Extracted from:

Test-Driven React

Find Problems Early, Fix Them Quickly, Code with Confidence

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Test-Driven React

Find Problems Early, Fix Them Quickly, Code with Confidence

Trevor Burnham
edited by Jacquelyn Carter

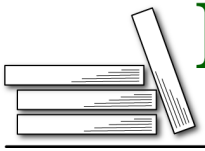
Test-Driven React

Find Problems Early, Fix Them Quickly, Code with Confidence

Trevor Burnham

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Susan Conant
Development Editor: Jacquelyn Carter
Copy Editor: Jasmine Kwityn
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

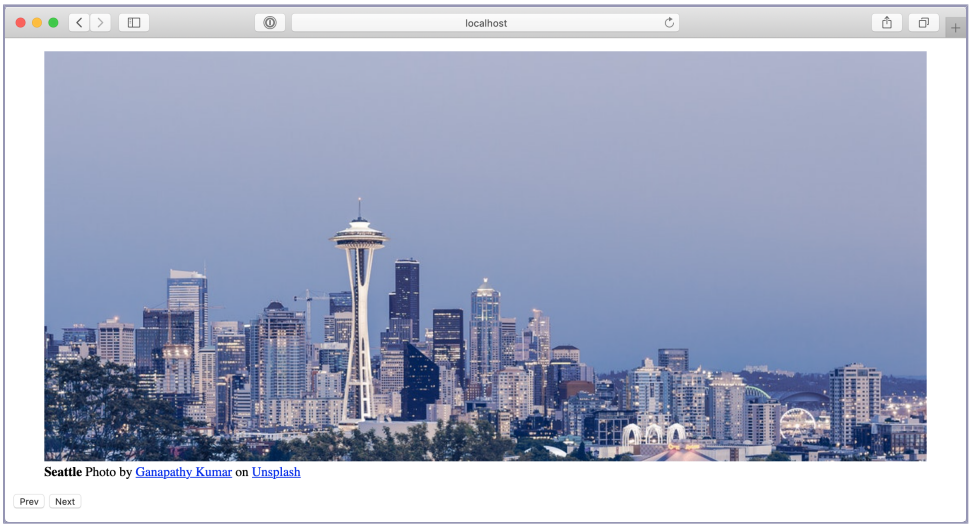
For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-646-4
Book version: P1.0—June 2019

Testing Simple Components with Enzyme

As our project for the rest of the book, we're going to build a carousel. A carousel is a widget that shows a series of images to the user one at a time. The user has the ability to move between adjacent images in the series. The carousel can also auto-advance, like a slideshow. By the end of the book, our carousel will look like this:



Since this is fairly complex, we're going to implement it as a primary component called `Carousel` with two secondary components:

1. `CarouselSlide`, an image shown in the carousel
2. `CarouselButton`, a button that lets the user control the slides

Let's start by writing some tests for the simplest component: `CarouselButton` is just going to render a `<button>`. (Why componentize it at all? Because we're going to add styling to that `<button>` in the next chapter.) We'll make assertions about the component using a library called Enzyme.

Getting Started with Enzyme

Airbnb's Enzyme has become the most popular library for testing React components. Enzyme is especially good at rendering a single component in isolation, a technique known as "shallow rendering," and letting you see how changes in the component's props and state cause its render tree to change. With shallow rendering, other components in the render tree are treated as black boxes: you can see what props they receive, but not their output. We'll be using shallow rendering for all of the React component tests in this book.

Enzyme vs. react-testing-library

As this book was written, a solid competitor to Enzyme emerged and gained traction: react-testing-library.[a] Whereas Enzyme is a Swiss Army knife for testing React components, react-testing-library is focused on a single approach: rendering components to the DOM and making assertions about that DOM tree.

Both Enzyme and react-testing-library have their pros and cons. This book advocates testing individual components in isolation, which isn't possible with react-testing-library. However, react-testing-library's approach is more conceptually straightforward. Furthermore, Enzyme's development has lagged somewhat; as of this writing, its shallow rendering mode lacks support for some cutting-edge React features, such as the new context API introduced in React 16.3.

