Jest is an open-source test framework created by Facebook that has a great integration with React.js.

It includes a command line tool for test execution.

It also allows us to create mock functions with almost zero configuration and provides a really nice set of matchers that makes assertions easier to read.

it offers a really nice feature called “snapshot testing,” which helps us check and verify the component rendering result

**Why Jest**

* Very fast.
* Snapshot testing.
* Awesome interactive watch mode that reruns only tests that are relevant to your changes.
* Helpful fail messages.
* Simple configuration.
* Mocks and spies.
* Coverage report with a single command line switch.
* Active development.
* Jest is already bundled with JSDOM to enable DOM testing.
* It supports running tests in parallel; so it is faster.
* It supports [Promise](http://jestjs.io/docs/en/tutorial-async.html) too.
* It easily tests React apps.
* It provides out of the box code coverage.
* It is backed by Facebook.

ENZYME:

Enzyme provides a mechanism to mount and traverse React.js component trees. This will help us get access to its own properties and state as well as its children props in order to run our assertions.

Enzyme offers two basic functions for component mounting: shallow and mount. The shallow function loads in memory only the root component whereas mount loads the full DOM tree.

We’re going to combine Enzyme and Jest to mount a React.js component and run assertions over it.

### Mount, Shallow, Render

Import {mount, shallow, render} from ‘enzyme';

In order to have a component to test one of the above must be used, as in the example further above.

#### [Mounting](http://airbnb.io/enzyme/docs/api/mount.html)

* Full DOM rendering including child components
* Ideal for use cases where you have components that may interact with DOM API, or use React lifecycle methods in order to fully test the component
* As it actually mounts the component in the DOM .unmount() should be called after each tests to stop tests affecting each other
* Allows access to both props directly passed into the root component (including default props) and props passed into child components.
* Full rendering and it doesn’t need an environment like a “browser”. This is useful when you want to test the children with less overhead than mount.

#### [Shallow](http://airbnb.io/enzyme/docs/api/shallow.html)

* Renders only the single component, not including its children. This is useful to isolate the component for pure unit testing. It protects against changes or bugs in a child component altering the behavior or output of the component under test
* As of Enzyme 3 shallow components do have access to lifecycle methods by default
* Cannot access props passed into the root component (therefore also not default props), but can those passed into child components, and can test the effect of props passed into the root component. This is as with shallow(<MyComponent />), you're testing what MyComponent renders - not the element you passed into shallow.
* Shallow rendering is useful to constrain yourself to testing a component as a unit, and to ensure that tests aren’t indirectly asserting the behavior of child components.

#### [Render](http://airbnb.io/enzyme/docs/api/render.html)

* Renders to static HTML, including children
* Does not have access to React lifecycle methods
* Less costly than mount but provides less functionality
* Full DOM rendering is ideal for use cases where you have components that may interact with DOM APIs. Full rendering actually mounts the component in the DOM. This is the only way to test componentDidMount and componentDidUpdate.

<https://medium.freecodecamp.org/components-testing-in-react-what-and-how-to-test-with-jest-and-enzyme-7c1cace99de5>

Unit test: **actions**

**export** **function** **addTodo**(text) {

**return** {

type: 'ADD\_TODO',

text

}

}

describe('actions', () => {

it('should create an action to add a todo', () => {

**const** text = 'Finish docs'

**const** expectedAction = {

type: types.ADD\_TODO,

text

}

expect(actions.addTodo(text)).toEqual(expectedAction)

})

})

**Reducer**

export default combineReducers({

todos,

visibilityFilter

})

describe('root reducer', () => {

it('should combine all reducers', () => {

expect(reducer({}, { type: '@@INIT' })).toEqual({

todos: [],

visibilityFilter: 'SHOW\_ALL'

});

});

});

<https://medium.com/@netxm/testing-redux-reducers-with-jest-6653abbfe3e1>

**dispatcher:**

describe('selectAvatar', () => {

test('Dispatches the correct action and payload', () => {

const expectedActions = [

{

'payload': 1,

'type': 'select\_avatar',

},

];

store.dispatch(selectActions.selectAvatar(1));

expect(store.getActions()).toEqual(expectedActions);

});

});