Enzyme Testing Cheatsheet

A comprehensive cheat sheet for testing React components with Enzyme, covering essential methods, selectors, and best practices for effective testing.

Enzyme Basics & Setup

Installation

Install Enzyme and an adapter for your React version. For React 16, use `enzyme-adapter-react-16`:

```
npm install --save-dev enzyme enzyme-adapter-react-16
```

For React 17, use `enzyme-adapter-react-17`:

```
npm install --save-dev enzyme enzyme-adapter-react-17
```

Configuration

Configure Enzyme in your test setup file (e.g., `src/setupTests.js` or `test/setup.js`):

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

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

Replace `enzyme-adapter-react-16` with the appropriate adapter for your React version.

Mounting Components

`shallow(node[, options])`	Shallow renders only the top-level component, not its children. Useful for isolating unit tests.
	`import { shallow } from 'enzyme';`
	`import MyComponent from './MyComponent';`
	`const wrapper = shallow(<MyComponent />);`

`mount(node[, options])`	Full DOM rendering, including child components. Requires a DOM environment (e.g., jsdom).
	`import { mount } from 'enzyme';`
	`import MyComponent from './MyComponent';`
	`const wrapper = mount(<MyComponent />);`

`render(node[, options])`	Renders to static HTML. Useful for testing the rendered output. Returns a Cheerio wrapper.
	`import { render } from 'enzyme';`
	`import MyComponent from './MyComponent';`
	`const wrapper = render(<MyComponent />);`

Selectors & Traversal

Finding Nodes

`wrapper.find(selector)`	Finds all elements matching the CSS selector, component, or React element.
	`wrapper.find('button'); // Finds all button elements`
	`wrapper.find(MyComponent); // Finds all instances of MyComponent`

`wrapper.findWhere(predicate)`	Finds elements based on a custom predicate function. `wrapper.findWhere(n => n.prop('name') === 'test');`
`wrapper.filter(selector)`	Filters the current selection of nodes based on the selector. `wrapper.find('div').filter('.my-class');`

Traversal

`wrapper.children()`	Returns the direct children of the current node(s). `wrapper.find('ul').children(); // Returns <li> elements if <ul> contains <li> elements`
`wrapper.parent()`	Returns the parent of the current node(s). `wrapper.find('li').parent(); // Returns the <ul> element`
`wrapper.closest(selector)`	Returns the first ancestor matching the selector. `wrapper.find('span').closest('div');`

Simulating Events & Assertions

Simulating Events

`wrapper.simulate(event[, mock])`	Simulates an event on the selected node(s).
	`wrapper.find('button').simulate('click');`
	`wrapper.find('input').simulate('change', { target: { value: 'new value' } });`

Common Events	`click`, `change`, `submit`, `keyDown`, `keyUp`, `mouseEnter`, `mouseLeave`

Assertions

`wrapper.exists(selector)`	Checks if elements matching the selector exist. `expect(wrapper.exists('button')).toBe(true);`
`wrapper.text()`	Returns the text content of the node(s). `expect(wrapper.find('p').text()).toEqual('Hello World');`
`wrapper.html()`	Returns the HTML content of the node(s). `expect(wrapper.find('div').html()).toEqual('<div class="my-class">...</div>');`
`wrapper.props()`	Returns all props of the root node. `expect(wrapper.props().className).toEqual('my-class');`
`wrapper.prop(key)`	Returns a specific prop of the root node. `expect(wrapper.prop('id')).toEqual('uniqueId');`

State & Props

`wrapper.state([key])`	Gets the state of a stateful component. If `key` is provided, returns the value of that key. Otherwise, returns the entire state object. (Only for `mount` and `shallow` with stateful components)
	`wrapper.setState({ count: 1 });`
	`expect(wrapper.state('count')).toEqual(1);`

`wrapper.setProps(props)`	Sets the props of the component. Triggers a re-render. (Only for `mount` and `shallow`)
	`wrapper.setProps({ name: 'New Name' });`
	`expect(wrapper.prop('name')).toEqual('New Name');`

`wrapper.update()`	Forces a re-render of the component. Useful when props or state have been updated indirectly. `wrapper.update();`

Advanced Techniques

Testing Higher-Order Components (HOCs)

When testing HOCs, test the *unwrapped* component directly to isolate its logic.

```
import { MyComponent } from './MyComponent';
// Assuming MyComponent is the unwrapped component

describe('MyComponent', () => {
  it('should render correctly', () => {
    const wrapper = shallow(<MyComponent prop="value" />);
    expect(wrapper).toBeDefined();
  });
});
```

Testing Connected Components (Redux)

When testing connected components, test the *unconnected* component directly and mock the Redux store.

```
import { MyComponent } from './MyComponent';
// Assuming MyComponent is the unconnected component

describe('MyComponent', () => {
  it('should render correctly', () => {
    const wrapper = shallow(<MyComponent prop="value" />);
    expect(wrapper).toBeDefined();
  });
});
```

Mocking Modules

Use Jest's mocking capabilities to isolate components and control dependencies.

```
jest.mock('./MyModule', () => ({
  myFunction: jest.fn(() => 'mocked value'),
}));

import { myFunction } from './MyModule';

expect(myFunction()).toBe('mocked value');
```