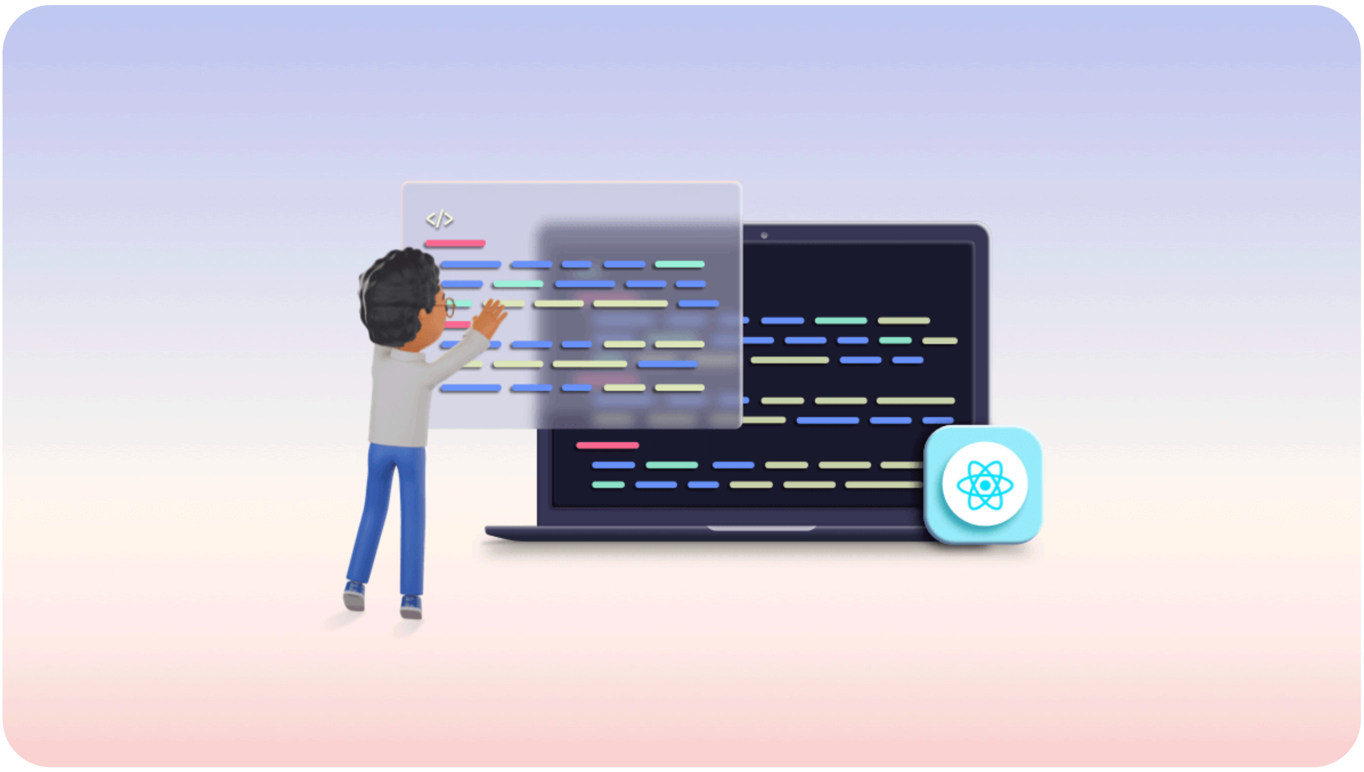
Unit Testing in React with Jest and Enzyme Frameworks



React is an open-source framework for building reusable UI components and apps. It is used by thousands of developers around the world to create complex and multifaceted applications. In this article, we will discuss unit testing in React and how to implement it using the Jest and Enzyme frameworks.

Let's get started!

Unit testing

- It is an essential and integral part of the software development process that helps us ensure the product's stability.
- It is the level of testing at which every component of the software is tested.

  - In a world of changing requirements, it helps us identify future breaks.

- Since bugs are found early, it helps us reduce the cost of bug fixes.

