**Unit Testing Behavior of React Components with Test-Driven Development (TDD)**

**Testing**

* [**Enzyme**](https://github.com/airbnb/enzyme/)**:** a JavaScript Testing utility for React that makes it easier to assert, manipulate, and traverse your React Components’ output.
* [**Jest**](https://facebook.github.io/jest/)**:** Delightful JavaScript testing used by Facebook to test all JavaScript code including React applications.
* [**react-testing-library**](https://github.com/kentcdodds/react-testing-library)**:** 🐐 Simple and complete React DOM testing utilities that encourage good testing practices.
* [**React-unit**](https://github.com/pzavolinsky/react-unit)**:** a lightweight unit test library for ReactJS with very few (js-only) dependencies.
* [**Skin-deep**](https://github.com/glenjamin/skin-deep)**:** Testing helpers for use with React’s shallowRender test utils.
* [**Unexpected-react**](https://github.com/bruderstein/unexpected-react/)**:** Plugin for the [unexpected](https://unexpected.js.org/) assertion library that makes it easy to assert over your React Components and trigger events.

Testing components that are not purely functional and are responsible for **behavior** isn’t difficult, but there aren’t as many resources on the web that describe how to do this. This article will show how to unit test these more complex React components. We will use a test-driven development (TDD) approach, writing our tests first.

### Example Use Case

Say we have a loading indicator component that we want to display when a request for data is in flight. If the data loads really quickly, we want to avoid briefly flashing the loading indicator for a split second. Our product manager has advised us that it would provide a smoother user experience if the user saw nothing at all for that split second before the data loads. So, we’d like our loading indicator component to wait 200ms before actually rendering the loading indicator. If less than 200ms have elapsed, we want to render nothing.

### Tools

For this article, we will be using the [Jest](https://facebook.github.io/jest/) testing platform. [Enzyme](http://airbnb.io/enzyme/) is a JavaScript testing utility for React that provides a way to render a React component in our unit tests and make assertions about its output and behavior. When using Enzyme with React, we need an adapter corresponding to the version of React we are running. In our example, we will use [enzyme-adapter-react-16](https://github.com/airbnb/enzyme/tree/master/packages/enzyme-adapter-react-16)

### Setup

The example code for this article is available in [this repo](https://github.com/bruceharris/react-unit-testing-example). Each commit in the repo maps to a step in this article. The repo uses [create-react-app](https://github.com/facebookincubator/create-react-app/) which is already instrumented with a Jest test runner. You can clone the example repo and follow along with the steps here.

Our first step is to get a test running and failing. We’ll then implement the code to make that test pass.

Use [npm](https://www.npmjs.com/get-npm" \t "_blank) or [Yarn](https://yarnpkg.com/en/) to install these packages if your project isn’t already using them.

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

Follow [these steps](https://github.com/airbnb/enzyme/tree/master/packages/enzyme-adapter-react-16#installation) to configure Enzyme to use the relevant React adapter in your project. This step is implemented in [this commit](https://github.com/bruceharris/react-unit-testing-example/commit/c58ddd4e96ffe8327c793bafcf39914c72914620) in the example repo.

To run the tests in interactive watch mode in the example repo you’ll need to install [Watchman](https://facebook.github.io/watchman/)

Run the test watcher:

npm test

If you prefer to run the tests manually without watching, then Watchman isn’t necessary.

CI**=**true npm test

We haven’t added any tests yet, but create-react-app provides one basic test. At this point you should see something like this when you run the tests:

PASS src/App.test.js  
 ✓ renders without crashing (24ms)

Test Suites: 1 passed, 1 total  
Tests: 1 passed, 1 total  
Snapshots: 0 total  
Time: 0.612s  
Ran all test suites.

### Requirements

Our component is going to take a single boolean prop: isLoading.

When isLoading is false, we will render the component’s children.

When isLoading is true…

* If 200ms have elapsed, we will display text to indicate that we are “loading”.
* If 200ms have not yet elapsed, we will display nothing.

### Implementation Steps

Let’s start our TDD cycle.

#### Create Component and Initial Failing Unit Test

Our next step is to create a file that exports a component and a unit test for it. Once we have a failing unit test, we can add the target behavior to our component and make it pass. This step is implemented in [this commit](https://github.com/bruceharris/react-unit-testing-example/commit/46ff6b4b417d91583dda3f1b0c565ba80732ff17) in the example repo.

