Learning and Applying JEST for MERN stack development

First of all, I've decided to learn Jest because we are working with the MERN stack, and we were only using ARC for some manual testing of our react application, specifically our API calls. I've also decided to learn Jest because I want to better apply Test-Driven Development (TDD) into my programming habits. Also, I picked Jest after researching what testing tools people are using these days (as of May 2021) and found that it seemed to be between Jest and Mocha. I chose Jest instead of Mocha after reading this article, and mainly because we are using React and not just Node.

Table of Contents

Resources

Here are a couple of resources for learning Jest that I found:

Learning the Basics of Jest

- good for configuration
- The official documentation from jestjs.io
- this blog includes some basic use cases with both Jest and Mocha
- free video tutorials:
  - Introduction To Testing In JavaScript With Jest
  - Jest Crash Course - Unit Testing in JavaScript

Testing with Jest and react-testing-library

- https://www.smashingmagazine.com/2020/07/react-apps-testing-library/
- https://kentcdodds.com/blog/avoid-nesting-when-youre-testing
- https://testing-library.com/docs/ecosystem-user-event/#!

Testing with Jest and Material-UI

- https://dev.to/sama/testing-material-ui-form-components-1cnh

Testing accessibility

Learning about ARIA

- A brief introduction: How to Use ARIA Roles, Properties, and States in HTML

- more complete documentation but a lot more dense: Aria in HTML

Testing with Jest and why it relates

- Development in testing-library
- Why it'd bad to use getByTestId

Testing with Jest Snapshots

- my favorite introduction
- having more than one snapshot per test
- official documentation from Jest
- https://medium.com/iqoqo-engineering/this-library-uses-your-jest-tests-to-generate-mocks-c07c322c58e3
- https://medium.com/styled-components/effective-testing-for-styled-components-67982eb7d42b
- https://www.digitalocean.com/community/tutorials/how-to-write-snapshot-tests-for-react-components-with-jest

important notes:

- don't update snapshots to record buggy behavior - only update when we changed the UI intentionally!
- to update snapshots, you can do it like this:

  npm test -- --updateSnapshot --testNamePattern=[part of name]

i.e. for a test that looks like

```
it("renders correctly", () => {
  ...
})
```

the snapshot update command could look like

```
npm test -- --updateSnapshot --testNamePattern=correctly
```

Basic Jest Tests

Here is a link to my codeSandbox that contains a bunch of basic Jest tests, does not use create-react-app, and uses ESM. Snapshots do not work well in codeSandbox.

Using ECMAScript modules

ESM must be manually enabled. create-react-app already does this for you, but if you are adding react or jest to your already existing project, you need to take a couple steps.

For Jest, you could refactor the code to use CommonJS, which will make the code work without changing any configurations for Jest, but, at least from what I've gathered, CommonJS is slowly becoming obsolete from a modern JavaScript perspective.

Here are three ways to make tests that check code using ESM work:

1. Refactor the code above to use CommonJs, seen in this tutorial.
2. Make some alterations to your package.json using "type":"module"
3. Add Babel to you code (recommended if you are also using React)

Type:Module (only recommended if you are not using React)

There is no overhead of installing Babel if you are not using React.

Here are what you need to add to your package.json file for Jest to work:

```
"type": "module",
"scripts": {
    "test": "node --experimental-vm-modules node_modules/jest/bin/jest.js --
coverage",
    "testWatch": "node --experimental-vm-modules node_modules/jest/bin/jest.js --
watchAll"
},
```

Take notice of adding "type":"module" and adding the experimental tag to your scripts. Also, you don't need the coverage or watchAll tags, but I recommend them for their utility.

Babel (Recommended if you are also using React)

Say you want to run a file like this, which uses react-testing-library:

```
/**
* @jest-environment jsdom
*/

import React from 'react'
import { render } from '@testing-library/react'
import '@testing-library/jest-dom'

const Hello = () => <h1>Hello World</h1>

test('first hello test', () => {
const { container } = render(<Hello />)

expect(container).toHaveTextContent('Hello World')
})
```

If you are using React, it seems you still need to have Babel installed for running your Jest tests. There seems to be some unofficial ways to get around using Babel but it does not seem stable yet so I recommend "biting the bullet", so to speak, and adding it to your devDependencies in Node for testing purposes. Also, if you install Babel, you don't need to use "type":"module" which also means you don't need to have the experimental tag for running Jest, which is a bonus.

Your scripts could be:

```
"scripts": {
    "test": "jest --coverage",
    "testWatch": "jest --watchAll"
},
```

Or, if you don't want to include the comment at the top about the jest environment, your test command could look like this (extra env tag):

```
"test": "jest --env=jsdom",
```

Unfortunately, with Babel, you do have the overhead of installing these packages through Node to your devDependencies:

```
"devDependencies": {
    "@babel/preset-env": "7.14.5",
    "@babel/preset-react": "7.14.5",
    ...
}
```

*For the testing library, you additionally need these devDependencies:

```
"@testing-library/jest-dom": "5.14.0",
"@testing-library/react": "11.2.7",
```

And having to add this file to make React work in your tests: **.babelrc.cjs** (it must be .cjs for it to work):

```
module.exports = {
presets: [
  [
    '@babel/preset-env',
    {
      targets: {
        node: 'current'
      }
    }
  ],
  ['@babel/preset-react', {targets: {node: 'current'}}]
]
```

}

According to this article, instead of .babelrc.cjs, they used **babel.config.js** and put this in it:

```
// babel.config.js
module.exports = {
    presets: ["@babel/preset-env", "@babel/preset-react"]
};
```

But that only worked in my codeSandbox and not in my local project. For my local project to work, I needed to use the .babelrc.cjs file above.

While using Babel seems like the longer route, it's better long term since it gives you room to add React to your project, which is likely why you were using Jest in the first place.

Documentation for Configuration

- ECMAScript Modules
- Configuring Jest

Running your tests

Once everything is configured, to run the tests once (with coverage), you would type from your terminal:

```
npm test
```

To run your tests automatically whenever you make a change and save, you would run:

```
npm run testWatch
```