React Unit Using Enzyme

https://scotch.io/tutorials/testing-react-components-with-enzyme-and-jest

Why Jest?

Jest is a fast JavaScript testing utility by Facebook that enables you to get started with testing your JavaScript code with zero configuration.

Setting Up Our React App

```
create-react-app enzyme-tests
cd enzyme-tests
yarn start
```

Setting Up Enzyme

To get started with Enzyme, go ahead and install the library via yarn or npm as a dev dependency.

```
yarn add --dev enzyme enzyme-adapter-react-16
```

src/enzyme.js

```
import Enzyme, { configure, shallow, mount, render } from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';

configure({ adapter: new Adapter() });
export { shallow, mount, render };
export default Enzyme;
```

Finally, create a `components` and `components/tests` folder inside `src` where our components and tests will live in respectively.

Shallow Rendering

Shallow rendering is the most basic version of testing with Enzyme. As the name suggests, shallow rendering limits it's scope to the component to be tested and not it's children.

This comes in handy in various scenarios:

1. For presentational/dummy components that simply render props, there is no need to try and render any other children.
2. For components with deeply nested children components, a change in behavior of the children should not affect the behavior of the parent component to be tested.

For this section, we will demonstrate testing a presentational component with shallow render.

Take a look at the `List` component below that expects an `items` prop and displays them in an unordered list.

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

/_*
 _ Render a list of items
 _
 _ @param {Object} props - List of items
 */
function List(props) {
  const { items } = props;
  if (!items.length) {
    // No Items on the list, render an empty message
    return <span className="empty-message">No items in list</span>;
  }

  return (
    <ul className="list-items">
      {items.map(item => <li key={item} className="item">{item}</li>)}
    </ul>
  );
}

List.propTypes = {
  items: PropTypes.array,
};

List.defaultProps = {
  items: [],
};

export default List;
```

Let's add a few tests for the component.

/src/components/tests/List.test.js

```
import React from 'react';
import { shallow } from '../enzyme';

import List from './List';

describe('List tests', () => {

  it('renders list-items', () => {
    const items = ['one', 'two', 'three'];
    const wrapper = shallow(<List items={items} />);

    // Expect the wrapper object to be defined
    expect(wrapper.find('.list-items')).toBeDefined();
    expect(wrapper.find('.item')).toHaveLength(items.length);
  });

  it('renders a list item', () => {
    const items = ['Thor', 'Loki'];
    const wrapper = shallow(<List items={items} />);

    // Check if an element in the Component exists
    expect(wrapper.contains(<li key='Thor' className="item">Thor</li
>)).toBeTruthy();
  });

  it('renders correct text in item', () => {
    const items = ['John', 'James', 'Luke'];
    const wrapper = shallow(<List items={items} />);

    //Expect the child of the first item to be an array
    expect(wrapper.find('.item').get(0).props.children).toEqual('John');
  });
});
```

The test suite imports a `shallow` enzyme method from the configuration we created in the previous section, wraps the `List` component and returns an instance of the rendered component.

We then make a series of assertions against the instance to check if the component renders the content correctly. While the tests are not optimal, we take advantage of a few methods exposed by the shallow API.

Full DOM rendering

in the last section, we were able to shallow render the `List` component and write tests to assert the actual text in the `li` tags. Sweet, right? In this section, we will look at full DOM rendering by making a few modifications to the component by breaking out the `li` element into a component of it's own called `ListItem`.

src/compoents/ListItem.js

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

/_*
 _ Render a single item
 _
 _ @param {Object} props
 */
function ListItem(props) {
  const { item } = props;
  return <li className="item">{item}</li>;
}

ListItem.propTypes = {
  item: PropTypes.string,
};

export default ListItem;
```

With the new component, replace the `li` tag with our shiny new component.

src/components/List.js

```
...
import ListItem from './ListItem';
...

return (
    <ul className="list-items">
      {items.map(item => <ListItem key={item} item={item} />)}
    </ul>
  );
```

Let's now run the tests we wrote in the previous section and see what happens. If you did this right, your tests should be failing as terribly as mine are.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL                                   1: node

