|  |
| --- |
| Webpack |
| **Introduction to Webpack**  **(A module bundler)** |
| Version 1.0 |
|  |

Please make reference to **NTNI Document Creation Guidelines** when completing this document.

**Amendment Record Sheet**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version No.** | **Date** | **Amendment Description** | **Amended By** | **Approved By** |
| 1.0 | 4 Feb 2017 | Approved document | Gerry Byrne | David Wilson |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Topic Overview**

This topic will introduce you to Webpack which is a very popular module bundler. Webpack takes modules with dependencies and generates static assets representing those modules.

To use Webpack we will install it through the Node Package Manager (npm) on our system.

**Topic Objectives**

At the end of this topic you will be able to:

* **Install Webpack on a computer (Mac or PC)**
* **Understand the role of Webpack in developing applications**
* **Use Webpack**
* **Understand and use dependencies in the Webpack config file**

Contents

[Webpack 7](#_Toc483205759)

[Exercise – Initialising our project that will use Webpack 8](#_Toc483205760)

[Exercise – Installing Webpack Locally 10](#_Toc483205761)

[Exercise – Creating the Webpack Config File 12](#_Toc483205762)

[Exercise – Creating the index.js JavaScript File 14](#_Toc483205763)

[Exercise – Creating the index.html File 16](#_Toc483205764)

[Project Structure 17](#_Toc483205765)

[Exercise – Creating the bundle.js File 18](#_Toc483205766)

[Exercise – Run the HTML File 20](#_Toc483205767)

[Exercise- Multiple modules 21](#_Toc483205768)

[- Creating the second.js JavaScript File 21](#_Toc483205769)

[Exercise – Adding a reference to the second.js file in the index.js file 23](#_Toc483205770)

[Exercise – Creating the bundle.js File 24](#_Toc483205771)

[Exercise – Run the HTML File 25](#_Toc483205772)

[Exercise – Using ReactJS, Webpack and a Babel-loader 27](#_Toc483205773)

[Exercise – Creating the Project Setup 28](#_Toc483205774)

[Exercise – Creating the React components 31](#_Toc483205775)

[Creating the Welcome.html File 34](#_Toc483205776)

[Creating the webpack.config.js File 35](#_Toc483205777)

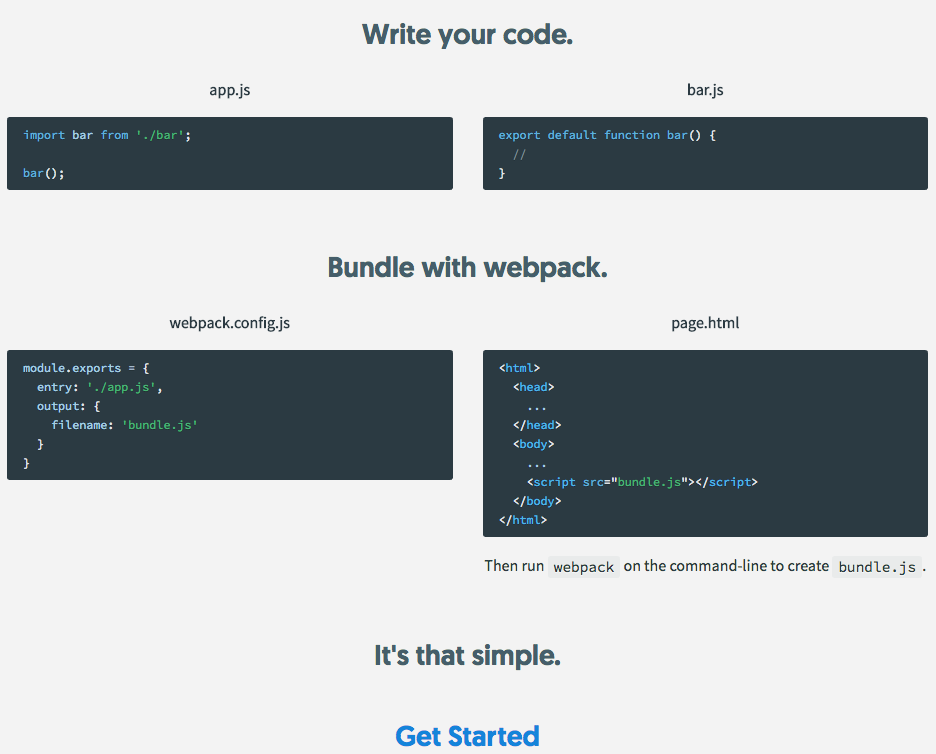
[References: 39](#_Toc483205778)