Prerequisites

- Visual Studio Code
- Node version 6 and above

Set up a React application

Follow these steps to set up a React application:

Step 1: Install the create-react-app **npm package** with the following command in the desired location.

```
npm install -g create-react-app
```

Step 2: Use the following commands to set up a basic **React** project.

```
create-react-app my-app --template typescript
cd my-app
npm install
```

React unit testing libraries

- Jest
- Enzyme

Jest

Jest is a JavaScript testing framework written by Facebook. It will help us make all our assertions.

Note: We need to install Jest before writing any test cases. When we run the **create-react-app** command, it will automatically install Jest in our React application.

Enzyme

Enzyme is a JavaScript testing utility for easily testing React components. It helps render React components in testing mode.

Enzyme installation

To get started with Enzyme, install it via npm with the following command.

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

Writing the first test case

Follow these steps to write a test case in your React application.

Step 1: We are going to render a simple button named **Click Me** using the following code.

```
import './App.scss';

class App extends React.Component<any, any> {
  constructor(props: any) {
    super(props);
    this.state = {
    };
}
  render() {
    return (
      <div>
        <button id="ClickMe" className="click-me">Click Me</button>
      </div>
    )
  }
};

export default App;
```

Step 2: Add the following code in the **App.test.tsx** file, which is the file where we write test cases.

Copy

```
import React from 'react'
import Enzyme, { shallow } from 'enzyme'
import Adapter from 'enzyme-adapter-react-16'
import App from './App'

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

describe('Test Case For App', () => {
  it('should render button', () => {
    const wrapper = shallow(<App />)
    const buttonElement  = wrapper.find('#ClickMe');
    expect(buttonElement).toHaveLength(1);
    expect(buttonElement.text()).toEqual('Click Me');
  })
})
```

In the previous code example, we stored the shallow renderer of the App component in the wrapper variable. We can access the length and text of the button inside the Button tag within the component's output, and we can also check if the text we passed matches the **toEqual** match function.

```
npm test
```

The test results will be shown like in the following screenshot.

Test with user interaction

Let's write a new test case that simulates **clicking a button** component and confirms its dependent actions.

Step 1: Add the following code in the **App.tsx** file.

```
import React, { Component } from 'react';
import './App.scss';

class App extends React.Component<any, any> {
  constructor(props: any) {
    super(props);
    this.state = {
}

ClickMe(){
  this.setState({
    ClickCount:this.state.ClickCount + 1
  });
}

  render() {
    return (
      <div>
        <button id="ClickMe" className="click-me" onClick={this.ClickMe}>Click Me</but
        <p>You clicked me :: {this.state.ClickCount}</p>
      </div>
    )
  }
};

export default App;
```

Step 2: Add the following code snippet in the **App.test.tsx** file.

Copy

```
import React from 'react'
import Enzyme, { shallow } from 'enzyme'
import Adapter from 'enzyme-adapter-react-16'
import App from './App'

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

describe('Test Case For App', () => {
  it('should render button', () => {
    const wrapper = shallow(<App />)
    const buttonElement  = wrapper.find('#ClickMe');
    expect(buttonElement).toHaveLength(1);
    expect(buttonElement.text()).toEqual('Click Me');
  }),

  it('increments count by 1 when button is clicked', () => {
    const wrapper = shallow(<App />);
    const buttonElement  = wrapper.find('#ClickMe');
    buttonElement.simulate('click');
    const text = wrapper.find('p').text();
    expect(text).toEqual('You clicked me :: 1');
```

Step 3: Run the **npm test** command to run the test cases in the desired application location. Then, we will get the test results as shown in the following screenshot.

Test case for state variable

Let's create a new test case to check the button's disabled/enabled state using the state variable.

Step 1: Add the below code snippet in the **App.tsx** file.

```
import React, { Component } from 'react';
import './App.scss';

class App extends React.Component<any, any> {
  constructor(props: any) {
    super(props);
    this.state = {
      ClickCount:0,
      IamDisabled : true
    };
    this.ClickMe = this.ClickMe.bind(this);
}

ClickMe(){
  this.setState({
    ClickCount:this.state.ClickCount + 1
  });
```

```
      <div>
        <button id="ClickMe" className="click-me" onClick={this.ClickMe}>Click Me</bu
        <p>You clicked me :: {this.state.ClickCount}</p>
        <button id=" IamDisabled " className="click-me" disabled={this.state.IamDisabl
      </div>
    )
  }
};

export default App;
```

Step 2: Add the following code snippet in the **App.test.tsx** file.

