React – Webpack

In this lab, you will webpack to package an existing React app.

# Objectives

In this lab, you will

* Use an existing React app
* Learn by trial and error
* Use webpack to bundle the app for the browser
* Use webpack to load JSX files
* Use webpack to load \*.less and \*.sass files
* Run the app

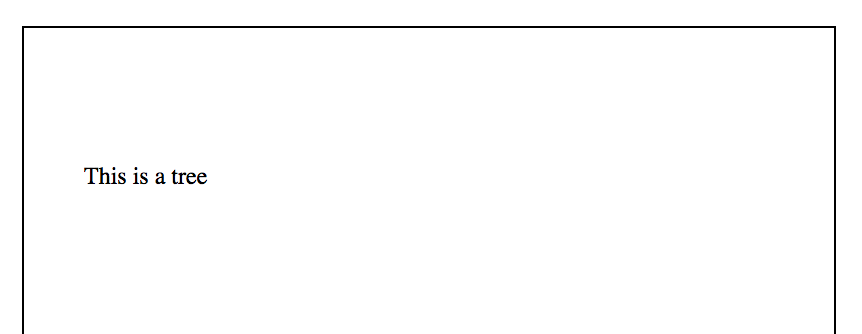
# Run the solution

1. Change to the solution/react-app folder and run

yarn install

yarn start

1. You should see the following page at <http://localhost:3000>



1. The above is simple because the application is simple. This lab is about configuring webpack to produce a bundle of JavaScript including CSS, LESS, SASS, JSX, and plain old JavaScript.
2. Stop the server.

# Examining the JSX Program

1. The index page, at lab/src/index.html, shown below only defines an area for the React code (at ‘react-container’) and loads the bundle.js file created by webpack.



1. In the above, line 11 defines the name and location of the bundled code.
2. The webpack configuration file, lab/webpack.config.js, looks like the following:



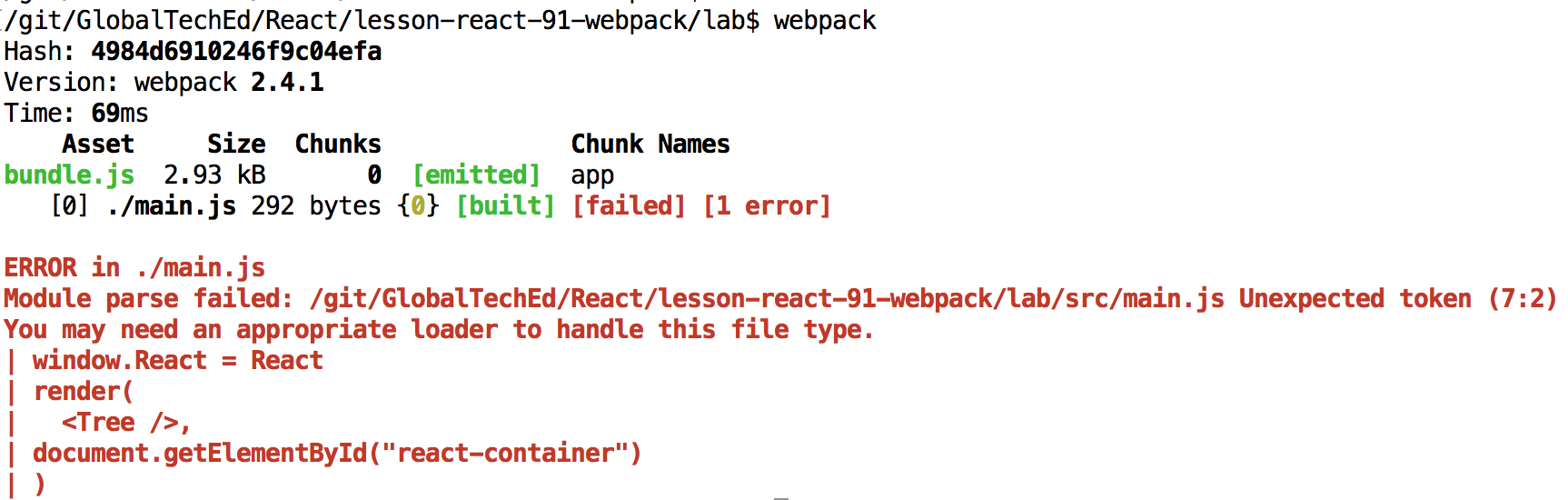
1. In the above, line 6 defines the JS file, lab/src/main.js, where webpack begins loading. Remember, webpack will load this file and ALL THE OTHER files required by it. It uses the require() function or the import method for determining all the dependencies.
2. In line 9 above, it defines the name and location of the bundle file. This must match the expected location (in the <script> tag) in the HTML file.