|  |
| --- |
| **Webpack** |
|  |

# Webpack

Webpack is a powerful module bundler. A bundle is a JavaScript file that incorporates assets that belong together and should be served to the client in a response to a single file request. A bundle can include JavaScript, CSS styles, HTML and almost any other kind of file. Webpack roams over our application source code, looking for import statements, building a dependency graph and emitting one (or more) bundles. With plugins and rules, Webpack can preprocess and minify different non-JavaScript files such as TypeScript, SASS, and LESS files.

Webpack is driven by configuration, normally named as *webpack.config.js*. That's the heart of it all. The configuration defines the inputs and the outputs of our project. Most importantly, it describes the types of transformations we perform. These transformations are defined using **loaders** and **plugins**, each of which serves a purpose of its own. The diagrammatical representation of the Webpack process is shown below:



# Exercise – Initialising our project that will use Webpack

We need to create a directory for our project and we will also initiliase Node Package Manager (npm).

1. Open a Terminal window.
2. Navigate to the location where you wish to create your project e.g.

**cd /Users** (then press Enter)

**cd /gerard** (then press Enter)

**cd Desktop** (then press Enter)

1. Enter the following command at the terminal prompt:

**mkdir reactwebpackproject && cd reactwebpackproject**

1. Press the Enter key.
2. Enter the following command at the terminal prompt:

**ls**

1. Press the Enter key.
2. The display will show there are no files just the directory.

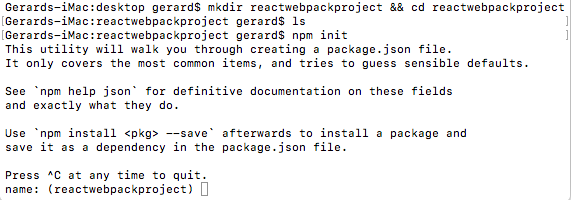


1. Enter the following command at the terminal prompt:

**npm init**

1. Press the Enter key.

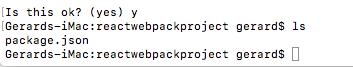
The display will show a message from mpm telling you about the setting up of a package.json file

.

1. For the name type **reactwebpackproject**.
2. Press the Enter key.
3. For the version Press the Enter key.
4. For the description type **my first react and webpack project**
5. Press the Enter key.
6. For the entry point type **index.js** or simply press the Enter key.
7. Press the Enter key.
8. For the test command simply press the Enter key.
9. For the git repository simply press the Enter key.
10. For the keywords type **react webpack**.
11. Press the Enter key.
12. For the author type **Gerry Byrne.**
13. Press the Enter key.
14. For the licence simply press the Enter key.
15. For the question **Is this ok? (yes)** simply press the Enter key.
16. Enter the following command at the terminal prompt:

**ls (dir)**

1. Press the Enter key.
2. The display will show there is one file in the directory – package.json.



# Exercise – Installing Webpack Locally

1. Navigate to the location where you created your project.
2. Enter the following command at the terminal prompt:

**npm install webpack**

(You may need to enter **sudo npm install webpack**)

1. Press the Enter key.
2. Look at the output that is displayed. It should be something similar to the following:



This gives us information about our installation.

