Testing guides

Contents:

Introduction

This wiki page contains details regarding how to set up tests, tips for using JEST, enzyme, and snapshots.

Setup

1. Firstly, ensure that yarn is installed and you are testing a version of the project code that runs without crashing. `yarn install setenv CI true yarn test`
2. It is best to create the test file in the same directory as the JavaScript class your testing. For example, if you are testing SelectButton.js which is in the components directory, then name your test file SelectButton.test.js in that same directory. It is important that you include **.test** in the filename as that lets yarn know that is a testing suite.
3. Import all the necessary dependencies. Usually we test for html changes and states. Here is a good set of dependencies to import: `javascript import React from "react"; import ReactDOM from "react-dom"; import SelectButton from "./SelectButton"; import { shallow } from "enzyme";`
4. When we test, we state what the test does in string inside the it() function. Here is an example of testing whether the Select Button component renders without crashing: `javascript it("renders button without crashing", () => { const div = document.createElement("div"); ReactDOM.render(<SelectButton />, div); ReactDOM.unmountComponentAtNode(div); });`
5. Final step of the setup process is to save what you wrote. Ensure that your tests is of correct JavaScript grammar, then in the root directory where yarn should be installed, enter the command: `yarn test`
6. Here is an extract of an example test file, based on SelectButton.test.js. Your test code should look similar to this:

“‘javascript import React from ”react”; import ReactDOM from ”react-dom”; import SelectButton from ”./SelectButton”; import { shallow } from ”enzyme”;

it(”renders button without crashing”, () => { const div = document.createElement(”div”); ReactDOM.render(, div); ReactDOM.unmountComponentAtNode(div); }); “‘

Testing for rendering HTML

1. We need to test for front end changes too. To do this we need to use snapshots. In tests when you render components, it exports a snapshot reference of what the HTML code should look like. In this example, we check if the component intialises correctly: `javascript it("renders button correctly as correct name 'Projects'", () => { const div = document.createElement("div"); ReactDOM.render(<SelectButton name="Projects" />, div); expect(div).toMatchSnapshot(); ReactDOM.unmountComponentAtNode(div); });` Notice how we added the extra line to expect the select button div to match the snapshot?
2. Essentially we mainly only need to test for the initial state because when we render components, we just make a new instance of that component. Before we can set sail into the happy sunset, we need to make sure that the snapshot that was export is correct. There should be a directory inside of the directory you are testing in. This directory will be named: `__snapshots__` If the html code looks wrong, then you need to revise your tests. One reason I found that may apply with your issue is the order of calling methods. Here are two different tests: `javascript it("renders button correctly as correct name 'Projects'", () => { const div = document.createElement("div"); ReactDOM.render(<SelectButton name="Projects" />, div); ReactDOM.unmountComponentAtNode(div); expect(div).toMatchSnapshot(); }); javascript it("renders button correctly as correct name 'Schools'", () => { const div = document.createElement("div"); ReactDOM.render(<SelectButton name="Schools" />, div); expect(div).toMatchSnapshot(); ReactDOM.unmountComponentAtNode(div); });` In the first test, we are unmounting the component before the snapshot is compared, which is *BAD*. Notice in the second test we unmount after comparing the snapshot. It is just good practice to unmount the component when you are done with the test - Friendly advice from Rhys.

Testing for Non-HTML Properties

1. Some aspects of a react component are not captured in the HTML code. For example, the SelectButton has a property called *checked* which determines if the button is selected or not by the end user. This is not shown

in the HTML code, but is shown in the react dev tools code. Therefore, to test for these scenarios, we would use *enzyme*.

2. In this example reference SelectButton.test.js, this test uses enzyme to capture the contents of the component to then compare with a JavaScript assertion statement: `javascript it("should have correct checked value of true", () => { const wrapper = shallow(<SelectButton id="showSchools" isChecked={true} />); expect(wrapper.find("input").props().checked).toBe(true); );` This ensures that the hidden property of checked has the correct value based on the parameters it was given. It makes a shallow copy of the component and uses **find()** to get the input tag, then uses **props()** to get the list of properties of input. This test cares about **checked** property, which is shown in the test.

3. If you need to test a component that holds other components inside it you will need to use `mount()` rather than `shallow()`. For Views.js, when expecting the SelectButton to update it's properties, you need to call `wrapper.update()` after any changes made for them to take effect.

Disclaimer

Unfortunately this is all the knowledge I have for testing. There is heaps of content online. However, feel free to update this wiki page if you find any useful tips. I will update this page myself if I do find anything. If in doubt, **shout out** and look at tests that already exists.

Contributors

- Kyle Claudio
- Will Pearson