



# Setting up a ReactJS Project

Developing Applications using ReactJS





## Objectives

- To understand the anatomy of a ReactJS project
- To have a basic understanding of the modules and files needed to set up a ReactJS project

## React Project Fundamentals



- **Main project folder**
- **Folder for all modules to be used in the application**
- **The main html page**
- **The entry point for the JavaScript**
- **File for meta data for node**
- **Configuration details for webpack**



## react and react-dom packages

- **Main react packages split into two:**
  - react and react-dom
  - **react** package:
    - Contains `React.createElement`, `.createClass`, `.Component`, `.Children` methods and other helpers related to elements and component classes
  - **react-dom** package:
    - Contains `ReactDOM.render`, `.unmountComponentAtNode` and `.findDOMNode` methods, along with server-side rendering support
- **Other packages for other add-ons can also be used**
- **These are placed in the `node_modules` folder**

## Babel packages

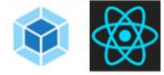
*BABEL*

- **Essentially an ES2015+ to ES5 compiler (aka a transpiler)**
  - Allows modern script to still be run on browsers not supporting ES2015
  - Polyfill plugin also available for older browsers
- **Comes packaged as a node module**
  - Plugins available for webpack, grunt, gulp, etc
- **For ReactJS development, the following Babel packages are used:**
  - babel-core – the core compiler
  - babel-loader – Webpack plugin for Babel
  - babel-preset-react and babel-preset-env – to transpile ES2015+ and React code
- **These files are placed in the node\_modules folder**

14

Live demonstration available at <http://babeljs.io/repl/>

Babel logo from: <https://github.com/babel/babel>



## Webpack

- **Module bundler**
  - Takes modules with dependencies and generates static assets representing those modules
- **Two packages required when setting up a ReactJS project with webpack:**
  - webpack – the core bundler
  - webpack-dev-server – a small server which serves a webpack bundle and essentially allows auto-refreshing so as changes are made, the browser updates without refresh
- **These files are placed in the node\_modules folder**
- **browserify is an alternative that can be used instead of webpack**

Webpack logo from: <https://github.com/webpack/webpack>



## Other files

- **index.html**
  - Essentially contains the HTML template that React will help populate.
  - Generally has at least one empty element with an id to address it by
  - Contains a script reference to a file index.js
    - Rarely stored on disk
    - Created and held by the server for the lifetime of the app
      - Can be created on disk by running webpack on the command line or terminal
- **main.js**
  - The entry point for JavaScript execution
  - Contains the information to render to the browser using React
- **package.json**
  - Contains information for node to be able to run the application
- **webpack.config.js**
  - Contains information needed by webpack to render the HTML correctly
  - Also contains some server information



## main.js

- **Simple example of a *main.js* file:**

```
import React from 'react';
import ReactDOM from 'react-dom';

ReactDOM.render(
  <h1>Hello, World!</h1>,
  document.querySelector('#app')
);
```

- **ES2015 is used to import classes that can be found in the React library**
- **ReactDOM.render () contains a mixture of HTML and JavaScript**
  - More on that later!





## package.json

- **Simple example of a *package.json* file:**

```
{
  "name": "starter",
  "version": "1.0.0",
  "description": "My First React App",
  "main": "index.js",
  "scripts": {
    "start": "webpack-dev-server --hot",
    "build": "webpack -p"
  },
  "author": "",
  "license": "ISC"
}
```

- **"start": "webpack-dev-server --hot" tells npm to use the installed server when the start command is used and auto-refresh the page if changes are made**
- **"build": "webpack -p" produces the bundled JS file whose name is specified in the webpack.config.js file**

18

When dependencies are present, running npm install in the same folder that contains the package.json file will install the dependencies listed. This means that the node\_modules folder is not absolutely necessary when pushing to a git repository, as any developer cloning the repository will get a copy of the package.json file and will be able to install the necessary dependencies.

In general, npm install will retrieve the most recent version of a package that is compatible with that listed in the package.json file, but a health warning, compatibility is not always guaranteed!

In the projects you will be working on, you may see an npm\_shrinkwrap.json. This is a file included to ensure that the projects install the versions of the packages that were used at the time of writing the course.



## webpack.config.js

- Simple example of a *webpack.config.js* file:

```
module.exports = {
  entry: ['babel-polyfill', './main.js'],
  output: {
    path: __dirname,
    filename: 'index.js'
  },
  resolve: {
    extensions: ['.js', '.jsx'],
  },
  devServer: {
    inline: true,
    port: 8080
  },
  devtools: 'source-map',
  module: {
    loaders: [{
      test: /\.jsx?$/,
      exclude: /node_modules/,
      loader: 'babel-loader',
      query: {
        presets: ['env', 'react']
      }
    }]
  }
}
```

## Setting up a Project



- **Done by a mixture of command line/terminal execution and code editing**
- **Each new project should be set up locally**
  - Avoids clashes with global installations on individual computers
- **After a single project has been set up, can be used as a template for all others**



## Objectives

- To understand the anatomy of a ReactJS project
- To have a basic understanding of the modules and files needed to set up a ReactJS project

## Exercise Time



- **Complete EG02 – Setting Up a ReactJS Project**