1. Enter the following command at the terminal prompt:

**ls (dir)**

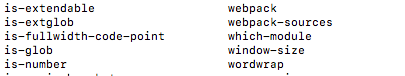
1. Press the Enter key.
2. Look at the output that is displayed. It should be something similar to the following showing us that the folder node\_modules has been created:



1. Enter the following command at the terminal prompt:

**ls node\_modules (dir node\_modules**)

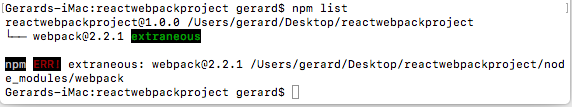
1. Press the Enter key.
2. Look at the output that is displayed. It should be something similar to the following showing us the materials that have been added to our node modules. We should see webpack:



1. Enter the following command at the terminal prompt:

**npm list**

1. Press the Enter key.
2. Look at the output that is displayed. It should be something similar to the following showing us the the locally installed packages. We should certainly see webpack:



# Exercise – Creating the Webpack Config File

1. Navigate to the location where you created your project.
2. Enter the following command at the terminal prompt:

**cat >webpack.config.js**

(or use a text editor to create the file and save it in this directory)

1. Press the Enter key.
2. Add the following lines to the config file we have just created:

**module.exports = {**

**entry: './index.js',**

**output: {**

**filename: 'bundle.js'**

**}**

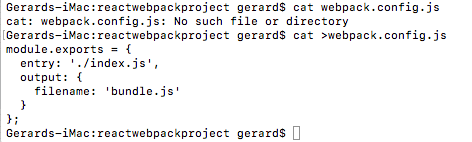
**};**

This gives us information about our bundling requirements. It tells Webpack that the input file is called **index.js** and we wish to outpt the bundle to a file called **bundle.js**.

1. Enter the following command at the terminal prompt to indicate that we have finished entering our data and wish to terminate the process, saving the file:

**Ctrl + D**

1. The display will now show the following:



1. Enter the following command at the terminal prompt :

**ls (dir)**

1. Look at the output that is displayed. It should be something similar to the following showing us that the webconfig file has been created:



# Exercise – Creating the index.js JavaScript File

1. Navigate to the location where you created your project.
2. Enter the following command at the terminal prompt:

**cat >index.js**

(or use a text editor to create the file and save it in this directory)

1. Press the Enter key.
2. Add the following lines to the index file we have just created:

**document.write(“My first React and Webpack project”);**

This will therefore simply display the message on the browser.

1. Enter the following command at the terminal prompt to indicate that we have finished entering our data and wish to terminate the process, saving the file:

**Ctrl + D**

1. The display will now show the following:



1. Enter the following command at the terminal prompt :

**ls**

1. Look at the output that is displayed. It should be something similar to the following showing us that the index.js file has been created:



# Exercise – Creating the index.html File

1. Navigate to the location where you created your project.
2. Enter the following command at the terminal prompt:

**cat >index.html**

(or use a text editor to create the file and save it in this directory)

1. Press the Enter key.
2. Add the following lines to the html file we have just created:

**<html>**

**<head>**

**<script type="text/javascript" src="bundle.js"></script>**

**</head>**

**<body>**

**<h1>Gerry Byrne - React and Webpack Project</h1>**

**</body>**

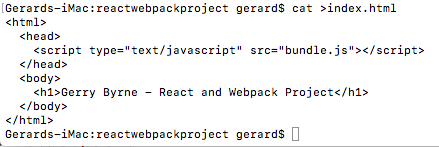
**</html>**

This will therefore simply run the bundle.js file which will be the bundle that is holding the index.js file we created.

1. Enter the following command at the terminal prompt to indicate that we have finished entering our data and wish to terminate the process, saving the file:

**Ctrl + D**

1. The display will now show the following:



1. Enter the following command at the terminal prompt :

**ls**

1. Look at the output that is displayed. It should be something similar to the following showing us that the index.js file has been created:



# Project Structure

In real-world webpack projects we will separate the source files from the bundled files by organising them in folders. The example below shows that the source files are in the **src folder** and the bundled files are in **bin** folder. The Node modules are in a separate directory created when they are installed.

Our final project structure will look like this:



# Exercise – Creating the bundle.js File

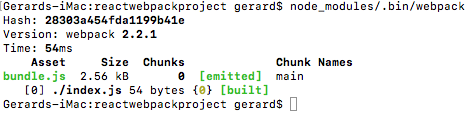
1. Navigate to the location where you created your project.
2. Enter the following command at the terminal prompt:

**node\_modules/.bin/webpack webpack**

1. Press the Enter key.

This will now look in the web.config.js and find that the file to be bundled is called index.js which is in the current directory (./). It will also know that the output will be to a file called bundle.js (in this case to the current directory).