# Create the bundle.js file

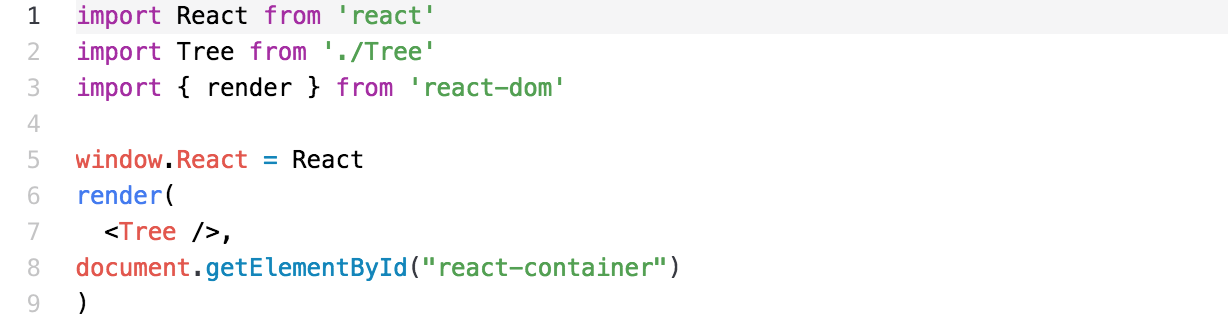
1. Let’s attempt to build the first bundle. Use webpack as follows:

webpack

1. Whoops there is an error shown below:



1. In the above, webpack does not know how to parse ./main.js. It fails at line 7, column 2 with the token, <Tree />.
2. Let’s examine ./main.js below:



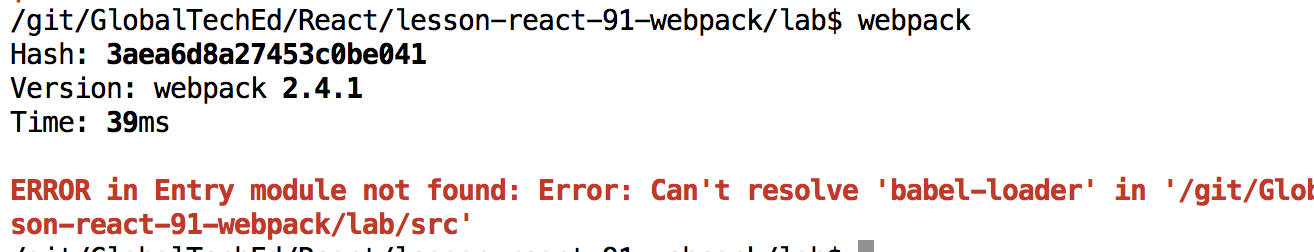
1. Hmmmmm… It appears that webpack does not know how to parse JSX.

# Parsing JSX

1. We depend on babel to parse JSX for us. So, let’s configure webpack to use babel.
2. We actually want babel to pre-process three attributes of JS so that browsers can handle it. Babel compiles the source code into the ES-5 version of JavaScript. Most React code is written in ES-6 (or JS2015). The configuration for babel is shown below:



