Exercise 2 – Setting Up a ReactJS Project

Objective

To be able to set up a ReactJS project using npm (Node Package Manager), Babel and Webpack

Overview

This exercise will give you the commands and information needed to set up a ReactJS project using the command line (or terminal) interface and then check to see if the setup has been successful.

Part 1 – Installing the required packages

1.1. On the command line or terminal, navigate to the folder:

Exercises/EG02_SettingUpReactJS/Starter

1.2. Install the react packages by entering the following:

```
npm install react react-dom
```

This will place the **node_modules** folder into the project. This folder includes the react and react-dom packages.

1.3. Install **babel** and its required plugins, by entering the following:

```
npm install babel-core babel-loader babel-preset-react
babel-preset-es2015
```

This process adds to the **node_modules** folder with all of the babel files.

1.4. Install the **webpack** module bundler and **development server** by typing:

```
npm install webpack webpack-dev-server
```

The server provides a development environment that will be configured to update automatically. Again, this process adds relevant files to the **node_modules** folder.

1.5. On the command line or terminal enter:

```
npm init -y
```

This effectively creates a **package.json** file in the folder. The **–y** extension automatically accepts the defaults for each key.

Part 2 – Creating the base files

2.1. In a suitable text editor, open **Exercises/EG02_SettingUpReactJS/starter** as a project and then create **index.html** in its root. Input the following code:

```html
<!DOCTYPE html>

<html>

 <head>

    <meta charset="utf-8" />

    <title>My First React App</title>

 </head>

 <body>

    <div id="app"></div>

    <script src="index.js"></script>

 </body>

</html>
```

2.2. Create **main.js** in the project root and enter:

```js
import React from 'react';

import ReactDOM from 'react-dom';


ReactDOM.render(

    <h1>Hello World</h1>,

    document.querySelector('#app')

);
```

These two files essentially create a page to view (**index.html**) and then some content that will be placed into the div with the id of app (through the render method in **main.js**). We will discuss this further later in the course.

Part 3 – Configure Webpack

3.1. Create a file in the project root called **webpack.config.js**.

3.2. Open this file for editing and then type the following code:

```
module.exports = {
 // Identify the entry file
 entry: './main.js',
 output: {
     path: __dirname,
     filename: 'index.js'
 },
resolve: {
     // Be able to import from file regardless of extension
     extensions: ['.js', '.jsx']
}
 // Set up the webpack-dev-server for use
 devServer: {
     inline: true,  // Auto-refresh page on the fly
     port: 8080     // Arbitrarily chosen for demo
 },
 module: {
     loaders: [
         {
             // All files that end with '.js' and '.jsx'
             test: /\.jsx?$/,
             // Do not use files in node_modules folder
             exclude: /node_modules/,
             // Use babel as the loader (-loader needed
             // from webpack 2.3.0+)
             loader: 'babel-loader',
```

```
                // Pass arguments/queries to the loader

                query: {

                        presets: ['es2015', 'react']

                }

            }

        ]

    }

}
```

3.3. Open the **package.json** file and replace the line "*test*" in "*scripts*" with:

```
"start": "webpack-dev-server --hot"
```

Save the file.

3.4. Build the application by running the following from the command line or terminal:

```
webpack
```

This essentially compiles the application for first running. The **index.js** file that is created can be deleted as it will subsequently be created and exist on the server when the application is running. Have a look in the file, it should have a lot of JavaScript that has been automatically generated.

3.5. To view the application, execute the following on the command line or terminal:

```
npm start
```

Open a browser and navigate to:

```
localhost:8080
```

'Hello, World!' should be displayed in the browser window.

Part 4 – Checking the refresh works

4.1. Open the **main.js** file and replace the text "Hello, World!" with your own message.

4.2. Switch back to the browser and observe that the text has changed. If it hasn't and you can't work out why, call your instructor.