1. The display will now show the following:



**Note:**

We should not have to use the pathname ./node\_modules/.bin. We should be able to make locally-installed node modules executable by using their name. This can be achieved by adding ./node\_modules/.bin to the $PATH.

1. Enter the following command at the terminal prompt to see what paths exist:

**Export $PATH**

1. Look at the output that is displayed. It should be something similar to the following showing us that paths exist and what they are:



1. Enter the following command at the terminal prompt to add a new path:

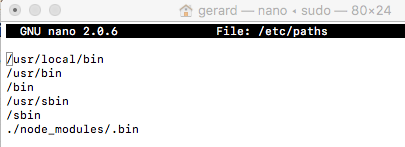
**sudo nano /etc/paths**

1. Press the Enter key.
2. Enter the password when prompted.

This paths file will appear and we can now add the additional path to the bottom of the text.

1. Add the following line to the script:

**./node\_modules/.bin**



1. Enter the following command at the terminal prompt to indicate that we have finished entering our data and wish to terminate the process, saving the file:

**Ctrl + X**

1. The display will now show the following:



1. Type Y at the command prompt.
2. Press the Enter key.

# Exercise – Run the HTML File

1. Start a server by entering the following command at the terminal prompt:

**python –m SimpleHTTPServer 8080**

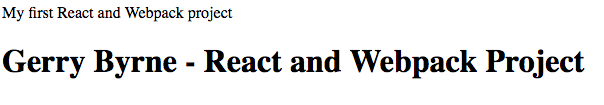
1. Press the Enter key.

This will start the server on port 8080.

1. Open the browser of your choice.
2. In the address bar type:

**localhost:8080**

1. Navigate to the project directory.
2. Click on the index.html file, if required.
3. Look at the output that is displayed. It should be as shown below:



# Exercise - Multiple modules

# - Creating the second.js JavaScript File

1. Navigate to the location where you created your project.
2. Enter the following command at the terminal prompt:

**cat >second.js**

(or use a text editor to create the file and save it in this directory)

1. Press the Enter key.
2. Add the following lines to the index file we have just created:

**module.exports = document.write(“This is from the second JavaScript file”);**

This will therefore simply display the message on the browser.

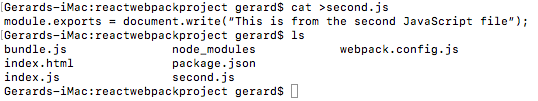
1. Enter the following command at the terminal prompt to indicate that we have finished entering our data and wish to terminate the process, saving the file:

**Ctrl + D**

1. The display will now show the following:
2. Enter the following command at the terminal prompt :

**ls**

1. Look at the output that is displayed. It should be something similar to the following showing us that the index.js file has been created:



# Exercise – Adding a reference to the second.js file in the index.js file

1. Navigate to the location where you created your project.
2. Open the index.js file in your editor.
3. Add the following line to the start of the file:

**require("./second.js");**

**document.write("My first React and Webpack project");**

1. Save the file.

# Exercise – Creating the bundle.js File

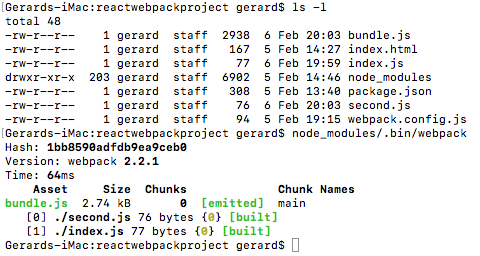
1. Navigate to the location where you created your project.
2. Enter the following command at the terminal prompt:

**node\_modules/.bin/webpack**

1. Press the Enter key.

This will now look in the web.config.js and find that the file to be bundled is called index.js which is in the current directory (./). This index.js file will point to the second.js file. It will also know that the output will be to a file called bundle.js (in this case to the current directory).