Let’s start with the simple case that renders the children when isLoading is false. We’ll create a file for our component, starting with a class-based component that renders nothing:

|  |
| --- |
| import React, { Component } from 'react'; |
|  | import PropTypes from 'prop-types'; |
|  |  |
|  | export default class LoadingIndicator extends Component { |
|  | render() { |
|  | return null; |
|  | } |
|  | } |
|  |  |
|  | LoadingIndicator.propTypes = { |
|  | isLoading: PropTypes.bool.isRequired, |
|  | }; |

Let’s create a file for our unit test. We’ll add a test to validate that childrenare rendered when isLoading is false.

|  |
| --- |
| import React from 'react'; |
|  | import { mount } from 'enzyme'; |
|  | import LoadingIndicator from './LoadingIndicator' |
|  |  |
|  | describe('LoadingIndicator', () => { |
|  | describe('when isLoading is false', () => { |
|  | it('should render children', () => { |
|  | const wrapper = mount( |
|  | <LoadingIndicator isLoading={false}> |
|  | <div>ahoy!</div> |
|  | </LoadingIndicator> |
|  | ); |
|  | expect(wrapper.html()).toEqual('<div>ahoy!</div>'); |
|  | wrapper.unmount(); |
|  | }); |
|  | }); |
|  | }); |

When we don’t need to be concerned with lifecycle methods or children, Enzyme’s [shallow](http://airbnb.io/enzyme/docs/api/shallow.html)() function allows us to isolate the component we want to test, ensuring that children are not rendered in our unit test. Since we want to validate that the component’s children are rendered, we use Enzyme’s [mount()](http://airbnb.io/enzyme/docs/api/mount.html) function to mount the component in the DOM in our test-running environment so we can make assertions about it.

We don’t want to leave our test component mounted in the test environment’s DOM, so we call [unmount()](http://airbnb.io/enzyme/docs/api/ShallowWrapper/unmount.html) to clean up after our assertions.

Our test runner should now show a failing test that looks something like this:

PASS src/components/LoadingIndicator.test.js  
FAIL src/components/LoadingIndicator.test.js  
 ● LoadingIndicator › when isLoading is false › should render children

expect(received).toEqual(expected)  
   
 Expected value to equal:  
 "<div>ahoy!</div>"  
 Received:  
 null  
   
 Difference:  
   
 Comparing two different types of values. Expected string but received null.  
   
 at Object.it (src/components/LoadingIndicator.test.js:13:30)  
 at Promise (<anonymous>)  
 at Promise.resolve.then.el (node\_modules/p-map/index.js:46:16)  
 at <anonymous>

Test Suites: 1 failed, 1 passed, 2 total  
Tests: 1 failed, 1 passed, 2 total  
Snapshots: 0 total  
Time: 0.556s, estimated 1s  
Ran all test suites

If you’re seeing a test failure in your project that doesn’t look similar, ensure you’ve [correctly configured Enzyme](https://github.com/airbnb/enzyme/tree/master/packages/enzyme-adapter-react-16#installation), and that your [Jest testEnvironment is the default jsdom.](http://facebook.github.io/jest/docs/en/configuration.html#testenvironment-string)

### Render Children to Get Failing Test to Pass

This step is implemented in [this commit](https://github.com/bruceharris/react-unit-testing-example/commit/f673f0527ebd5e228cfb6926cb8b9a22daf5638f) in the example repo.

Getting this test to pass is super easy; we simply change our render method to return this.props.children instead of null. This obviously isn’t the correct implementation in light of our other requirements, but in the spirit of TDD, we write the minimum amount of code to get the test to pass.

|  |
| --- |
| import React, { Component } from 'react'; |
|  | import PropTypes from 'prop-types'; |
|  |  |
|  | export default class LoadingIndicator extends Component { |
|  | render() { |
|  | return this.props.children; |
|  | } |
|  | } |
|  |  |
|  | LoadingIndicator.propTypes = { |
|  | isLoading: PropTypes.bool.isRequired, |
|  | }; |

We’ll also add code to App.js in the example app so we can see our component render in the browser.

|  |
| --- |
| class App extends Component { |
|  | render() { |
|  | return ( |
|  | <div className="App"> |
|  | <header className="App-header"> |
|  | <img src={logo} className="App-logo" alt="logo" /> |
|  | <h1 className="App-title">Welcome to React</h1> |
|  | </header> |
|  | <p className="App-intro"> |
|  | To get started, edit <code>src/App.js</code> and save to reload. |
|  | </p> |
|  | <LoadingIndicator isLoading={false}> |
|  | <div>ahoy!</div> |
|  | </LoadingIndicator> |
|  | </div> |
|  | ); |
|  | } |
|  | } |

Run the example app with npm start— this will run a local web server on port 3000 and open a browser window pointing to it.

#### Add Test for Displaying Nothing When 200ms Haven’t Elapsed

This step is implemented in [this commit](https://github.com/bruceharris/react-unit-testing-example/commit/dad3332ab55413223977b6eb286f2c6bcb59c608) in the example repo.

