Lesson Monday

Intermediate JavaScript (/intermediate-javascript) / Test-Driven Development and Environments with JavaScript (/intermediate-javascript/test-driven-development-and-environments-with-javascript) / Setting Up Babel

Text

Before we begin writing our tests, we'll need to add Babel to our application. **Babel** is a tool that is most commonly used for transforming newer versions of JavaScript into older versions of JavaScript. Babel is essential because there are many newer features in JavaScript that older browsers don't understand. However, we still want to use these new JS features because they will make our code cleaner, more efficient and easier to read. In fact, we'll be covering some of these newer JavaScript features over the next few weeks.

Fortunately, we don't have to worry about Chrome understanding many of these newer features — that's because Chrome is an evergreen browser that's already compatible with the features we will be using. However, if we were working on an enterprise application where it's likely some of our customers are using outdated browsers, we'd need to use tools like Babel to make our sites compatible with those browsers.

However, that's not why we need Babel for our projects. After all, we know Chrome is good to go with any new features of JavaScript we'll be using.

Our issue is we are using `import` and `export` statements in our applications from a newer version of ECMAScript, and Jest uses NodeJS's `require()` statements. So, we use Babel to translate the `import` and `export` statements into `require()` statements that Jest needs. So we aren't transforming our code for other users — we are transforming our newer JavaScript syntax into syntax that our test runner Jest will be able to read.

Installing Babel and a Babel Plugin

Let's start by adding Babel to our application. In the root of the Shape Tracker directory, run this command to install Babel.

```
$ npm install @babel/core@7.18.6 --save-dev
```

Next, we'll need to install a specific Babel plugin that will transform ES6 module syntax:

```
$ npm install @babel/plugin-transform-modules-commonjs@7.1
8.6 --save-dev
```

Similar to webpack, Babel uses plugins (https://babeljs.io/docs/en/plugins/) to specify how we want our code to be transformed. In this case, we're installing a plugin that will transform `import` and `export` statements into `require` statements.

Configuring Babel

Finally, we need to set up our Babel configuration. Just as we use an `.eslintrc` file to configure ESLint, we'll use a `.babelrc` file to configure Babel. The file should go in the root directory of the project.

.babelrc

```
{
  "env": {
    "test": {
      "plugins": ["@babel/plugin-transform-modules-commonj
s"]
    }
  }
}
```

The configuration above states that our test environment should use the plugin we just installed to transform ES6 modules into CommonJS modules, which is what Node uses.

This is the only Babel configuration we'll do in this course. Even though we are doing very little with it, it's good to have some exposure since this tool is very common in real world applications.

If you are interested in learning more about using Babel in general, check out the Babel Usage Guide (https://babeljs.io/docs/en/usage/). If you are interested in using Babel with webpack specifically, check out webpack's Babel documentation (https://webpack.js.org/loaders/babel-loader/).

Now we're ready to start writing tests!

Previous (/intermediate-javascript/test-driven-development-and-environments-with-javascript/setting-up-jest)
Next (/intermediate-javascript/test-driven-development-and-environments-with-javascript/tdd-with-jest-testing-the-triangle-constructor)

Lesson 27 of 49
Last updated more than 3 months ago.

disable dark mode

(http://www.epicodus.com)