1. The display will now show the following:



# Exercise – Run the HTML File

1. Start a server by entering the following command at the terminal prompt:

**python –m SimpleHTTPServer 8080**

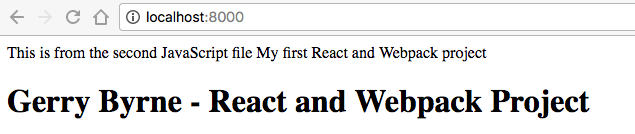
1. Press the Enter key.

This will start the server on port 8080.

1. Open the browser of your choice.
2. In the address bar type:

**localhost:8080**

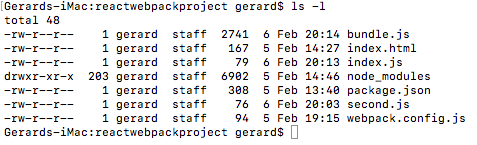
1. Navigate to the project directory.
2. Click on the index.html file, if required.
3. Look at the output that is displayed. It should be as shown below:



1. Enter the following command at the terminal prompt:

**ls -l**

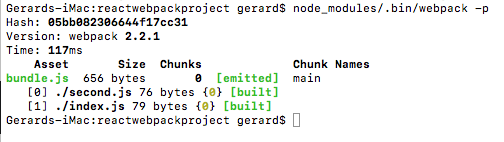
1. Look at the output that is displayed. It should be something similar to the following showing us the bundle.js file size:



1. Enter the following command at the terminal prompt:

**node\_modules/.bin/webpack -p**

1. Look at the size of the bundle.js file now.



Now your bundle.js is minified.

# Exercise – Using ReactJS, Webpack and a Babel-loader

**ReactJS**for front end development is the V in MVC. React allows us to build reusable UI components and when we combine it with a bundler utility like **Webpack**, ReactJS greatly simplifies building and maintaining Single Page Applications.

Facebook has kept React up to date and has made it compatible with new features from ECMAScript 6 (ES6). Unfortunately browser support for ES6 is not widespread but this is where utilities like **Babel**come into play. Babel lets us write code that uses new ES6 features and then transpiles that code into standard ES5 code that can run in older JavaScript environments.

In this example we will set up two basic React components that use ES6 features, then use Babel to transpile them to ES5 and bundle them using Webpack. React, ES6, Babel and Webpack are ideally suited to each other, they are good partners.

What we need for this exercise will be:

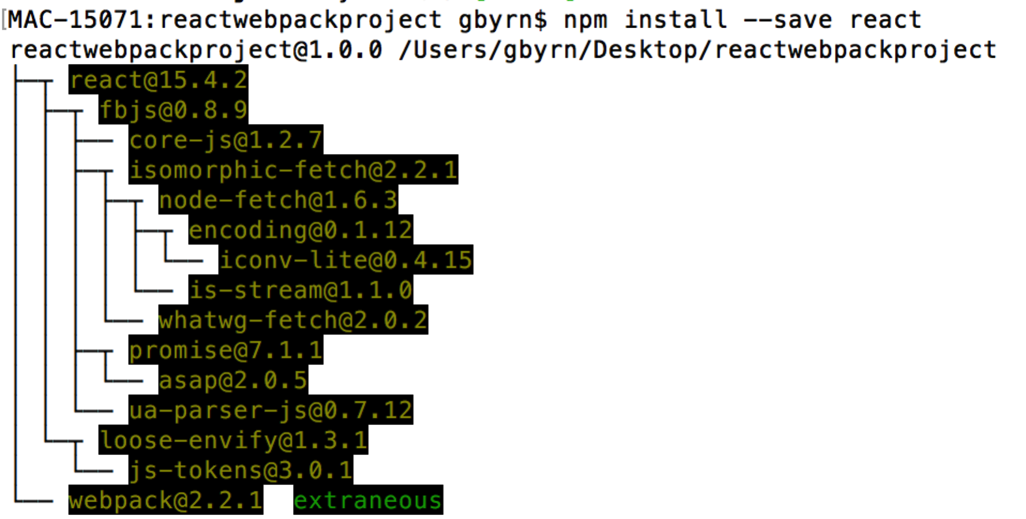
* NodeJS
* Node Package Manager (npm)
* React
* React-DOM
* Webpack
* Webpack dev server
* Babel loader
* Babel core
* Babel preset es2015
* Babel preset react

