Keeping Mocks Externalized

In this lesson, we'll be separating all mocks in their own Javascript file.

WE'LL COVER THE FOLLOWING

Mocks Separation

Jest allows us to have all our mocks separated in their own JavaScript file, placing them under a `__mocks__` folder, keeping the tests as clean as possible.

So we can take the `jest.mock` block from the top of the `Form.test.js` file out to its own file:

```
require('./check-versions')()

process.env.NODE_ENV = 'production'

var ora = require('ora')
var rm = require('rimraf')
var path = require('path')
var chalk = require('chalk')
var webpack = require('webpack')
var config = require('../config')
var webpackConfig = require('./webpack.prod.conf')

var spinner = ora('building for production...')
spinner.start()

rm(path.join(config.build.assetsRoot, config.build.assetsSubDirectory), err => {
  if (err) throw err
  webpack(webpackConfig, function (err, stats) {
    spinner.stop()
    if (err) throw err
    process.stdout.write(stats.toString({
      colors: true,
      modules: false,
      children: false,
      chunks: false,
      chunkModules: false
    }) + '\n\n')
```

```
    console.log(chalk.cyan('  Build complete.\n'))
    console.log(chalk.yellow(
      '  Tip: built files are meant to be served over an HTTP server.\n' +
      '  Opening index.html over file:// won\'t work.\n'
    ))
  })
})
```

Just like that, with no effort, Jest automatically applies the mock in all our tests so we don’t have to do anything extra or mock it in every test manually. Notice the module name must match the file name. If you run the tests again, they should still pass.

Keep in mind the modules registry and the mocks state are kept as it is, so if you write another test afterward, you may get undesired results:

```
require('./check-versions')()

process.env.NODE_ENV = 'production'

var ora = require('ora')
var rm = require('rimraf')
var path = require('path')
var chalk = require('chalk')
var webpack = require('webpack')
var config = require('../config')
var webpackConfig = require('./webpack.prod.conf')

var spinner = ora('building for production...')
spinner.start()

rm(path.join(config.build.assetsRoot, config.build.assetsSubDirectory), err => {
  if (err) throw err
  webpack(webpackConfig, function (err, stats) {
    spinner.stop()
    if (err) throw err
    process.stdout.write(stats.toString({
      colors: true,
      modules: false,
      children: false,
      chunks: false,
      chunkModules: false
    }) + '\n\n')

    console.log(chalk.cyan('  Build complete.\n'))
    console.log(chalk.yellow(
      '  Tip: built files are meant to be served over an HTTP server.\n' +
      '  Opening index.html over file:// won\'t work.\n'
    ))
  })
})
```

The second test like below should fail:

```
// Second Test
  it("Axios should not be called here", () => {
  expect(axios.get).toBeCalledWith(
    "https://jsonplaceholder.typicode.com/posts?q=an"
  );
  });
```

But it will not! That's because `axios.get` was called on the test before.

For that reason, it's good practice to clean the module registry and the mocks, since they're manipulated by Jest in order to make mocking happen. For that, we have added this in your `beforeEach`:

```
require('./check-versions')()

process.env.NODE_ENV = 'production'

var ora = require('ora')
var rm = require('rimraf')
var path = require('path')
var chalk = require('chalk')
var webpack = require('webpack')
var config = require('../config')
var webpackConfig = require('./webpack.prod.conf')

var spinner = ora('building for production...')
spinner.start()

rm(path.join(config.build.assetsRoot, config.build.assetsSubDirectory), err => {
  if (err) throw err
  webpack(webpackConfig, function (err, stats) {
    spinner.stop()
    if (err) throw err
    process.stdout.write(stats.toString({
      colors: true,
      modules: false,
      children: false,
      chunks: false,
      chunkModules: false
    }) + '\n\n')

    console.log(chalk.cyan('  Build complete.\n'))
    console.log(chalk.yellow(
      '  Tip: built files are meant to be served over an HTTP server.\n' +
      '  Opening index.html over file:// won\'t work.\n'
    ))
  })
})
```

That will ensure that each test starts with clean mocks and modules, as it should be in unit testing.

Wrapping Up

The Jest mocking feature, along with snapshot testing, is the beauty of Jest! It makes very easy what usually is quite hard to test. As a result, you can focus on writing faster and better-isolated tests and keeping your codebase bullet-proof.

Let’s test a running project of what we have done so far in the next lesson.