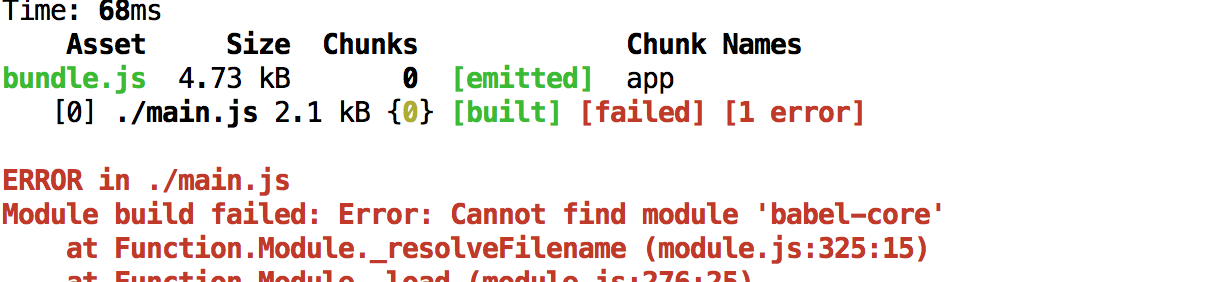
1. In the above, line 15 tells webpack to apply this transformation rule to all files named \*.js or \*.jsx. Well, all except files living in the folder, node\_modules.
2. The transformation is the loader, babel-loader, with the options, es2015 and react.
3. Let’s run webpack again:



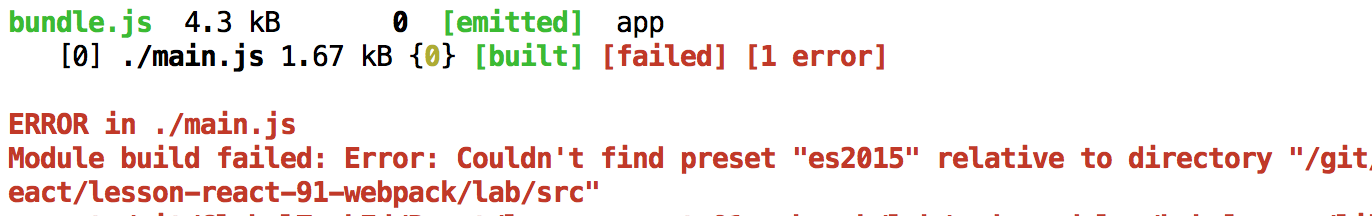
1. Another error. This time, webpack can’t resolve the module, babel-loader. Add this module to the project with:

yarn add babel-loader

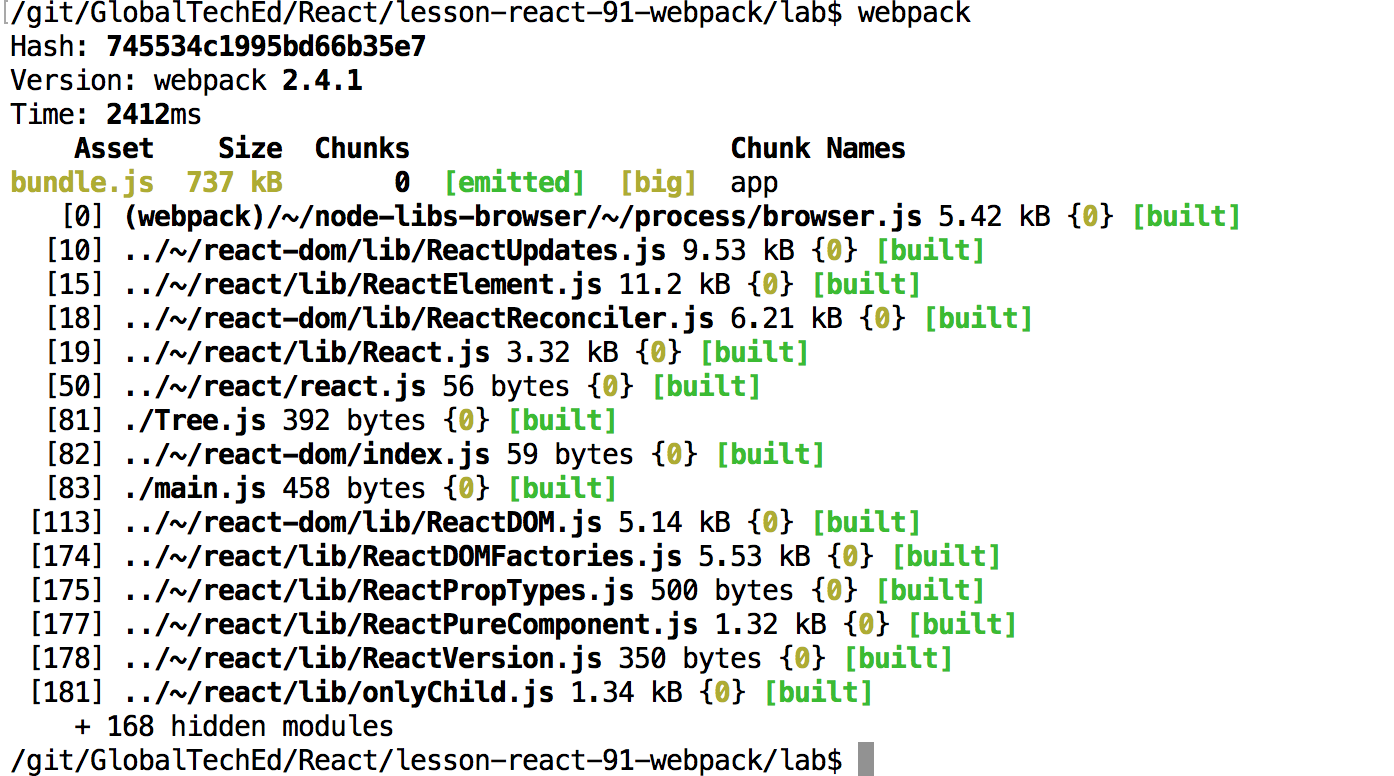
Run webpack and discover another error.