# Exercise – Creating the Project Setup

1. Navigate to the location where you created your project.
2. Enter the following command at the terminal prompt:

**npm install --save react**

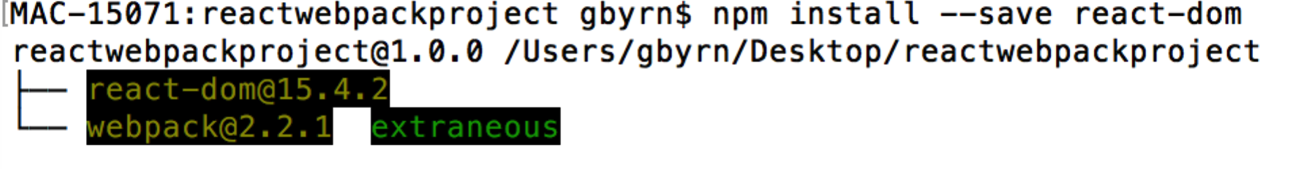
1. Press the Enter key.
2. The display will now show the following:

­­­­­­­

1. Enter the following command at the terminal prompt:

**npm install --save react-dom**

1. Press the Enter key.
2. The display will now show the following:

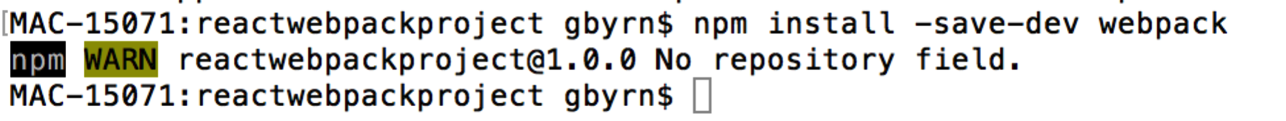


We will also need to install Webpack and the Webpack development server for serving our bundled JavaScript application. We may need to use “sudo” to install the dev server package globally.

1. Enter the following command at the terminal prompt:

**npm install –save-dev webpack**

1. Press the Enter key.
2. The display will now show the following:

 ­­­­­­­

1. Enter the following command at the terminal prompt:

**npm install webpack-dev-server –save-dev**

1. Press the Enter key.

We have the bundling we need but now we need a transpiler for interpreting our ES6 code. Babel will be the tool we will use so we need to install the babel-loader and babel-core packages that we will use to work with Webpack, as well as the ES2015 and React presets for loading the code that we’ll write.

1. ­­­­­­­ Enter the following command at the terminal prompt:

**npm install --save-dev babel-loader**

1. Press the Enter key.
2. ­­­­­­­ Enter the following command at the terminal prompt:

**npm install --save-dev babel-core**

1. Press the Enter key.
2. ­­­­Enter the following command at the terminal prompt:

**npm install --save-dev babel-preset-es2015**

1. Press the Enter key.
2. ­­­­­­­ Enter the following command at the terminal prompt:

**npm install --save-dev babel-preset-react**

1. Press the Enter key.

# Exercise – Creating the React components

We should know that components in React are the individual building blocks of how our data is viewed. In React we write components to handle how our data should look and to automatically render state changes. So now we will write a simple component.

1. Navigate to the location where you created your project.
2. Open your text editor.
3. Create a new JSX file called **MyWelcomePage.jsx** saving it in the project directory.

**import React from 'react';**

**class Welcome extends React.Component {**

**render() {**

**return <h1>Welcome to my React Project</h1>**

**}**

**}**

1. Save the file.

**Analysis**

* we have ES6 import statements and class definitions, which makes our code more concise by not having to call React.createClass.
* there is also some inline HTML type stuff in the component class definition’s render function.
* This XML-like syntax being returned from the function is called JSX.
* It was designed to make building React components easier because it is concise and familiar for defining tree structures with attributes.

This is new syntax and as not all browsers are ready for such JSX syntax we will need to use Babel to transpile both the ES6 syntax and the JSX syntax into ES5 JavaScript that can be run in a browser.