  ● List tests › renders list-items

    expect(received).toHaveLength(length)

    Expected value to have length:
      3
    Received:
      {"length": 0, Symbol(enzyme.__root__): {"length": 1, Symbol(enzyme.__root__): [Circular], Symbol(enzyme.__unrendered__): <List items={["one", "two", "three"]} />, Symbol(enzyme.__renderer__): {"batchedUpdates": [Function batchedUpdates], "getNode": [Function getNode], "render": [Function render], "simulateEvent": [Function simulateEvent], "unmount": [Function unmount]}, Symbol(enzyme.__node__): {"instance": null, "key": undefined, "nodeType": "host", "props": {"children": [<ListItem item="one" />, <ListItem item="two" />, <ListItem item="three" />], "className": "list-items"}, "ref": null, "rendered": [{"instance": null, "key": "one", "nodeType": "function", "props": {"item": "one"}, "ref": null, "rendered": null, "type": [Function ListItem]}, {"instance": null, "key": "two", "nodeType": "function", "props": {"item": "two"}, "ref": null, "rendered": null, "type": [Function ListItem]}, {"instance": null, "key": "three", "nodeType": "function", "props": {"item": "three"}, "ref": null, "rendered": null, "type": [Function ListItem]}], "type": "ul"}, Symbol(enzyme.__nodes__): [{"instance": null, "key": undefined, "nodeType": "host", "props": {"children": [<ListItem item="one" />, <ListItem item="two" />, <ListItem item="three" />], "className": "list-items"}, "ref": null, "rendered": [{"instance": null, "key": "one", "nodeType": "function", "props": {"item": "one"}, "ref": null, "rendered": null, "type": [Function ListItem]}, {"instance": null, "key": "two", "nodeType": "function", "props": {"item": "two"}, "ref": null, "rendered": null, "type": [Function ListItem]}, {"instance": null, "key": "three", "nodeType": "function", "props": {"item": "three"}, "ref": null, "rendered": null, "type": [Function ListItem]}], "type": "ul"}]}, Symbol(enzyme.__options__): {"adapter": {"options": {"enableComponentDidUpdateOnSetState": true}}}}, Symbol(enzyme.__unrendered__): null, Symbol(enzyme.__renderer__): {"batchedUpdates": [Function batchedUpdates], "getNode": [Function getNode], "render": [Function render], "simulateEvent": [Function simulateEvent], "unmount": [Function unmount]}, Symbol(enzyme.__node__): undefined, Symbol(enzyme.__nodes__): Array [], Symbol(enzyme.__options__): {"adapter": {"options": {"enableComponentDidUpdateOnSetState": true}}}}
    received.length:
      0

      at Object.it (src/components/List.test.js:14:35)
          at new Promise (<anonymous>)
      at Promise.resolve.then.el (node_modules/p-map/index.js:46:16)
          at <anonymous>

  ● List tests › renders a list item

    expect(received).toBeTruthy()

    Expected value to be truthy, instead received
      false

      at Object.it (src/components/List.test.js:22:184)
          at new Promise (<anonymous>)
      at Promise.resolve.then.el (node_modules/p-map/index.js:46:16)
          at <anonymous>

