React Testing

Download Demo Code <../react-testing-demo.zip>

Goals

- Describe what ***react-testing-library*** is and how it works
- Write smoke tests
- Write snapshot tests
- Write tests to mimic user interaction with our applications

Testing in React

- React can use any testing framework
- ***create-react-app*** ships with ***jest*** and ***react-testing-library***
- ***npm test*** is set up to find & run jest tests
  - Files like *.test*, *.spec*, or in ***__tests__*** folder
  - At Rithm, we prefer `*.test.js` in same folder as component
- ***npm test*** runs all tests; can optionally specify a file

React Testing Library

- React Testing Library <https://github.com/testing-library/react-testing-library> is a popular testing add-on for React
- Guiding principle: "The more your tests resemble the way your software is used, the more confidence they can give you."
- Tests focus on mimicking how people interact with your site, not implementation details

Types of Tests

Smoke Tests

- Does the component render, or does it blow up (like a smoking box)?

Snapshot Tests

- Does the component's rendered HTML render in the way you expect?

Smoke Tests

Smoke Tests Working Example

demo/testing-demo/src/App.js

```
import React from "react";
import Counter from "./Counter";

function App() {
  return (
    <div>
      Hello, I'm an App!
      <Counter />
    </div>
  );
}

export default App;
```

demo/testing-demo/src/App.test.js

```
import React from "react";
import { render } from "@testing-library/react";
import App from "./App";

it("renders without crashing", function() {
  render(<App />);
});
```

```
$ npm test App.test.js

PASS src/App.test.js
  ✓ renders without crashing (4ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.283s
Ran all test suites matching /App.test.js/i.
```

Render

React Testing Library's ***render*** method…

- Creates a <div>
- Renders your JSX into the div
- Returns an object of methods (more on this later)

Smoke Tests Broken Example

demo/testing-demo/src/BrokenComponent.js

```
import React from "react";

function BrokenComponent(props) {
  return (
    <div>
      <p>Hello, I'm an BrokenComponent!</p>
      <p>Here are some numbers:</p>
      <p>{props.favNum}</p>
      <p>{props.favNum++}</p>
      <p>{props.favNum++}</p>
    </div>
  );
}

BrokenComponent.defaultProps = {
  favNum: 42
};

export default BrokenComponent;
```

demo/testing-demo/src/BrokenComponent.test.js

```
import React from "react";
import { render } from "@testing-library/react";
import BrokenComponent from "./BrokenComponent";

it("renders without crashing", function() {
  render(<BrokenComponent />);
});
```

What's the problem?

```
$ npm test BrokenComponent.test.js

FAIL  src/BrokenComponent.test.js
 × renders without crashing (33ms)

 ● renders without crashing

  TypeError: Cannot assign to read only property 'favNum' of object
  '#<Object>'

     7 |       <p>Here are some numbers!</p>
     8 |       <p>{props.favNum}</p>
  >  9 |       <p>{props.favNum++}</p>
       |           ^
    10 |       <p>{props.favNum++}</p>
    11 |     </div>
    12 |   );

    at BrokenComponent.render (src/BrokenComponent.js:9:11)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:   0 total
Time:        3.073s
Ran all test suites matching /BrokenComponent.test.js/i.
```

Snapshot Tests

Smoke tests are easy to write, but only test that component renders.

Snapshot tests are also easy, but go a bit further: did rendering change?

Render Revisited

React Testing Library's ***render*** method…

- Returns an object of methods
- One method is called ***asFragment***
- ***asFragment*** returns the underlying DOM structure of the component
  - very useful for snapshot tests!

Snapshot Tests Example

demo/testing-demo/src/FixedComponent.js

```
import React from "react";

function FixedComponent(props) {
  return (
    <div>
      <p>Hello, I'm a FixedComponent!</p>
      <p>Here are some numbers:</p>
      <p>{props.favNum}</p>
      <p>{props.favNum + 1}</p>
      <p>{props.favNum + 2}</p>
    </div>
  );
}

FixedComponent.defaultProps = {
  favNum: 42
};

export default FixedComponent;
```

demo/testing-demo/src/FixedComponent.test.js

```
import React from "react";
import { render } from "@testing-library/react";
import FixedComponent from "./FixedComponent";

// smoke test
it("renders without crashing", function() {
  render(<FixedComponent />);
});

// snapshot test
it("matches snapshot", function() {
  const {asFragment} = render(<FixedComponent />);
  expect(asFragment()).toMatchSnapshot();
});
```

```
$ npm test FixedComponent.test.js

PASS  src/FixedComponent.test.js
✓ renders without crashing (20ms)
✓ matches snapshot (16ms)

 › 1 snapshot written.
 Snapshot Summary
 › 1 snapshot written from 1 test suite.

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   1 written, 1 total
```

Jest compares the current snapshot with the saved snapshot, and throws a test error if they are different.

Jest stores these in ***__snapshots__***.

Let's change ***FixedComponent*** rendering slightly…

```
$ npm test FixedComponent.test.js

FAIL  src/FixedComponent.test.js
✓ renders without crashing (19ms)
× matches snapshot (21ms)

 ● matches snapshot

  - Snapshot
  + Received

    <p>
  -   44
  +   62
    </p>

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 passed, 1 total
Snapshots:   1 failed, 1 total
```

If you intended change, press **u** to tell Jest to update snapshot.