The code below shows what our Welcome component would look like without using ES6 and JSX:

**var React = require('react');**

**var Welcome = React.createClass({displayName: Welcome,**

**render: function() {**

**return React.createElement("h1", null, " Welcome to my React Project ");**

**}**

**});**

When we use JSX, we are able to define our virtual DOM element more concisely without having to call React.createElement and passing which attributes the element should have. Our simple Hello component may have the same number of lines of code but JSX makes things much easier as we continue building components and combine them together.

Now that we have our component class, we need to add some code to “mount” this component to a DOM element. This will take our React component and render it to display within an element of an HTML page. To do this we import the React DOM and call its render function, passing in a component object as well as an actual DOM element to attach to.

1. Open the **MyWelcomePage.jsx** file.
2. Amend the code as shown below:

F

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**class Welcome extends React.Component {**

**render() {**

**return <h1>Welcome to my React Project</h1>**

**}**

**}**

**ReactDOM.render(<Welcome/>, document.getElementById(‘element1goeshere’));**

1. Save the file.

Now we will create our second component whose job is to render another phrase, in this case it will be “- my name is Gerry Byrne”.

1. Open your text editor.
2. Create a new JSX file called **MyNameDetailsPage.jsx** saving it in the project directory.
3. Amend the code as shown below:

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**class NameDetails extends React.Component {**

**render() {**

**return <h1>- my name is Gerry Byrne</h1>**

**}**

**}**

**ReactDOM.render(<NameDetails/>, document.getElementById(‘element2goeshere’));**

1. Save the file.

Now we have two React components, and we will want to diplay them on a HTML page that contains a <div> for each component we want to mount. Create an “index.html” file and write this bare bones web page:

# Creating the Welcome.html File

1. Open your text editor.
2. Create a new HTML file called **Welcome.html** saving it in the project directory.
3. Amend the code as shown below:

**<!doctype html>**

**<html>**

**<head>**

**<meta charset="UTF-8">**

**<title>Gerry Byrne – React, Webpack and Babel</title>**

**</head>**

**<body>**

**<div id="element1goeshere"></div>**

**<div id="element2goeshere"></div>**

**</body>**

1. Save the file.

This is great as we have now created all the code we need to display a message, made from two files, in our browser. Now we need to use Webpack to bundle the files into one file which we will call bundle.js.

# Creating the webpack.config.js File

1. Open your text editor.
2. Create a new JavaScript file called **welcomeindex.js** saving it in the project directory.
3. Amend the code as shown below:

**import Hello from './MyWelcomePage.jsx';**

**import World from './MyNameDetailsPage.jsx';**

1. Create a new JavaScript file called **webpack.config.js** saving it in the project directory.
2. Amend the code as shown below:

