Callum Rimmer Follow
Full Stack Web Developer in London
Nov 3, 2015 · 4 min read

Simple React/Redux Testing

that could help you architect your applications better

You can find the accompanying source-code, for this tutorial, in the test folder at: https://github.com/caljrimmer/isomorphic-redux-app.

Testing React components is made really easy with Reacts Test Utilities. There are many article on how to test your web application but not so many on what to test or how to structure of those tests.

This article will focus on how I test my Flux Architected React web applications (built with Redux). I don't profess to document the prescribed way of testing, just what works for me.

What to test

I believe you should test the following:

- The **rendering** of a React component given a particular state
- The **behaviour** of a React component and how it affects the state via dispatched actions.
- The **state** or the application (the default values and how it changes with to subscribed actions)

The ***rendered Component*** uses ***Behaviour*** to amend ***State*** which affects the ***rendered Component.*** If you cover all this in your tests, then you are giving your web app extensive test coverage. This will give you confidence when developing and deploying.

So lets break this down:

- All components will be rendered. Thus you will need ***rendering tests*** for each component.

- Some components will have events bound to them which will affect the state (via dispatched actions). Thus you will will need a ***behavioural tests*** for all components with bound events.
- State can be accessed by multiple components so they are agnostic to React components. You need to ***test state*** against each action it subscribes too.

Test harnessing frameworks

I use expect, Reacts test utilities and JSdom for all my testing.

Code

My test are put in to a test directory with the following structure e.g.

https://github.com/caljrimmer/isomorphic-redux-app/tree/master/test

In our app we have a Sidebar component to test. We create a render/Sidebar.spec.js , behaviour/Sidebar.spec.js and state/layout.spec.js.

Render Test *(render/Sidebar.spec.js)*

```
// <app_dir>/test/render/Sidebar.spec.js
```

```
import React from 'react';
import addons from 'react/addons';
import expect from 'expect';

import Sidebar from
'../../src/common/components/layout/Sidebar';

const TestUtils = React.addons.TestUtils;

describe('Sidebar component', function(){
```

```
  //Mock State
  const user =  {
        name : 'John Smith',
        dept : 'Web Team',
        lastLogin : new Date(),
        email : 'john@smith.com',
        id : 'abcde1234'
  };
```

```
  before('render and locate element', function() {

    const renderedComponent = TestUtils.renderIntoDocument(
        <Sidebar user={user} />
    );

    const sidebar =
TestUtils.findRenderedDOMComponentWithClass(
      renderedComponent,
      'sidebar'
    );

    const username =
TestUtils.findRenderedDOMComponentWithClass(
      renderedComponent,
      'user-name'
    );

    this.sidebar = sidebar.getDOMNode();
    this.username = username.getDOMNode();

  });

  it('Sidebar should exist', function() {
    expect(this.sidebar).toExist();
  });

  it('user name should be "' + user.name+ '"', function() {
    expect(this.username.textContent).toBe(user.name);
  });


});
```

We are mocking a state and the testing what the component renders.

Behaviour Test *(behaviour/Sidebar.spec.js)*

```
// <app_dir>/test/behaviour/Sidebar.spec.js

import React from 'react';
import addons from 'react/addons';
import expect from 'expect';

import App from '../../src/common/containers/App';
import { Provider } from 'react-redux';
import configureStore from
'../../src/common/store/configureStore';
```

```
const TestUtils = React.addons.TestUtils;

describe('App component', function(){

  before('render and locate element', function() {
    const store = configureStore({});

    const renderedComponent = TestUtils.renderIntoDocument(
      <Provider store={store}>
        {()=>
           <App />
        }
      </Provider>
    );

    const wrapper =
TestUtils.findRenderedDOMComponentWithClass(
      renderedComponent,
      'wrapper'
    );

    const sidebar =
TestUtils.findRenderedDOMComponentWithClass(
      renderedComponent,
      'sidebar'
    );

    const sidebarToggle =
TestUtils.findRenderedDOMComponentWithClass(
      renderedComponent,
      'sidebar-toggle'
    );

    this.wrapper = wrapper.getDOMNode();
    this.sidebar = sidebar.getDOMNode();
    this.sidebarToggle = sidebarToggle.getDOMNode();

  });

  it('sidebar should exist', function() {
    expect(this.sidebar).toExist();
  });

  it('sidebar should be closed', function() {

expect(this.sidebar.getAttribute('class')).toBe('sidebar');
  });

  it('sidebar toggle should exist', function() {
    expect(this.sidebarToggle).toExist();
  });

  it('clicking sidebar toggle should open sidebar',
function() {

expect(this.wrapper.getAttribute('class')).toBe('wrapper');
    TestUtils.Simulate.click(this.sidebarToggle);
    expect(this.wrapper.getAttribute('class')).toBe('wrapper
open');
  });
```

```
});
```

We are testing the behaviour of the component by simulating a click and seeing how it effects the Sidebar DOM. We are not testing the state directly, just looking at the outcome of our interaction.

State Test *(state/layout.spec.js)*

```
// <app_dir>/test/state/layout.spec.js

import React from 'react';
import addons from 'react/addons';
import expect from 'expect';

import * as LayoutActions from
'../../src/common/actions/layout';
import configureStore from
'../../src/common/store/configureStore';

describe('Layout State', function(){

  before('render and locate element', function() {
    const store = configureStore({});
    this.store = store;
  });

  it('layout state should exist', function() {
    expect(this.store.getState().layout).toExist();
  });

  it('layout state to instansiate with...', function() {
    expect(this.store.getState().layout.present).toEqual({
      sidebarOpen : false
    });
  });

  it('dispatch TOGGLE_SIDEBAR, state should change layout
state', function() {
    expect(this.store.getState().layout.present).toEqual({
      sidebarOpen : false
    });
    this.store.dispatch(LayoutActions.toggleSidebar(true));
    expect(this.store.getState().layout.present).toEqual({
      sidebarOpen : true
    });
  });

});
```

Layout state is used by the sidebar. The layout actions include toggleSidebar() which is dispatched by the click event in the Sidebar component.

The difference with state tests, is that they are independent of the React Components. We are just testing state and actions.

How can this help me build my app?

Breaking up your tests mean you can concentrate on building your understanding of your application before creating React Components.

You can create your different application states and define your actions first. You can run and test them completely before committing to building your components. ***You build your state tests first.***

Once you know your states, ***then you can mock out your rendering tests***. You will know all the possible states of your component and can render test against them. i.e. Sidebar can be open or closed. We render test for an 'open' class when {sidebarOpen : true} is in the state.

You states and actions will now already be tested. You can start binding events which dispatch actions to your new render tested components. ***The final part of your tests will be the behaviour of your component.*** You are testing the resultant DOM after a component event has been triggered.

Conclusion

Testing applications can sometime be a case of ad-hoc tests which give the developer and the product owner confidence. The structure of testing and the nomenclature of the test files can become confused over time.

I like this approach because I can look at the React components, Actions and Stores of my app and see exactly what my test files are meant to achieve, and where I have gaps in my test coverage.

I find I build apps quicker and that they are easier to maintain and improve.