  ● List tests › renders correct text in item

    TypeError: Cannot read property 'props' of null

Ln 3, Col 1 (34 selected)   Spaces: 2   UTF-8   LF   JavaScript   ESLint
```

Why would this happen? I mean the UI did not change at all. All we did was move things a littlte. Let's debug this further. The enzyme wrapper exposes a `debug` method that allows use to peek into the wrapped instance of our component and see what went wrong.

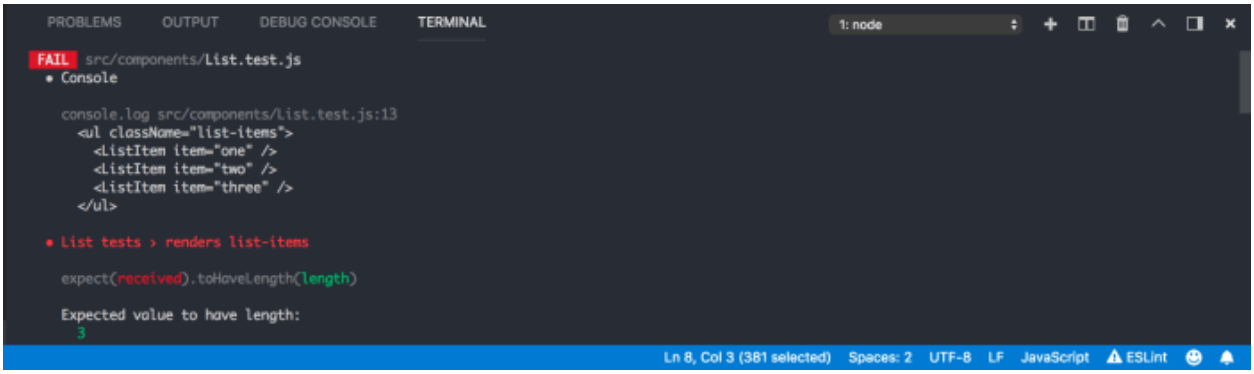
Let's add a log in our tests to see what went wrong.

/src/components/tests/List.test.js

```
...

it('renders list-items', () => {
    const items = ['one', 'two', 'three'];
    const wrapper = shallow(<List items={items} />);
```

```
    // Let's check what wrong in our instance
    console.log(wrapper.debug());

    // Expect the wrapper object to be defined
    expect(wrapper.find('.list-items')).toBeDefined();
    expect(wrapper.find('.item')).toHaveLength(items.length);
  });

...
```

Run the tests again and look through the terminal output, you should see our component instance log.



As you can see, the wrapper method does not render the `ListItem` Children as we would have expected. Therefore, our tests that checked for a class or `li` element failed.

It may not seem necessary to shallow render such a simple component where the child is a presentational component but it comes in handy when you are writting tests for components that are wrapped by libraries such as react-redux's `connect` or `reduxForm`.

The idea here is that we do not want to test the inner workings of such high order components, therefore, no need to concern ourselves with their rendering.

Okay enough chatter. Let's fix the failling tests. We could stop checking for the `li` elements in our tests and check for the `ListItem` tag as shown below

/src/components/tests/List.test.js

```
...

it('renders list-items', () => {
    const items = ['one', 'two', 'three'];
    const wrapper = shallow(<List items={items} />);
```

```
    // Expect the wrapper object to be defined
    expect(wrapper.find('ListItem')).toBeDefined();
    expect(wrapper.find('ListItem')).toHaveLength(items.length);
  });

...
```

n this case, we actually want to test the entire tree of children in the List component. so instead, we will replace the `shallow` component with `mount`. Mount enables us to perform a full render. Here is a snippet of the updated code and a log of the debug instance.

/src/components/tests/List.test.js

```
import React from 'react';
import { mount } from '../enzyme';

import List from './List';

describe('List tests', () => {

  it('renders list-items', () => {
    const items = ['one', 'two', 'three'];

    // Replace shallow iwth mount
    const wrapper = mount(<List items={items} />);

    // Let's check what wrong in our instance
    console.log(wrapper.debug());

    // Expect the wrapper object to be defined
    expect(wrapper.find('.list-items')).toBeDefined();
    expect(wrapper.find('.item')).toHaveLength(items.length);
  });

  ...
});
```

```
PROBLEMS 2    OUTPUT    DEBUG CONSOLE    TERMINAL                    1: node

PASS  src/components/List.test.js
  • Console

    console.log src/components/List.test.js:13
      <List items={{...}}>
        <ul className="list-items">
          <ListItem item="one">
            <li className="item">
              one
            </li>
          </ListItem>
          <ListItem item="two">
            <li className="item">
              two
            </li>
          </ListItem>
          <ListItem item="three">
            <li className="item">
              three
            </li>
          </ListItem>
        </ul>
      </List>

PASS  src/App.test.js

Test Suites: 2 passed, 2 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        0.565s, estimated 1s
Ran all test suites.

Watch Usage: Press w to show more.
```

As you can see, `mount`'s full rendering API renders the entire DOM, including that of the children. And our tests are fixed!