If you're starting a new React project, do your research. Take a close look at both Enzyme and react-testing-library and decide which is the best fit for you.

a. https://github.com/kentcdodds/react-testing-library

You'll need to install Enzyme, plus the "adapter" that lets it plug into the version of React you're using:

```
$ npm install --save-dev enzyme@3.8.0 enzyme-adapter-react-16@1.7.1
+ enzyme@3.8.0
+ enzyme-adapter-react-16@1.7.1
```

Delete the hello.test.js file. Then create a quick implementation of CarouselButton:

```
// src/CarouselButton.js
import React from 'react';

const CarouselButton = () => <button />;

export default CarouselButton;
```

CarouselButton is defined as a function that returns an empty <button>. Simple as it is, this is a valid React component. Note the use of capitalization: JSX treats <Uppercase /> as an instance of a *component* named Uppercase, and <lowercase /> as an instance of a *DOM element* named lowercase.

Now put this test in place:

```
// src/tests/CarouselButton.test.js
import React from 'react';
import { configure, shallow } from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';
import CarouselButton from '../CarouselButton';

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

describe('CarouselButton', () => {
```

```
  it('renders a <button>', () => {
3   const wrapper = shallow(<CarouselButton />);
    expect(wrapper.type()).toBe('button');
  });
});
```

❶ Even though our code never references React directly, we need to import it in both the component module and the test module because both use JSX expressions, which compile to React.createElement statements.

❷ Enzyme needs us to pass a React version-appropriate adapter to its configure function before we can use it, so we do that at the top of the test file. Later in this chapter, we'll move that setup code elsewhere to avoid duplicating it across all test files.

❸ Enzyme's shallow() method returns a shallow wrapper.[7]

If you run Jest, the test output should be all-green. However, savvy React developers will notice that this isn't a very useful CarouselButton implementation yet—there's no way to put content inside of the <button />! So let's get into full TDD mode, after we commit using the gitmoji for a work in progress:

```
:construction: Starting work on CarouselButton
```

Working with Props

Currently, CarouselButton renders an empty <button> element, which isn't very useful. We need to add support for setting the children of the <button> element, which will be the text that the user sees. Add a test for that to the existing describe() block:

```
// src/tests/CarouselButton.test.js
...
➤ it('passes `children` through to the <button>', () => {
➤   const text = 'Button text';
➤   const wrapper = shallow(<CarouselButton>{text}</CarouselButton>);
➤   expect(wrapper.prop('children')).toBe(text);
➤ });
...
```

The wrapper.prop(propName) method returns the value of the prop with the given name. Remember that wrapper, in this case, represents the <button> rendered by CarouselButton. Currently that button is rendered without children, failing the test. To fix that, add some prop-passing logic to the component:

```
// src/CarouselButton.js
...
```

7. https://github.com/airbnb/enzyme/blob/master/docs/api/shallow.md

```
const CarouselButton = ({ children }) => <button>{children}</button>;
...
```

When a component is defined as a function, that function receives the component instance's `props` object as the first argument. The argument list `({ children })` uses ES6's destructuring syntax to extract `props.children` as `children`, which is then passed through to the rendered `<button>`. Any other props are ignored.

One subtle point here: the JSX code

```
<div>{children}</div>
```

is equivalent to

```
<div children={children} />
```

That is, anything inserted between an opening tag and a closing tag is treated as that element's `children` prop in JSX. (If `children` is set in both places, the value between the tags has precedence.)

With that component change, your tests should be in the green! However, the linter isn't happy:

```
'children' is missing in props validation (react/prop-types)
```

Since you're using the recommended ESLint config from the React plugin, you're going to see a lot of constructive criticism like this. In this case, it wants you to use propTypes to declare what type of prop children is. propTypes serve two purposes. First, they're useful for documentation. Just looking at a component's propTypes often gives a good sense of its feature set. Second, they provide validation when React is running in development mode. If, for instance, you declared that children had to be a React element and a developer passed in a string instead, that developer would get a console warning.