**var path = require('path');**

**var webpack = require('webpack');**

**module.exports = {**

**entry: './welcomeindex.js',**

**output: { path: \_\_dirname, filename: 'bundle.js' },**

**module: {**

**loaders: [**

**{**

**test: /.jsx?$/,**

**loader: 'babel-loader',**

**exclude: /node\_modules/,**

**query: {**

**presets: ['es2015', 'react']**

**}**

**}**

**]**

**},**

**};**

1. Save the file.
2. Open the file called **Welcome.html** in the project directory.
3. Amend the code as shown below:

**<!doctype html>**

**<html>**

**<head>**

**<meta charset="UTF-8">**

**<title>Gerry Byrne – React, Webpack and Babel</title>**

**</head>**

**<body>**

**<div id="element1goeshere"></div>**

**<div id="element2goeshere"></div>**

**<script src="bundle.js"></script>**

**</body>**

1. Save the file.
2. Enter the following command at the terminal prompt:

**node\_modules/.bin/webpack**

1. Press the Enter key.
2. Start a server by entering the following command at the terminal prompt:

**python –m SimpleHTTPServer 8080**

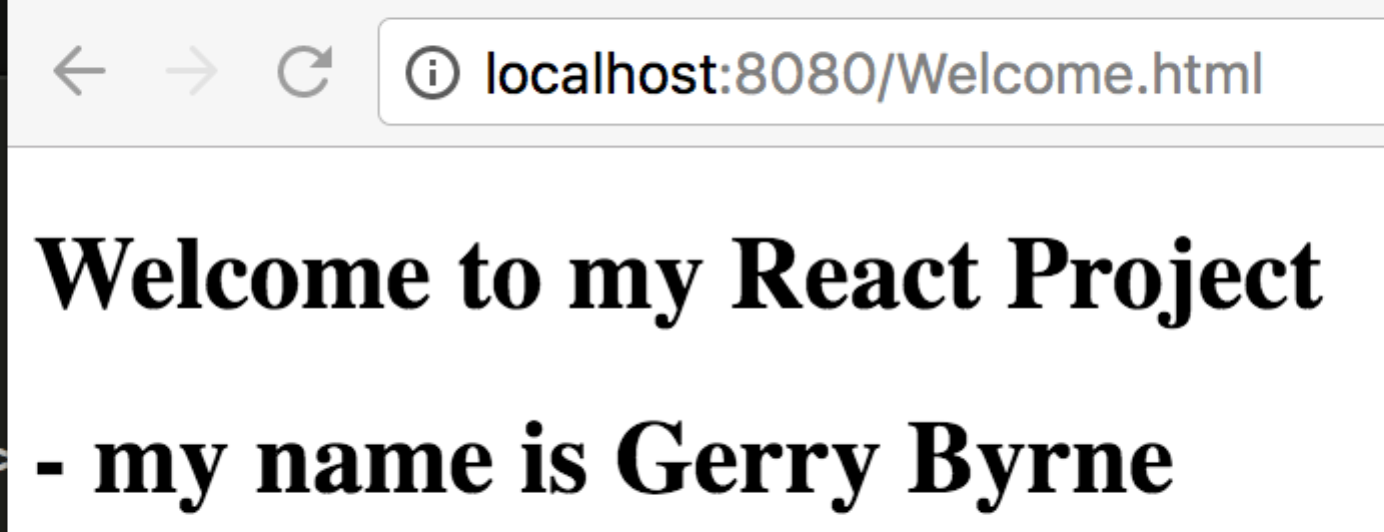
1. Press the Enter key.

This will start the server on port 8080.

1. Open the browser of your choice.
2. In the address bar type:

**localhost:8080/Welcome.html**

1. Press the Enter key.
2. The display will now show the following:



The two files we have created have been bundled and displayed through the welcome.html page which calls the bundle.js file.

Note at step 25 we could have typed:

**node\_modules/.bin/webpack –d --watch**

This would have started a server, which will be listening for connections at localhost on port 8080.

Now, when you change something in our assets, it should live-reload the files.

1. Simply change the MyNameDetailsPage.jsx file as shown below:

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**class NameDetails extends React.Component {**

**render() {**

**return <h1>- my name is Gerry Byrne and I live in Belfast where I have lived for my whole life</h1>**

**}**

**}**

**ReactDOM.render(<NameDetails/>, document.getElementById('element2goeshere'));**

1. Save the file.
2. Open the browser again and refresh the page.
3. The new details will be displayed:



# References:

|  |
| --- |
|  |
| <http://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js> |
| <https://www.sitepoint.com/beginners-guide-node-package-manager/> |
| <https://angular.io/docs/ts/latest/guide/webpack.html> |
| <https://web-design-weekly.com/2014/09/24/diving-webpack/> |
| <https://robots.thoughtbot.com/setting-up-webpack-for-react-and-hot-module-replacement> |
| <https://webpack.github.io/docs/tutorials/getting-started/> |
| <https://www.codementor.io/tamizhvendan/beginner-guide-setup-reactjs-environment-npm-babel-6-webpack-du107r9zr> |
| <https://www.twilio.com/blog/2015/08/setting-up-react-for-es6-with-webpack-and-babel-2.html> |
| <https://webpack.github.io/docs/usage.html> |
| <https://web-design-weekly.com/2014/09/24/diving-webpack/> |
| <https://angular.io/docs/ts/latest/guide/webpack.html> |
| <https://robots.thoughtbot.com/setting-up-webpack-for-react-and-hot-module-replacement> |