Let’s move on to the next easiest part of our spec. When isLoading is true, if 200ms have not yet elapsed, we should display nothing.

Let’s add the following test:

|  |
| --- |
| describe('when isLoading is true', () => { |
|  | describe('given 200ms have not yet elapsed', () => { |
|  | it('should render nothing', () => { |
|  | const wrapper = mount( |
|  | <LoadingIndicator isLoading={true}> |
|  | <div>ahoy!</div> |
|  | </LoadingIndicator> |
|  | ); |
|  | expect(wrapper.html()).toBe(null); |
|  | wrapper.unmount(); |
|  | }); |
|  | }); |
|  | }); |

We should now see that test failing:

FAIL src/components/LoadingIndicator.test.js  
 ● LoadingIndicator › when isLoading is true › given 200ms have not yet elapsed › should render nothing

expect(received).toBe(expected)  
   
 Expected value to be (using ===):  
 null  
 Received:  
 "<div>ahoy!</div>"  
   
 Difference:  
   
 Comparing two different types of values. Expected null but received string.  
   
 at Object.it (src/components/LoadingIndicator.test.js:26:32)  
 at Promise (<anonymous>)  
 at Promise.resolve.then.el (node\_modules/p-map/index.js:46:16)  
 at <anonymous>  
 at process.\_tickCallback (internal/process/next\_tick.js:188:7)

PASS src/App.test.js

Test Suites: 1 failed, 1 passed, 2 total  
Tests: 1 failed, 2 passed, 3 total  
Snapshots: 0 total  
Time: 1.495s  
Ran all test suites.

## [Shallow Rendering](https://github.com/airbnb/enzyme/blob/master/docs/api/shallow.md)

import React from 'react';

import { expect } from 'chai';

import { shallow } from 'enzyme';

import sinon from 'sinon';

import MyComponent from './MyComponent';

import Foo from './Foo';

describe('<MyComponent />', () => {

it('renders three <Foo /> components', () => {

const wrapper = shallow(<MyComponent />);

expect(wrapper.find(Foo)).to.have.lengthOf(3);

});

it('renders an `.icon-star`', () => {

const wrapper = shallow(<MyComponent />);

expect(wrapper.find('.icon-star')).to.have.lengthOf(1);

});

it('renders children when passed in', () => {

const wrapper = shallow((

<MyComponent>

<div className="unique" />

</MyComponent>

));

expect(wrapper.contains(<div className="unique" />)).to.equal(true);

});

it('simulates click events', () => {

const onButtonClick = sinon.spy();

const wrapper = shallow(<Foo onButtonClick={onButtonClick} />);

wrapper.find('button').simulate('click');

expect(onButtonClick).to.have.property('callCount', 1);

});

});

Read the full [API Documentation](https://github.com/airbnb/enzyme/blob/master/docs/api/shallow.md)

## [Full DOM Rendering](https://github.com/airbnb/enzyme/blob/master/docs/api/mount.md)

import React from 'react';

import sinon from 'sinon';

import { expect } from 'chai';

import { mount } from 'enzyme';

import Foo from './Foo';

describe('<Foo />', () => {

it('allows us to set props', () => {

const wrapper = mount(<Foo bar="baz" />);

expect(wrapper.props().bar).to.equal('baz');

wrapper.setProps({ bar: 'foo' });

expect(wrapper.props().bar).to.equal('foo');

});

it('simulates click events', () => {

const onButtonClick = sinon.spy();

const wrapper = mount((

<Foo onButtonClick={onButtonClick} />

));

wrapper.find('button').simulate('click');

expect(onButtonClick).to.have.property('callCount', 1);

});

it('calls componentDidMount', () => {

sinon.spy(Foo.prototype, 'componentDidMount');

const wrapper = mount(<Foo />);

expect(Foo.prototype.componentDidMount).to.have.property('callCount', 1);

Foo.prototype.componentDidMount.restore();

});

});

Read the full [API Documentation](https://github.com/airbnb/enzyme/blob/master/docs/api/mount.md)

## [Static Rendered Markup](https://github.com/airbnb/enzyme/blob/master/docs/api/render.md)

import React from 'react';

import { expect } from 'chai';

import { render } from 'enzyme';

import Foo from './Foo';

describe('<Foo />', () => {

it('renders three `.foo-bar`s', () => {

const wrapper = render(<Foo />);

expect(wrapper.find('.foo-bar')).to.have.lengthOf(3);

});

it('renders the title', () => {

const wrapper = render(<Foo title="unique" />);

expect(wrapper.text()).to.contain('unique');

});

});