Appendix A: Installation

Section A.1: Simple setup

Setting up the folders

This example assumes code to be in `src/` and the output to be put into `out/`. As such the folder structure should look something like

```
example/
|-- src/
|   |-- index.js
|   `-- ...
|-- out/
`-- package.json
```

Setting up the packages

Assuming a setup npm environment, we first need to setup babel in order to transpile the React code into es5 compliant code.

```
$npm install --save-dev babel-core babel-loader babel-preset-es2015 babel-preset-react
```

The above command will instruct npm to install the core babel libraries as well as the loader module for use with webpack. We also install the es6 and react presets for babel to understand JSX and es6 module code. (More information about the presets can be found here Babel presets)

```
$npm i -D webpack
```

This command will install webpack as a development dependency. (**i** is the shorthand for install and **-D** the shorthand for --save-dev)

You might also want to install any additional webpack packages (such as additional loaders or the webpack-dev-server extension)

Lastly we will need the actual react code

```
$npm i -D react react-dom
```

Setting up webpack

With the dependencies setup we will need a webpack.config.js file to tell webpack what to do

simple webpack.config.js:

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

module.exports = {
  entry: './src/index.js',
  output: {
    path: path.resolve(__dirname, 'out'),
    filename: 'bundle.js'
  },
  module: {
    loaders: [
      {
        test: /\.js$/,
```

```
        exclude: /(node_modules)/,
        loader: 'babel-loader',
        query: {
          presets: ['es2015', 'react']
        }
      }
    ]
  }
};
```

This file tells webpack to start with the index.js file (assumed to be in src/) and convert it into a single bundle.js file in the `out` directory.

The `module` block tells webpack to test all files encountered against the regular expression and if they match, will invoke the specified loader. (`babel-loader` in this case) Furthermore, the `exclude` regex tells webpack to ignore this special loader for all modules in the `node_modules` folder, this helps speed up the transpilation process. Lastly, the `query` option tells webpack what parameters to pass to babel and is used to pass along the presets we installed earlier.

Testing the setup

All that is left now is to create the `src/index.js` file and try packing the application

src/index.js:

```
'use strict'

import React from 'react'
import { render } from 'react-dom'

const App = () => {
  return <h1>Hello world!</h1>
}

render(
  <App />,
  document. getElementById('app')
)
```

This file would normally render a simple `<h1>Hello world!</h1>` Header into the html tag with the id 'app', but for now it should be enough to transpile the code once.

`$./node_modules/.bin/webpack .` Will execute the locally installed version of webpack (use `$webpack` if you installed webpack globally with -g)

This should create the file `out/bundle.js` with the transpiled code inside and concludes the example.

Section A.2: Using webpack-dev-server

Setup

After setting up a simple project to use webpack, babel and react issuing `$npm i -g webpack-dev-server` will install the development http server for quicker development.

Modifying webpack.config.js

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

```
module.exports = {
  entry: './src/index.js',
  output: {
    path: path.resolve(__dirname, 'out'),
    publicPath: '/public/',
    filename: 'bundle.js'
  },
  module: {
    loaders: [
      {
        test: /\.js$/,
        exclude: /(node_modules)/,
        loader: 'babel',
        query: {
          presets: ['es2015', 'react']
        }
      }
    ]
  },
  devServer: {
    contentBase: path.resolve(__dirname, 'public'),
    hot: true
  }
};
```

The modifications are in

- `output.publicPath` which sets up a path to have our bundle be served from (see Webpack configuration files for more info)
- `devServer`
    - `contentBase` the base path to serve static files from (for example `index.html`)
    - `hot` sets the webpack-dev-server to hot reload when changes get made to files on disk

And finally we just need a simple index.html to test our app in.

index.html:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>React Sandbox</title>
  </head>
  <body>

    <div id="app" />

    <script src="public/bundle.js"></script>
  </body>
</html>
```

With this setup running `$webpack-dev-server` should start a local http server on port 8080 and upon connecting should render a page containing a `<h1>Hello world!</h1>`.

Appendix B: React Tools

Section B.1: Links

Places to find React components and libraries;

- Catalog of React Components
- JS.coach