React Testing

Overview

- Unit testing
- Snapshot testing
- Interactive testing

Unit testing

- Testing mostly requires a DOM (jsdom is a virtual DOM that ships with `create-react-app`)
- Jest is an assertion library library bundled with `create-react-app`
- ReactUtils helps us render, interact with and fetch DOM elements
- Enzyme is a library from AirBnB that provides a more jQuery-like API for the same functionality

First unit test

```
import React from 'react'
import ReactDOM from 'react-dom'
import App from './App'

describe('<App />', () => {
  it('renders without crashing', () => {
    const div = document.createElement('div')
    ReactDOM.render(<App />, div)
    ReactDOM.unmountComponentAtNode(div)
  })
})
```

First unit test continued...

- `describe` and `it` similar to rspec in Ruby
- Create a root element to mount a component to
- Render component
- Cleanup component

Problems with this approach

- Rendering a component renders all of its children
- Best practice is to test components in isolation
- We want to render the parent component and none of the children

So how do we do this?

Setup enzyme to project

→ Create a file in the `src` folder `setupTests.js`

```
import { configure } from 'enzyme'
import Adapter from 'enzyme-adapter-react-16'

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

First unit test revised

```
...
import { shallow } from 'enzyme'

describe('<App />', () => {
  it('renders without crashing', () => {
    const app = shallow(<App />)
  })

  it('displays the title', () => {
    const app = shallow(<App />)
    expect(app.find('h1').text()).toEqual('React API Tester')
  })
})
```

Snapshot testing

- Every react element has a data representation in the virtual DOM
- Snapshot testing allows us to record this representation and assert on it
- Effective way of making multiple assertions against a component

```
import React from 'react'
import renderer from 'react-test-renderer'
import Form from './Form'

describe('<Form />' ,() => {
  it('default state matches snapshot', () => {
    const form = renderer.create(<Form />).toJSON()

    expect(form).toMatchSnapshot()
  });
});
```

Interactive testing

Adding interactions - 1

```
it('submits the form with valid values when button clicked', () => {
  const submitFormData = jest.fn()
  const fakeAPIUrl = 'https://fake.com/api/'
  const fakeJSONBody = '{"foo": "bar"}'
  const fakeFormMethod = 'POST'

  const root = TestUtils.renderIntoDocument(
    <Form submitFormData={submitFormData} />
  )
...
```

Adding interactions - 2

```
const form = TestUtils.findRenderedDOMComponentWithTag(root, 'form')

const methodSelect = ReactDOM.findDOMNode(root).querySelector('select')

TestUtils.Simulate.change(methodSelect, { target: { value: fakeFormMethod } })

const urlInput = ReactDOM.findDOMNode(root).querySelector(
  'input[name="url"]'
)

const bodyInput = ReactDOM.findDOMNode(root).querySelector('textarea')
```

...

Adding interactions - 3

```
  TestUtils.Simulate.change(urlInput, { target: { value: fakeAPIUrl } })
  TestUtils.Simulate.change(bodyInput, { target: { value: fakeJSONBody } })
  TestUtils.Simulate.submit(form)

  expect(submitFormData).toHaveBeenCalledWith({
    body: fakeJSONBody,
    method: formMethod,
    url: fakeAPIUrl
  })
})
```