Setting up React

Navigate to your folder with the command line.

`npm init`

or

`npm init --yes` to skip having to hit enter multiple times.

You will want to create `dist` and `src` folders. `dist` will be where you store all of your web ready files that have been transpiled.

`mkdir dist`

`mkdir src`

`cd dist`

`touch index.html`

Now that you've done all that, edit `index.html` so that it contains the following:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
    </head>
    <body>
        <div id="app"></div>
        <script src="js/bundle.js" charset="utf-8"></scri
pt>
    </body>
```

```
</html>
```

Note that `bundle.js` will be your transpiled JavaScript.

Next create a new folder in the `src` folder named `js` and into that put a new file; `app.js`.

Whilst still in the `dist` folder, although it should make no difference, run the following at the command line:
`npm -i -S webpack`

At the root of your project create a new file named `webpack.config.js`. Add the following:

```
module.exports = {
    entry: './src/js/app.js',
    output: {
        filename: './dist/js/bundle.js'
    }
}
```

If you now go to the root of your project with the command line and type `webpack` it will do its thing and you'll end up with a copy of the file in your `src` folder and it'll come out as a copy in `dist` inside `bundle.js`. Actually that's a lie, it'll have some code in it, but it'll have a load of what currently appears to be crap because that's what it does. It takes modern JavaScript and turns it into good and current

JavaScript. With some crap apparently.

This on its own isn't very useful!

Another thing to note is that you had to run the `webpack` command manually. To change this, if you go to the command line and type `webpack --watch`

If you now add a line of code to your JavaScript in `src/app.js` such as

```
let name = 'Ash';
```

and hit save, then the changes will have been made and run through WebPack automatically.

However! Another way that this can be done is by opening your `package.json` file, add a new script named `start` like this

```
"start": "webpack --watch",
```

Now when you're in the root directory and you have your command line pointed at it, you can just do `npm start` and you'll have the same effect. It will. You won't, that's silly.

Once you have this setup, it's possible to do imports from one JavaScript file to another. For example, if in your `src` folder you create a file named `test.js` right next to your `app.js` file and write

```
export default {
    name: 'Ash!'
}
```

Then whilst you're in your own `app.js` you can do:

```
import obj from './test.js'
console.log(obj.name);
```

and it'll work! To test, get your `npm start` going and open your `index.html` file in your `dist` folder. You should see your log. This is pretty cool!

Now we need to be able to convert our ECMAScript 2015 and convert it to ECMAScript 5 so it's something which most browsers can currently handle. To do this! At your root enter the following command:

`npm install --save-dev babel-loader babel-core babel-preset-env webpack`

The next thing to do is to make use of these things. Inside `webpack.config.js` add the following:

```
module.exports = {
    entry: './src/js/app.js',
```

```
    output: {
        filename: './dist/js/bundle.js'
    },
    module: {
        rules: [
            {
                test: /\.js$/,
                exclude: /(node_modules|bower_components)
/,
                use: {
                    loader: 'babel-loader',
                    options: {
                        presets: ['env']
                    }
                }
            }
        ]
    }
}
```

So! If you make sure your `npm start` command has been run and you go to your `app.js` file and add a line like `let name = 'Ash'`, you should be able to locate it in your `bundle.js` file as a `var` declaration. Hooray! We're transpiling!

Now, according to the React website, Babel will also need some extra installations. This can be done in the following way. Run the following

command:

`npm i -D babel-preset-es2015 babel-preset-react`

Now in your `package.json` file, you should add this entry:

```
    "babel": {
        "presets": [
            "babel-preset-es2015",
            "babel-preset-react"
        ]
    }
}
```

This should be right underneath the dev dependencies.

So now! When Babel transforms your JS, it will do it in this way.

Installing React

Run this command:
`npm i -S react react-dom`

Now change `app.js` to look like, in fact don't just make it look like, make it identical to the following:

```
import React from 'react'
```

```
import ReactDOM from 'react-dom'

const component = {
    <h1>Hello, world!</h1>
}

const target = document.getElementById('app')
ReactDOM.render(component, target)
```

You should now be able to do `npm start`, navigate to your page and you should see your hello world header component!

The Cheating Bastard Method

You could have run the following:

```
npm install -g create-react-app

create-react-app my-app
cd my-app/
npm start
```

You can then navigate to localhost:3000 to see your app! Then, when you're ready to deploy to production, just create a minified bundle with `npm run build`.