Introduction to Enzyme

It is simpler to test the output of your React Components with Enzyme, a JavaScript Testing tool for React. Given the output, you may also alter, navigate, and in some ways imitate runtime.

By imitating jQuery's API for DOM manipulation and traversal, Enzyme's API aims to be simple and versatile.

Enzyme should work with all of the popular test runners and assertion libraries because it has no preference as to which of them you choose. Although mocha and chai are used in the documentation and examples for enzyme, you should be able to extrapolate to your own framework.

For testing your React components using enzyme with personalized assertions and handy utilities, you might think about using:

- chai-enzyme with Mocha/Chai.
- jasmine-enzyme with Jasmine.
- jest-enzyme with Jest.
- should-enzyme for should.js.
- expect-enzyme for expect.

Integration of Enzyme with JEST

Enzyme supports react hooks with some limitations in .shallow() due to upstream issues in React's shallow renderer:

- useEffect() and useLayoutEffect() don't get called in the React shallow renderer
- useCallback() doesn't memoize callback in React shallow renderer

Enzyme’s Page Rendering APIs

Enzyme support multiple ways of rendering a page depending upon the user’s available resources such as memory and network bandwidth. Those include:

- Shallow Rendering API
- Full DOM Rendering API
- Static Rendering API

Shallow rendering

Shallow rendering is useful to constrain yourself to testing a component as a unit, and to ensure that your tests aren't indirectly asserting on behavior of child components.

As of Enzyme v3, the shallow API does call React lifecycle methods such as componentDidMount and componentDidUpdate.

A few methods available in shallow rendering API include:

- .find(selector) => ShallowWrapper
- .findWhere(predicate) => ShallowWrapper
- .filter(selector) => ShallowWrapper
- .filterWhere(predicate) => ShallowWrapper
- .hostNodes() => ShallowWrapper
- .contains(nodeOrNodes) => Boolean

Full Rendering API

When testing components that are encased in higher order components or when you have components that may interact with DOM APIs, full DOM rendering is the best option.

A full DOM API must be accessible at the global level in order for full DOM rendering to be possible. It must therefore be used in a setting that at the very least "looks like" a browser environment. The suggested method for utilising mount if you don't want to run your tests inside of a browser is to rely on a library called jsdom, which is essentially a headless browser developed entirely in JS.

A few methods available in Full DOM rendering API include:

- .find(selector) => ReactWrapper
- .findWhere(predicate) => ReactWrapper
- .filter(selector) => ReactWrapper
- .filterWhere(predicate) => ReactWrapper
- .hostNodes() => ReactWrapper
- .contains(nodeOrNodes) => Boolean

Static Rendering API

Create HTML from your React tree using enzyme's render function, then examine the HTML's structure.

Render employs a third-party HTML parsing and traversal library called Cheerio, but it nevertheless outputs a wrapper that is quite similar to the other renderers in enzyme, mount, and shallow. It would be a disservice to Cheerio's users if we attempted to duplicate its excellent HTML parsing and traversal capabilities.

Cheerio's function Object() { [native code] } will be referred to as CheerioWrapper for the duration of this description because it is comparable to our ReactWrapper and ShallowWrapper constructors. For information on the methods offered by a CheerioWrapper instance, consult the Cheerio API documentation.