Copy

```
import React from 'react'
import Enzyme, { shallow } from 'enzyme'
import Adapter from 'enzyme-adapter-react-16'
import App from './App'

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

describe('Test Case For App', () => {
  it('should render button', () => {
    const wrapper = shallow(<App />)
    const buttonElement  = wrapper.find('#ClickMe');
    expect(buttonElement).toHaveLength(1);
    expect(buttonElement.text()).toEqual('Click Me');
  }),

  it('increments count by 1 when button is clicked', () => {
    const wrapper = shallow(<App />);
    const buttonElement  = wrapper.find('#ClickMe');
    buttonElement.simulate('click');
    const text = wrapper.find('p').text();
    expect(text).toEqual('You clicked me :: 1');
  });
})


describe('Test Case for App Page', () => {
    test('Validate Disabled Button disabled', () => {
        const wrapper = shallow(
            <App />
        );
```

Test with router

Let's assume we have a router option in our React application.

The following code shows how to deal with the router, label text, button availability, and enabled/disabled state using a state variable and an input box value passing event.

Copy

```
describe('Test Case for Create Customer Page', () => {
    test('Validate Create Customer Label render', () => {
      const wrapper = render(
        <MemoryRouter>
          <CreateCustomerComponent/>
       </MemoryRouter>
      );
      const linkElements = wrapper.queryAllByText('Create Customer');
      expect(linkElements).toHaveLength(1);
    });
});


describe('Test Case for Create Customer Page', () => {
    test('Validate Create Button render', () => {
      const wrapper = render(
        <MemoryRouter>
          <CreateCustomerComponent/>
       </MemoryRouter>
      );
      const linkElements = wrapper.queryAllByText('Create');
      expect(linkElements).toHaveLength(1);
    }),
    test('Validate Create Button disabled', () => {
        const {queryAllByText} = render(
        );
        expect(queryAllByText(/Create/i)[1].closest('button') as HTMLElement).toHaveAt
    });
});

describe('Test Case for Create Customer Page', () => {
    test('Validate Form fields', () => {
      const wrapper = shallow(
        <MemoryRouter>
          <CreateCustomerComponent/>
       </MemoryRouter>
      );
      console.log(wrapper.find('Enter first name'));
      wrapper.find('#first_name').simulate('change', { target: { name: 'name', value:
    })
});
```

Reference

You can also reference the following articles to learn more about the use of Jest and Enzyme to test a React app:

- How to unit test React applications with Jest and Enzyme
- Unit Testing in ReactJS using Jest and Enzyme
- Unit Testing in ReactJS GitHub demo

Explore the endless possibilities with Syncfusion's outstanding React UI components.

Try It Free

Conclusion

In this blog, we have learned how to do unit testing in React with the Jest and Enzyme frameworks. With this in your toolkit, you can check the stability of your

The Syncfusion React UI components library is the only suite you will ever need to build a charming web application. It contains over 65 high-performance, lightweight, modular, and responsive UI components in a single package.

For existing customers, the newest version is available for download from the License and Downloads page. If you are not yet a Syncfusion customer, you can try our 30-day free trial to check out the available features. Also, check out our samples on GitHub.

You can contact us through our support forums, support portal, or feedback portal. We are always happy to assist you!

If you like this blog post, we think you'll like the following articles too:

- How to Use Syncfusion's React Rich Text Editor with React Redux Form
- Create Progressive Web App with Syncfusion React UI Components
- Overview of Syncfusion React Gantt Chart Component
- React Succinctly

Tags:

React Testing Unit Testing Web Web Development

Be the first to get updates

< Syncfusion Blogs

Subscribe

MEET THE AUTHOR

Sangeetha Periyaiah

Sangeetha Periyaiah is a Software Engineer at Syncfusion for Consulting Projects and has been active in development since 2017. She is passionate about exploring new technologies. She is currently working as a full-stack developer in ASP.NET MVC, which uses SQL as the backend.

Syncfusion Blogs