**Creating React App :-**

* You need [Node.js](https://nodejs.org/en/download/) (v6 and above) and [npm](https://www.npmjs.com/get-npm" \t "_blank) installed on your machine.   
  You also need to install yarn since that’s what create-react-app uses.

npm install -g yarn

* Install create-react-app by running the following command in your terminal:

npm install -g create-react-app

* Then create your React app with the following command:

create-react-app counter-app

Initialized a git repository.

Success! Created counter-app at C:\TestData\_share\JS\00Training\Jest-Testing\counter-app

Inside that directory, you can run several commands:

yarn start

Starts the development server.

yarn build

Bundles the app into static files for production.

yarn test

Starts the test runner.

yarn eject

Removes this tool and copies build dependencies, configuration files

and scripts into the app directory. If you do this, you can’t go back!

* Once the application has been created, cd into the counter-app directory and run yarn start to launch the development server.

We suggest that you begin by typing:

cd counter-app

yarn start

C:\TestData\_share\JS\00Training\Jest-Testing\counter-app>yarn start

yarn run v1.15.2

$ react-scripts start

Starting the development server...

Compiled successfully!

You can now view counter-app in the browser.

Local: http://localhost:3000/

On Your Network: http://192.168.43.128:3000/

Note that the development build is not optimized.

To create a production build, use yarn build.

* Now open the counter-app folder in your favorite text editor and locate src/App.js. Change its contents to look like this:

Our React app has some initial state count which is set to zero, and a button that, once clicked, increments this count state through the increment function which simply adds 1 to the value of countand updates the application state.

Normally, we’d need to install and configure Jest before writing any tests, but since create-react-appships with Jest already installed, we don’t have to do any of that. We can jump straight into writing our first test.

If you look at the src/App.test.js, you will see that a test has already been written for us. It tests that the App component can render without crashing.

// src/App.test.js

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

it('renders without crashing', () => {

const div = document.createElement('div');

ReactDOM.render(<App />, div);

ReactDOM.unmountComponentAtNode(div);

});

Yarn test

* Let’s add a dummy test below this one in App.test.js:

// src/App.test.js

...

describe('Addition', () => {

it('knows that 2 and 2 make 4', () => {

expect(2 + 2).toBe(4);

});

});

A describe() function groups related tests together inside one test suite. It takes a name parameter, which should describe the component you’re testing, and a callback function where individual tests are defined with it.

You might see individual tests with test in some projects. To be sure, it and test are [one and the same thing](https://jestjs.io/docs/en/api.html#testname-fn-timeout). it is only an alias for test.

What you want to test is wrapped in a call to the expect() function, before calling what is termed a “matcher” function on it. In the above example, toBe() is the matcher function used. It checks that the value provided equals the value that the code within thWe expect() function produce

Before we begin writing our own tests, we need to add a few packages to our application for it to be able to test via Enzyme’s shallow renderer:

**yarn add enzyme enzyme-adapter-react-16 --dev**

You also need to create a setupTests.js file within your src folder that tells Jest and Enzyme what Adapters you will be making use of. create-react-app has been configured to run this file automatically before any of our tests, so that Enzyme is set up correctly.

*// src/setupTests.js*

import { configure } from 'enzyme';

import Adapter from 'enzyme-adapter-react-16';

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

**Testing React components with snapshots:**

Snapshot testing helps you check that the rendered output of a component is correct at all times. When you run a snapshot test, Jest renders the React component being tested and stores the output in a JSON file.

To use Jest’s snapshot feature, we need an additional package, [react-test-renderer](https://www.npmjs.com/package/react-test-renderer), which can be installed through yarn:

**yarn add react-test-renderer –dev**

Then import it at the top of App.test.js:

// src/App.test.js

import renderer from 'react-test-renderer';

...

Next, create a test below all the previously created ones:

// src/App.test.js

it('matches the snapshot', () => {

const tree = renderer.create(<App />).toJSON();

expect(tree).toMatchSnapshot();

});

The first time this test is run, there is no snapshot for this component so Jest creates it. You can inspect the contents of the snapshots inside the src/\_\_snapshots\_\_ directory.

You can see that the rendered output of the App component is saved in this file. The next time this test is run, Jest will confirm that the outputs are the same. We can demonstrate this concept by changing the rendered output of App slightly.