To declare propTypes, you'll need a package called prop-types:

```
$ npm install prop-types@15.7.2
+ prop-types@15.7.2
```

Then import that package and attach a propTypes object to the component:

```
// src/CarouselButton.js
import React from 'react';
import PropTypes from 'prop-types';

const CarouselButton = ({ children }) => <button>{children}</button>;

CarouselButton.propTypes = {
  children: PropTypes.node.isRequired, (1)
};

export default CarouselButton;
```

(1) The node type means that children can be either a React element or a primitive, such as a string. And since we can reasonably expect every button to have children, it's marked as isRequired, meaning that null and undefined values are unacceptable. Prop types are strictly a development aid, and are ignored by React in production mode. You can learn more from the React docs.[8]

If you run Jest from the console, you'll notice that there's a console error, though it doesn't affect the results:

```
$ npx jest

 PASS  src/tests/CarouselButton.test.js
  CarouselButton
    ✓ renders a <button> (8ms)
    ✓ passes `children` through to the <button> (1ms)

  console.error node_modules/prop-types/checkPropTypes.js:19
    Warning: Failed prop type: The prop `children` is marked as required in
    `CarouselButton`, but its value is `undefined`.
        in CarouselButton
```

8. https://reactjs.org/docs/typechecking-with-proptypes.html

```
Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.116s
Ran all test suites.
```

It's a good idea to always provide required props in tests, to better reflect realistic component usage. So let's provide children to the CarouselButton element in both tests. To avoid duplication, extract the shallow wrapper creation logic to a separate block called beforeEach():

```
// src/tests/CarouselButton.test.js
...
describe('CarouselButton', () => {
  const text = 'Button text';
  let wrapper;

  beforeEach(() => {
    wrapper = shallow(<CarouselButton>{text}</CarouselButton>);
  });

  it('renders a <button>', () => {
    expect(wrapper.type()).toBe('button');
  });

  it('passes `children` through to the <button>', () => {
    expect(wrapper.prop('children')).toBe(text);
  });
});
```

❶ beforeEach() executes before each test in the parent describe() block. Since there are two tests, beforeEach() will execute twice, producing two independent instances of wrapper. Giving each test its own wrapper instance ensures that no tests fail due to changes an earlier test made to its wrapper.

Now that the console error is gone, let's think ahead to what other functionality CarouselButton needs to support. In addition to passing children through to the button, we'll want to support passing an onClick event handler through. We'll also want to support passing a className prop through for styling. And we'll want to support data- attributes, too. Come to think of it, what if we just pass *every* prop through?

This is actually a very common practice in React, and a sensible one. Add another test to the existing describe() block with more prop assertions:

```
// src/tests/CarouselButton.test.js
...
it('passes other props through to the <button>', () => {
  const onClick = () => {};
  const className = 'my-carousel-button';
  const dataAction = 'prev';
  wrapper.setProps({ onClick, className, 'data-action': dataAction }); (1)
  expect(wrapper.prop('onClick')).toBe(onClick);
  expect(wrapper.prop('className')).toBe(className);
  expect(wrapper.prop('data-action')).toBe(dataAction);
});
...
```

(1) wrapper.setProps(props) simulates props being passed into the wrapped React element after the initial render, making it useful for testing lifecycle methods like componentWillReceiveProps() and componentDidUpdate(). The props passed in with setProps() are merged into the existing props object.

To satisfy the new test, update CarouselButton to pass all props through:

ch3/src/CarouselButton.js

```
import React from 'react';
import PropTypes from 'prop-types';

const CarouselButton = props => <button {...props} />; (1)

CarouselButton.propTypes = {
  children: PropTypes.node.isRequired,
};

export default CarouselButton;
```

(1) {...props} is the JSX spread operator. It's equivalent to passing each prop in the props object through individually. That includes children, since the tag itself has no children.

By the way, now that children is no longer referenced directly, ESLint no longer requires a propTypes declaration for children. Even so, let's keep the declaration in place. It's a useful reminder that the <button> should always have text.

And we're back in the green! This is a good point for a commit:

```
:sparkles: Initial implementation of CarouselButton
```

Next, we will clear out the Enzyme configuration boilerplate from the top of the test file.