowserRouter>

<Provider store={appStore}>

<Header />

</Provider>

</BrowserRouter>

)

const loginButton = screen.getByRole("button", { name: "Login"});

fireEvent.click(loginButton)

const logoutButton = screen.getByRole("button", { name: "Logout"});

expect(logoutButton).toBeInTheDocument();

})

How to pass props in test and how to test them.

We have to create mock data and we have to pass to it.

After passing search text we are getting or not?

Integration Testing

As we know its browser like env where it test so it do not have all power which browser have

Like it dont know what is fetch() method.

We can create exact same function like fetch function to use it.

We can not call api so we have to mack exact data we will get in API.

If we dont want to run npm run test again and again

We can use Jest watch method.

Go to package.json

In scripts

Write

“Watch-test::”jest –watch’

If we are using any async or state functionality in module and if you want to test

You have to wrap it in act( ()=>{ /\*body\*/ }) function

Means render will be in act it will import from react-dom/test-utils

Test ( “” async () =>{

Await act(

Async () =>{

render( /\*body\*/ )

}

)

})

Writing something in inputbox via testing that is an event

Const searchInput = getByTestId(“searchinp”)

fireEvent.change(searchInput, {target: { value: “burger” ”}}) // to add data to input

In case if you not able to find by role or placeholder or anything

Then jo to component add one attribute to it

data-testid=’value’ and you can get it by getByTestId(‘value”)

Now we have searchBar, searchButton and we have filled text also in input box

We fire click event on search

fireEvent.click(searchBtn)

// const RestaurantCard = ({resName,cousins,rating,cost,time,image}) =>{

const RestaurantCard = (props) => {

// console.log(props.resData.resName);

props = props?.resData;

return (

<div data-testid="resCardTest" className="m-4 p-4 w-[250px] bg-gray-100 rounded-lg hover:bg-green-200">

<div className="card">

<img src={'https://media-assets.swiggy.com/swiggy/image/upload/fl\_lossy,f\_auto,q\_auto,w\_660/'+props.info.cloudinaryImageId} alt="Card Image" className="rounded-lg" />

<div className="card-content">

<h2 className="font-bold py-4 text-lg">{props.info.name}</h2>

<p className="card-description">{props.cousins}</p>

<div className="card-details">

<span className="card-rating">Rating: {props.info.avgRating}</span>

<span className="card-cost">Cost: {props.info.costForTwo}</span>

<span className="card-time">Time: {props.info.sla.slaString}</span>

</div>

</div>

</div>

</div>

);

}

export const withPromotedLabel = (RestaurantCard)=>{

return (data)=>{

return(

<>

<label className="absolute p-4 bg-red-400 text-white">Recommended</label>

<RestaurantCard {...data} />

</>

)

}

}

export default RestaurantCard;

There are some methods

In describe

beforeAll( ()=>{

console.log(Before All”)

})

// this run once before all the test cases

beforeEach( ()=>{

console.log(Before Each”)

})

// it will return before each test caese

afterAll

afterEach

Add to cart Test

Conclusion

Install react testing Library

Install jest

Install babel dependencies

Configure babel

Condfier parcel config file to disable default babel transpliuation

Jest npm jest init

Install jsdom library

Install babel react

Configure it

Npm install jest dom