Tooling for building React applications

Goal

The goal of this lab is to transpile the *React* source code using *Webpack* and check for common errors using *ESLint*.

The Mission

In this lab you will be adding the tooling to a working React application. This tooling set will use *Webpack* to transpile the JSX source code to ECMAScript 5 understood by existing generation of browsers. In the second part of this lab you will add *ESLint* to the project to check the source code for common programming errors.

The starter application in this exercise contains a working movie editor. Even though the code is fully functional the JSX source code is not transpiled to JavaScript yet. You can start the application by running the START.BAT file located in the folder but the browser will just display a blank screen.

Your first task will be to use NPM to add Webpack and Babel and the required modules and configuration to transpile the JSX source code to ECMAScript 5. Once this step has been completed you should be able to run and use the application.

Your second task will be to detect and fix several small coding errors in the application. This will be done by adding the ESLint checker and several plugins to detect the errors.

Please note that the first time you start this project it will take some time, as a number of NPM packages will download. This requires an Internet connection where the NPM registry is not blocked by a firewall.

Note: The **FINISHED** folder contains a finished version of this lab as a reference.

Type it out by hand?
Typing it drills it into your brain much better than simply copying and pasting it. Because you're forming new neuron pathways that are going to help you in the future, help them out now!

Assignment 1

Start the application by running the START.BAT file located in the in the **BEGIN** folder and observe that the client side React application isn't loaded yet. Leave the server application running while you fix this.

The first task is to install Webpack, Babel and other required NPM modules

Use NPM to install *webpack*, *webpack-cli*, *@babel/core* and the required *babel-loader*. As these are development dependencies you can use the NPM option *–save-dev* to install them as devDependencies. Use the following command:

```
npm install --save-dev webpack webpack-cli babel-loader @babel/core
```

Once these are installed you can configure Webpack by creating a **WEBPACK.CONFIG.JS** in the root folder of the project. Use */app/app.jsx* as the entry point for the Webpack bundle. Specify */wwwroot/build* as the output path and *app-bundle.js* as the output file name.

You will need only a single module loader for the application. Add *babel-loader* as the module loader and use it for all JS and JSX files. Make sure to exclude all files from *node_modules* as these should not need to be transpiled using Babel.

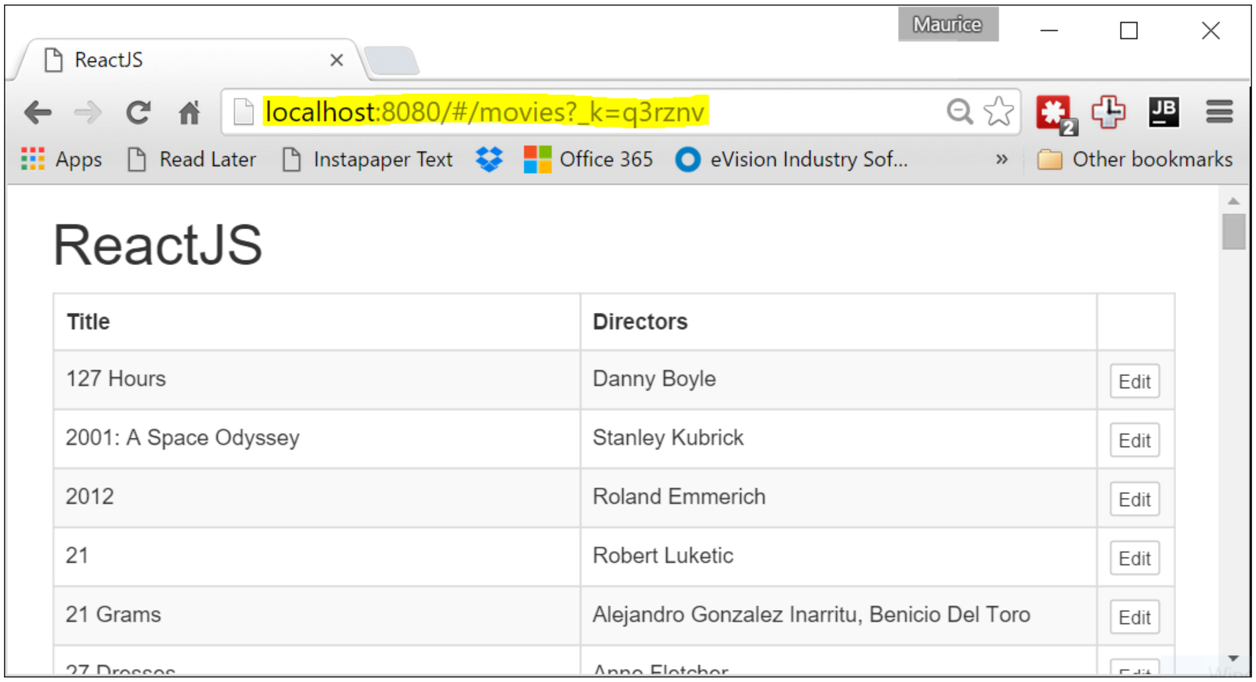
Next you will need to configure how Babel transpiles the source files. You can do this by adding a **.BABELRC** file. Make sure not to forget the leading period. Add a JSON object to this configuration file with a property *presets*. The value should be an array containing the following Babel presets: *@babel/preset-env* and *@babel/preset-react*. Make sure to install these presets as well as the required Babel package using the following NPM command.

```
npm install --save-dev @babel/preset-env @babel/preset-react
```

In order to transpile some of the class properties syntax you will need to add a *@babel/plugin-proposal-class-properties* to the plugins collection.

```
npm install --save-dev @babel/plugin-proposal-class-properties
```

With these in place you should be able to start Webpack and transpile the source code using *npm run webpack*. Now you can refresh the browser and you should see a functional application.



ReactJS

Title	Directors	
127 Hours	Danny Boyle	Edit
2001: A Space Odyssey	Stanley Kubrick	Edit
2012	Roland Emmerich	Edit
21	Robert Luketic	Edit
21 Grams	Alejandro Gonzalez Inarritu, Benicio Del Toro	Edit

Finally, make sure the application works and shows the list of movies.

Assignment 2

The second task is to improve on the code quality by adding ESLint and fixing the reported errors.

In this task you are going to install *ESLint* to check the source code for errors and bad practices every time a change is saved. Instead of defining our own rules you are going to use the popular *eslint-config-airbnb* as the set of rules to validate both the ECMAScript as well as the React JSX source code.

Start by installing ESLint and the required plugins using:

npm install --save-dev eslint babel-eslint eslint-plugin-react eslint-config-airbnb eslint-plugin-import eslint-plugin-jsx-a11y

When these NPM packages have installed you need to validate all the source code using ESLint. Implement the *eslint* script in the **PACKAGE.JSON**. The script should take all source files in the app folder and check them using eslint.

We still need to configure how ESLint should check the source code. In order to do that you need to add an **.ESLINTRC** file. Again, make sure not to forget the leading period. Add a JSON object to this configuration file with three properties: *parser*, *extends* and *plugins*.

1. The *parser* should be set to *babel-eslint*. This will automatically use the **.BABELRC** configuration file created before.
2. The *extends* property should be *airbnb* to use the AirBNB configuration.
3. The *plugins* property should be an array with *react* to lint React JSX code.

With this in place you should be able to restart *ESlint* and see a number of ESLint errors appear in the console window.

Fix all errors reported by ESLint. Note that quite a few can be fixed by running ESLint with the –fix option.