**How to Write Test Cases for React Components with Jest and Enzyme**

Let’s build healthy, tested components

Text

Description automatically generated

Snapshot of command “npm run test”

In this article, we will write the test cases for our React component from scratch using Jest and Enzyme. Also, we will learn more about Jest and Enzyme. Let’s start!

**What Is Jest?**

[Jest](https://jestjs.io/) is a unit testing framework to write test cases for JavaScript and React codes. Jest was created by Facebook.

The command create-react-app installs Jest and its set of dependencies, so we don’t have to worry about that.

**What Is Enzyme?**

[Enzyme](https://github.com/enzymejs/enzyme) is a JavaScript utility to test React. With the help of Jest, Enzyme becomes more powerful and takes full control of the React app to test each and every aspect of React components. Enzyme is specifically used for testing React components.

Now we will create an app with a component. Later we will write test cases for this component.

Let’s start from scratch.

**Step 1. Create an App jest-and-enzyme-demo**

To start this, we need to create a demo app using the command below. Let’s run this command and create a demo app:

npx create-react-app jest-and-enzyme-demo

After successful completion of the above command:

cd jest-and-enzyme-demonpm start

**Step 2. Open localhost:3000 in the Browser**

Check your current port (here the default port is 3000) and confirm it’s working, like so:

Graphical user interface

Description automatically generated

snapshot of URL [http://localhost:3000](http://localhost:3000/)

If we take a look at the newly created project (by create-react-app) in the src folder, you will see there is already a file App.test.js, which is used to test the app component using Jest.

If we run this test case by using the command npm run test, it will run the test cases using Jest.

This command runs Jest and picks the App.test.js file because of its specific extension .test.js. It will run it using Jest, and you will get the following result:

A screenshot of a computer

Description automatically generated with medium confidence

snapshot of “npm run test”

**Step 3. Delete the App.test.js File**

Now delete the App.test.js file. We will write our own test cases using Jest and Enzyme in a different file. We are deleting this file to get a better result on the command/terminal. If we don’t delete it, then we won’t get a detailed result of the test case. Instead, we will get the combined result on the terminal (command prompt).

**Step 4: Create a React Component**

We will create a component that will have a button to hide and show the text, as you can see in the GIF below:

Text

Description automatically generated

A component to be tested

Create a file ToggleComponent.js in the src folder (src/ToggleComponent.js) and write the following code:

The above code is very simple, and it’s a basic React component, so I don’t think I need to describe it.

Let’s move ahead.

**Step 5. Use the “ToggleComponent” Component in App.js:**

Now let’s use this component in our App.js. Delete all code in App.js and write the following:

In the above App.js file, we are simply using the ToggleComponent.  
Now run the app using the command npm start and look in the browser ( URL localhost:3000). You will see that there is a button with the text “Hide” or “Show,” and after clicking the button, the text will be shown or hidden, as we stated in the GIF above.

**Step 6. How to Write Test Cases Using Jest and Enzyme**

Now that we have created a component, it’s time to write test cases and verify the component.

Let’s create a file ToggleComponent.test.js in the src folder (src/ToggleComponent.test.js ) and write the following code. (First I will show you a simple test case and then run it):

describe('ToggleComponent', () **=>** { it('Should be true', () **=>** { **const** test = true; expect(test).toBe(true);  
 });});

Let’s understand the code above.

Jest has a function called describe(), which is used to create test suites in the block. We are testing ToggleComponent, so we will pass ToggleComponent as its first argument. The first argument is a string, and the second argument is a function that contains all the rest of the suites. It will contain a list of tests.

We will define a test case using the it() function. it() takes the first argument as a string that describes the test case. The second argument is a function where we will write the logic of the test.

Here I have a variable test whose value is true, and now we expect the test to be true, which is fine and the test case would be passed. Let’s run the test (maybe auto-run ) and see the result of this test case:

Text

Description automatically generated

snapshot of command “npm run test”

We can see that in the above result, it was passed. Let’s create a test case that would be failed. Let’s add the following test case in the same file. (Write the second it()):

it('Should be false', () **=>** { **const** test = 'Shubham'; expect(test).toBe('Ankit');});

Let’s run it and see the result:

Text

Description automatically generated

snapshot of command “npm run test”

You can see in the above snapshot, one is passed and the second one is failed. The assertion ( expect(test).toBe(‘Ankit’) ) is failed because:

Graphical user interface, text

Description automatically generated

The above two cases are just to show you how the test cases work.

Now let’s begin.

**Step 7. Write Test Cases for the Component Using Jest and Enzyme**

Now we will write our main test case to test ToggleComponent.js. Here we will test the behavior of the button click. The problem is, how can we achieve this? Not to worry; Enzyme will help us.

We need to install Enzyme and enzyme-adapter-react-16 (according to the React version; I am using React 16.X ). Check for the right Enzyme on GitHub, according to your React version.

To do this, run the following command:

npm install --save-dev enzyme enzyme-adapter-react-16

Let’s write the code for ToggleComponent to test behavior. Here we will write two test cases.

**Test case 1.**

Our ToogleComponent has the text “This will be toggled” inside a nested div. In the first test case, we will test to see that the component has the text “This will be toggled.”

Let’s write the test case for this. Delete all the code inside ToggleComponent.test.js and write the following:

Let’s understand the code first.

Below, the line is used to import React into this file:

import React from ‘react’;

In this line, we have imported Enzyme and its shallow function:

import Enzyme, { shallow } from ‘enzyme’;

Import the adapter from enzyme-adapter-react-16:

import Adapter from ‘enzyme-adapter-react-16’;

We want to test the ToogleComponent, so we will import this component:

import ToggleComponent from ‘./ToggleComponent’;

We need to configure Enzyme, so we need to do this:

Enzyme.configure({ adapter: new Adapter() });

In the code below, we use the define() and it() functions to write our test case logic:

describe(‘ToggleComponent’, () **=>** { it(‘should show the text’, () **=>** {  
 …… });});

The next line is used to get the instance of our component to test:

const toggleInstance = shallow(<ToggleComponent />);

**What “shallow()” does**

To test the React component, we need the instance of that component so that we can easily examine and write the code for the assertion. For this, Enzyme’s shallow() function helps a lot.

It takes a node as the first argument. It allows us to render the React component in the object and then in the memory, instead of in the DOM. This technique speeds up the process. It doesn’t require DOM. It wraps the object in the wrapper, and what the wrapper does is allow us to search through the element rendered using CSS. It returns the wrapper instance around the rendered output.

In the next line, we will search a div inside a div:

**const** element = toggleInstance.find(‘div div’);

In the next line, we write the assertion that expects the text to be “This will be toggled,” as we want:

expect(element.text()).toBe(‘This will be toggled’);

Let’s run the test cases using the command npm run test (maybe it automatically happens):

Text

Description automatically generated

Snapshot of command “npm run test”

You can see that our first test case is passed. (Make sure there was no space, and write the correct text in the code. You can also try altering the text and running the test case result.)

**Test case 2.**

We want to test the clicking of the button that causes the text to be hidden. So let’s write the code to test this scenario:

Let’s run the test cases using the command npm run test:

Text

Description automatically generated

Snapshot of command “npm run test”

You can see in the above result that both our test cases are passed. It means our component is perfect. For more clarity, you can remove the condition from  
{isShowText && <div>This will be toggled</div>} and make it simply <div>This will be toggled</div>. Run the test and see the result:

Text

Description automatically generated

Snapshot of command “npm run test”

It’s being failed. This means our test case is working fine.