1. Load the babel-core module and try again.



1. We are getting there. Now load the module, babel-preset-es2015 and try again.
2. This time it fails because of the other preset, react. So load the module, babel-preset-react and try again.



1. Finally. It succeeded. Webpack loaded almost 200 modules to create bundle.js.

# Configuring the webpack-dev-server

1. Webpack comes with a development server with Hot Module Replacment allowing the server to automatically compile and reload files that we edit. It is very handy. Configure the server by modifying the config file as shown below:



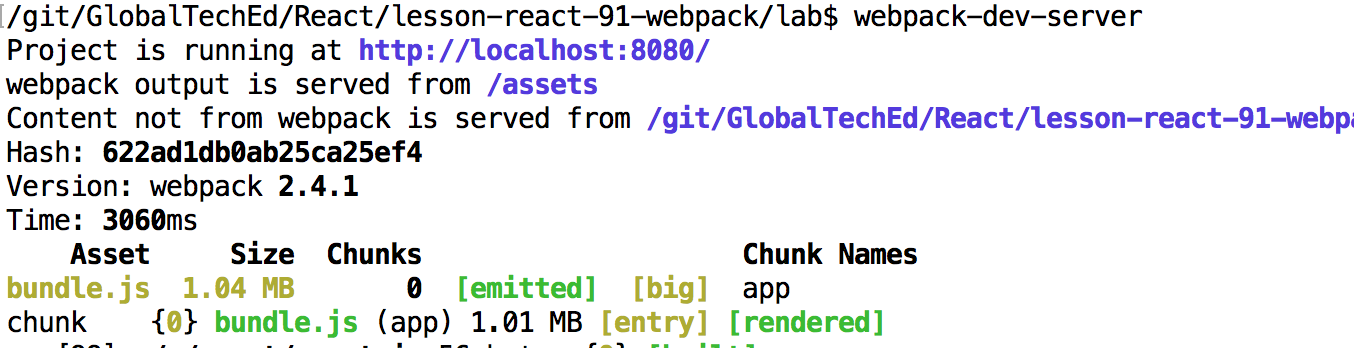
1. Start the development server with:

webpack-dev-server

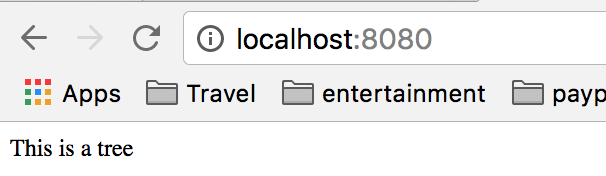
1. NOTE: you might need to install it globally with:

yarn global add webpack-dev-server

1. You should see the following:



1. In the above, the webpack-dev-server bundles the project and starts an embedded Express application running on <http://localhost:8080>. Open this page and see the following:



