**How to Set up React Test with Mocha and Enzyme for TypeScript**

[Enzyme](https://github.com/airbnb/enzyme) is created by Airbnb for React component testing. You can easily test components’ output and it works with different testing frameworks like [Mocha](https://mochajs.org/) or [Jest](https://jestjs.io/). This post focuses on setting up the test environment with Mocha for [TypeScript](https://www.typescriptlang.org/).

If you are used to JavaScript and fairly new to TypeScript, setting up unit test can be cumbersome. Basically, TypeScript complains about almost everything that you used to do with JavaScript. However, once you get your head around it, it becomes easier. When I set this up first time, it took me a long time to figure out why TypeScript is complaining about things like window does not exist on type Global or mount method not working because a global document is not loaded. But, getting this work helped me to understand TypeScript a little bit better.

The complete example can be found on [this repo](https://github.com/mydatahack/book-reading-tracker-typescript-react-redux).

Let’s get started.

1. **Install dependencies**

With TypeScript, you need to install a regular module and @types module. Most of the common modules has @types available these days. So, you can simply install the regular module as well as the one prefixed with @types. Some modules are written in TypeScript and you don’t need to install @types. For example, a test coverage tool, [istanbul](https://istanbul.js.org/) is written in TypeScript and does not require any type module.

Using chai for assertion library. ts-node is also required to run TypeScript in the node environment. Later, you need to register ts-node in the mocha command.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

npm i chai mocha ts-node --save-dev

npm i @types/chai @types/mocha @types/sinon --save-dev

[/cc]

Now, let’s install dependencies for Enzyme. Mocha doesn’t run tests in a Node environment, not in a browser environment. Therefore, enzyme’s mount API requires a DOM. To simulate a browser environment, we are using JSDOM.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

npm i enzyme jsdom jsdom-global enzyme-adapter-react-16

npm i @types/enzyme @types/jsdom @types/enzyme-adapter-react-16 --save-dev

[/cc]

1. **Creating test setup file**

Node environment where mocha runs tests does not have browser properties. We first need to add types for required browser environment properties by extending interface and adding properties.

Newly created globalNode object has window, document and navigator properties. All we need to do is assign JSDOM window object to the window. This will create window globally.

Here is the entire test set up code. [Configuring enzyme adapter](https://airbnb.io/enzyme/docs/installation/react-16.html) is the same as regular JavaScript.

<pre class=" toolbar:2 lang:js font-size:12px decode:true" >

</pre>

1. **Getting the test script right**

In package.json file, you can use the test script as below. Component test uses jsx file because you need to use React. Make sure to include both .ts and .tsx files.

[cc lang="bash" tab\_size="4" lines="-1" theme="mac-classic" line\_numbers="false"]

mocha -r ts-node/register -r jsdom-global/register -r unitTestSetup.ts \*\*/test/\*\*/\*.ts test/\*\*/\*.tsx --recursive --timeout 5000

[/cc]

1. **Simple test example**

Here is the sample component test example. It should work by typing npm test

<pre class=" toolbar:2 lang:js font-size:12px decode:true" >

import \* as React from 'react';

import { mount } from 'enzyme';

import { expect } from 'chai';

import FormSectionHeader from '../src/components/FormSectionHeader';

describe('<Form /> component test', () => {

it('should have an input field for bookTitle', () => {

const component = mount(<FormSectionHeader headerTitle="Hello"/>)

expect(component.find('h4').text()).to.equal("Hello")

})

})

</pre>

**Common Errors**

Here are the list of common errors and how to handle them.

**It looks like you called mount() without a global document being loaded**

* Mocha doesn't run the test in a browser environment & enzyme's mount API requires a DOM. JSDOM is required to simulate a browser environment in a Node environment.

**'window is not defined'**

* It can be overcome by installing jsdom-global and adding -r jsdom-global/register in mocha command

**Property 'window' does not exist on type 'Global' error**

* When we create setup file for JSDOM (unitTestSetup.ts on the root folder), we need to add browser properties to Node global environment as it does not have browser properties. We can extend NodeJS.Global properties by adding interface and redefining the global variable with global Node variable.