Exercise 15 – The Modern Developer Toolset

Objectives

To be able to set up a development environment so that modern JavaScript can be written and then transpiled to allow older browsers to be able to execute the intended code.

Overview

So far, we have relied on the fact that the browser we are using complies to ES2015+ standards. This is OK if we are managing the environment our JavaScript is being executed in but in practice, we will have a plethora of users, using a plethora of browsers of varying support for modern JavaScript. To combat this, we are going to set up a development environment that will convert the JavaScript we write into ES5 JavaScript so as long as the browser met ES5 standards, our code will work. The development environment will use Babel to transpile and the module bundler, Webpack to put the JavaScript into a file.

Exercise Set Up

1. In VSCode, navigate to the **Starter** folder inside **Ex15** of the **Exercises** folder.
2. If you have rebooted your computer or stopped *live-server* from running since the last exercise, then use VSCode's integrated terminal to navigate to the **Exercises** folder and run the command **live-server**.
3. Open your browser (if it doesn't automatically) and navigate to **http://localhost:8080** and follow the path to open **Ex15→Starter.**

Exercise Instructions

To use Babel and Webpack, we will need to install them for our project. To install them, we will use Node Package Manager which is installed when NodeJS is installed.

1. Check that Node and NPM are both installed globally by accessing the command prompt and using **node -v** and **npm -v** as commands - both should return a version number.

NPM needs a project to be initialised so that it knows where to install local dependencies.

2. Add another integrated terminal in VSCode and navigate to the **Starter** folder.
3. Run **npm init -y**.

The switch -y on this command accepts all of the default settings for a project initialisation.

4. Check in the folder. You should see a package.json file, open it an examine.
   If you can't see the package.json file, click the folder refresh button. If it is still not there, check the folder you have run the command from before calling for help.
5. Install Webpack, its associated Command Line Interface (CLI) using and the development server using:

```
npm i webpack webpack-cli webpack-dev-server --save-dev
```

The **--save-dev** switch here installs these as development dependencies. You should find that your project folder now has a **node_modules** folder. This contains all of the packages that Webpack (and any future dependencies) need to be able to run.

6. Modify the **package.json** file by changing the scripts object to reflect:

```
"scripts": {
    "build": "webpack --mode production",
    "dev": "webpack --mode development"
}
```

7. Ensuring that you are in the Starter folder, create a sub folder called **src**.
8. Switch into the **src** sub folder and create an empty file called **index.js**.
9. Ensuring that you are in the Starter folder run the command:

```
npm run build
```

10. Examine the file, **main.js**, that has now been placed in the **dist** folder.

You should find that, without any configuration at all, webpack has produced a deployment-ready JavaScript file.

11. In the index.js file add a line:

```
console.log("It all works!");
```

12. Run the command:

```
npm run dev
```

13. Examine the **main.js** file now.

You should find that we now have a JavaScript file that has your line of code near the bottom enclosed in an **eval** statement. This is much easier to debug than the production version of the file!

Now we are able to run our JavaScript files through Webpack, the next stage is to let Webpack know that we would like it to use Babel to transpile the files.

14. Install Babel and its associated loaders and plugins using the following command:

```
npm i @babel/core babel-loader @babel/preset-env --save-dev
```

Babel needs to be configured. This is done in a file which has to be named **.babelrc**.

15. Create this file in the **Starter** folder and place the following code within it:

```
{
    "presets": [
        "@babel/preset-env"
    ]
}
```

The next stage is to configure Webpack through a configuration file called **webpack.config.js**. This file tells Webpack how to use Babel.

16. In the **Starter** folder, create a file called **webpack.config.js** and add the following code:

```
module.exports = {
    module: {
        rules: [
            {
                test: /\.js$/,
                exclude: /node_modules/,
                use: {
                    loader: "babel-loader"
                }
            }
        ]
    }
};
```

To check the transpilation, we will write some ES2015+ code and check the output file Webpack creates.

17. Open **index.js** from the **src** folder and add the following code:

```
const arr = [1, 2, 3];
const arrowFunction = () => console.log(...arr);
arrowFunction();
```

18. In the command line execute the following command:

```
npm run dev
```

19. Examine the **dist/main.js** file - find the transpiled code at the bottom. It should be on line 97 and uses the **eval()** function and a string to execute the code as ES5.

Running this command every time that you make a change to your JavaScript is not very efficient. Earlier, we installed webpack's development server and we will leverage this to launch a server that will monitor the JavaScript files and update our application on the fly.

20. Open package.json and add the following line to the scripts that we set up earlier for "dev" and "build":

```
"start": "webpack-dev-server --mode development --open"
```

Before we can see the result of this, we need to link the file produced by the development server to the **index.html** file

21. Open **index.html** and add a script tag to execute the JavaScript found in **./main.js**.

For Production release, this will be changed to dist/main.js - for now we need the virtual file created on the server to be used.

22. Finally, set the application running by executing the following command on the command line:

```
npm run start
```

23. Examine the page that is presented (by default on port 8081 - although this can be changed through configuration). On inspection of the console, you should see the messages we asked for in **index.js**.
24. To check that the development server updates, add another console log to the **index.js** file and save it.
25. You should observe that the command line shows a recompilation and that the output has changed on the console.

Webpack and Babel are commonly used in ReactJS and Angular applications amongst many others. There are a lot of different configurations that can be set, both for producing the bundled JavaScript files and to help the developer debugging. It is suggested that you look into these further from the Webpack documentation.

---

26. As a final check, use the **npm run build** command to rebuild the production file and change the link to the JavaScript file in **index.html** to **'./dist/main.js'**.

There you go, ready for deployment!