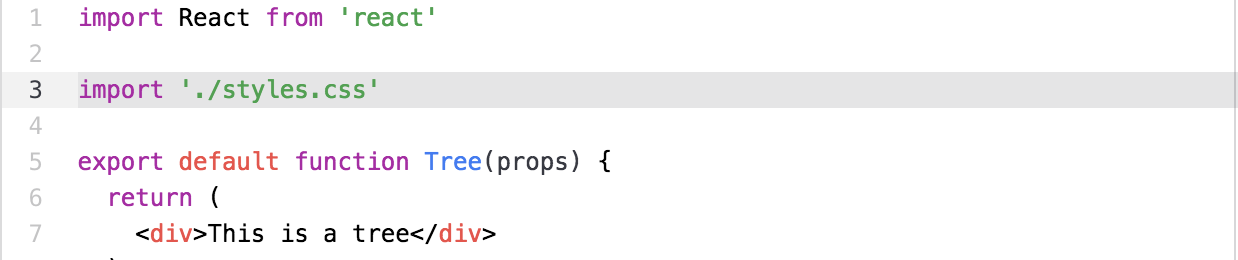
1. This results from the <Tree /> component as shown:



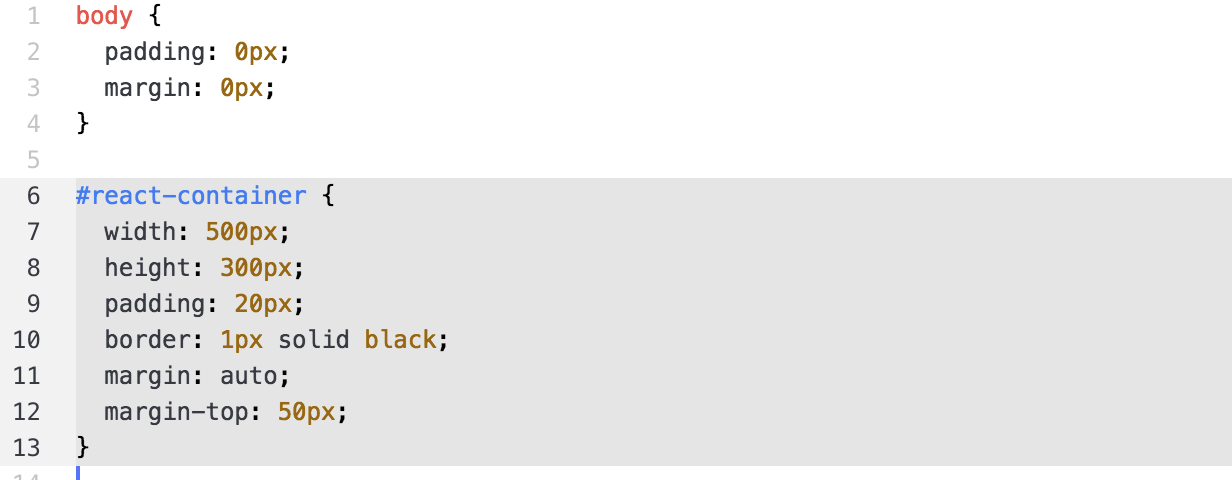
1. Notice there is no CSS styling involved with the <div>