Snapshot Tests with Different Props

If your component can have different UI, you may want to create multiple snapshots.

Example:

Dog.js

```
import React from "react";

function Dog(props) {
  return (
    <div className="Dog">
      <h1>{props.name}</h1>
      {props.isAdopted ? (
        <p>{props.name} has been adopted!</p>
      ) : (
        <button>Adopt me!</button>
      )}
    </div>
  );
}

export default Dog;
```

Dog.test.js

```
import React from "react";
import { render } from "@testing-library/react";
import Dog from "./Dog";

it("matches snapshot", function() {
  const { asFragment } = render(<Dog name="Whiskey" isAdopted />);
  expect(asFragment()).toMatchSnapshot();
});

it("matches snapshot", function() {
  const { asFragment } = render(<Dog name="Tubby" />);
  expect(asFragment()).toMatchSnapshot();
});
```

Takeaway

Snapshot tests are almost as easy to write as smoke tests …

… while offering a great benefit: did you change the output?

Specialized Testing

Snapshot testing can be helpful.

But often you’ll need to test what happens as people interact with your app.

How do we test that events (button clicks, form submits, etc.) produce the desired change?

Render Revisited (Again)

Here are some other methods that ***render*** returns to you:

.getByText()

Find first matching element by its text (throws error if nothing found)

.queryByText()
: Find first matching element by its text (returns null if nothing found)

.getAllByText()
: Like ***getByText*** but finds all matches

.queryAllByText()
: Like ***queryByText*** but finds all matches

.getByPlaceholderText() / queryByPlaceholderText()
: Find first matching element by placeholder text

.getAllByPlaceholderText() / queryAllByPlaceholderText()
: Like above but finds all matches

.getByTestId() / queryByTestId()
: Find matching element by a ***data-testid*** attribute (helpful if there's no other convenient way to grab an element)

.getAllByTestId() / queryAllByTestId()
: Like above but finds all matches

Full list <https://testing-library.com/docs/react-testing-library/api#render-result>

Extended Matchers

By default, projects with Create React App come with a setup file that extends the matching capability of jest.

demo/testing-demo/src/setupTests.js

```
// jest-dom adds custom jest matchers for asserting on DOM nodes.
// allows you to do things like:
// expect(element).toHaveTextContent(/react/i)
// learn more: https://github.com/testing-library/jest-dom
import '@testing-library/jest-dom/extend-expect';
```

Some helpful matchers:

.toHaveClass()
: Check whether an element has a certain class

.toBeInTheDocument()
: Check whether an element is in the document

.toContainHTML()
: Check whether the element contains a certain HTML string

.toBeEmpty()
: Check whether the element has any content

Full list <https://github.com/testing-library/jest-dom#custom-matchers>

Testing Events

React Testing Library provides a ***fireEvent*** method that you can use to mimic user interaction with your app.

.fireEvent.click(HTMLElement)
: Fire a click event

.fireEvent.submit(HTMLElement)
: Fire a submit event

.fireEvent.input(HTMLElement)
: Fire an input event

Documentation <https://testing-library.com/docs/dom-testing-library/api-events>

Specialized Testing Example

demo/testing-demo/src/Counter.js

```
import React, { useState } from "react";

function Counter() {
  const [count, setCount] = useState(0);
  const increment = () => setCount(count + 1);

  return (
    <div>
      <h1>Let's count!</h1>
      <h2>Current count: {count}</h2>
      <button onClick={increment}>Add</button>
    </div>
  );
}

export default Counter;
```

Let's test that the clicking functionality works!

demo/testing-demo/src/Counter.test.js

```
import React from "react";
import { render, fireEvent } from "@testing-library/react";
import Counter from "./Counter";

it("handles button clicks", function() {
  const { getByText } = render(<Counter />);
  const h2 = getByText("Current count: 0");

  // click on the button
  fireEvent.click(getByText("Add"));

  // is the count different?
  expect(h2).toHaveTextContent("1");
  expect(h2).not.toHaveTextContent("0");
});
```

Debugging Tests

Debugging from render

render provides a ***debug*** method that will console.log a component's DOM structure.

```
const { debug } = render(<Counter />)
debug(); // see the structure of the component

fireEvent.click(getByText("Add"));
debug(); // how has the structure changed?
```

Debugging Tests

If you want to set break points, edit your package JSON to include:

package.json

```
"scripts": {
  // ... keep other things and add this
  "test:debug": "react-scripts --inspect-brk test --runInBand"
}
```

Add ***debugger*** line in test or component you want to test

And now `npm run test:debug` will run tests where you can use Chrome debugger!

Visit ***chrome://inspect*** to debug in Chrome.

Wrap Up

- Smoke tests are incredibly simple — always make one!
- Snapshot tests are very easy if output is predictable
- Use specialized testing if you need more specific tests or to test interactions

Will I Use React Testing Library?

Maybe.

There are other testing libraries for React.

The most popular alternative is Enzyme, created by Airbnb.

Enzyme focuses more on testing React; React Testing Library focuses more on the end result in the DOM.