# Loading CSS Files

1. Modify the lab/src/Tree.js file to import a CSS style sheet as follows:



1. The file, lab/src/styles.css, contains the following CSS rules.



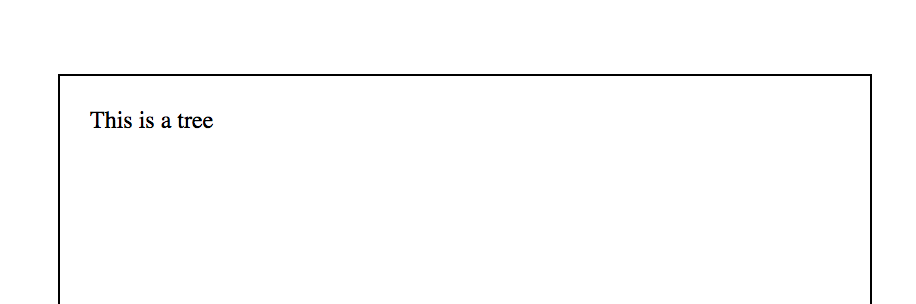
1. In the above, line 6 defines the styles for the ‘react-container’. When we save the Tree.js file, the server detects an error.



1. Webpack does not know how to process a file named, \*.css. We need another rule as shown below: NOTE: place this rule before the babel compiler because babel does NOT know how to handle the file.



1. In the above, line 20 defines two pre-processors. The last one defined is the first one executed. The processor, css-loader, understands the instruction, import ‘./styles.css’ and loads the contents of the file. The next processor, style-loader, creates the JavaScript code required to generate a <style> tag and inject the CSS data into the <head> tag in the HTML file.
2. NOTE: When we change the webpack.config.js file, we must re-start the webpack-dev-server. The server automatically notices when other files get modified, but does not recognize changes to this file.
3. Examine the browser again:



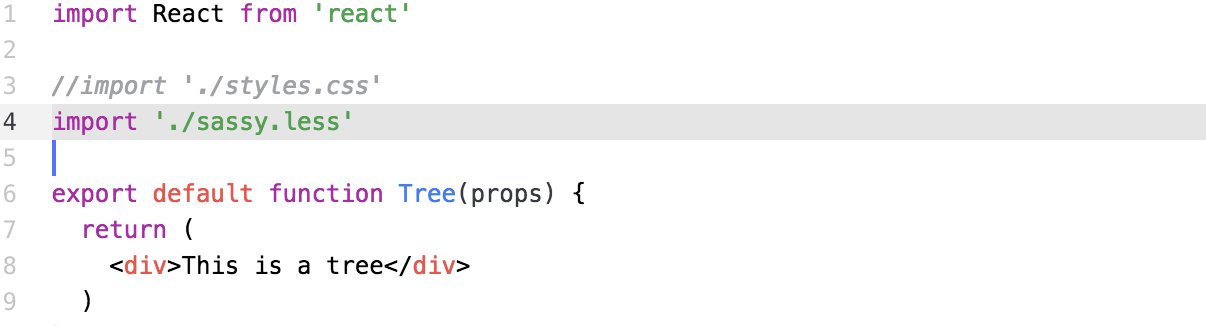
1. We have styling!!!

# Process LESS and SASS Files

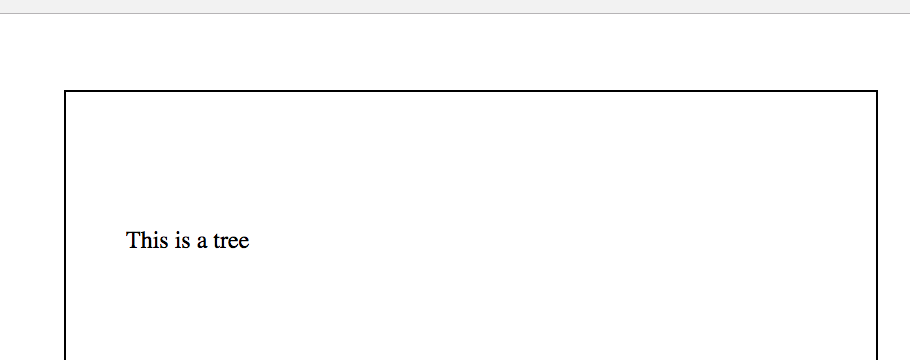
1. Add the following code to enable processing of LESS and SASS files.



1. We will need to add the new modules, less, less-loader and sass-loader.
2. Change the Tree.js file to import a sassy.less style sheet as follows:



1. Restart the server and view the new page:



Congratulations. You have completed this lab.