1. **Setup React.js with Npm, Babel 6 and Webpack in under 1 hour**

Ok so this is not a article where you will understand everything about React.js, Npm, Babel and Webpack.



This is a step to step on how to get your first React project up in under 1 hour ;). I use Windows and found good articles for Linux , but no one for windows so here you go.

Create a new folder ‘first-react-project’ and initialize it with npm.

npm init

Then install webpack

npm i webpack -S

Then create file

**Linux**  
touch webpack.config.js

**Windows**  
Right click in folder and create new file, webpack.config.js

Update “webpack.config.js” with

var webpack = require('webpack');  
var path = require('path');  
  
var BUILD\_DIR = path.resolve(\_\_dirname, 'src/client/public');  
var APP\_DIR = path.resolve(\_\_dirname, 'src/client/app');  
  
var config = {  
 entry: APP\_DIR + '/index.jsx',  
 output: {  
 path: BUILD\_DIR,  
 filename: 'bundle.js'  
 }  
};  
  
module.exports = config;

Create index.jsx file in the “src/client/app” and add console.log('Hello World!'); in it.

In the terminal run the following command

**Linux**  
./node\_modules/.bin/webpack -d

**Windows**  
node\_modules\.bin\webpack -d

The above command runs the webpack in the development mode and generates the *bundle.js* file in *src/client/public* directory.

Now create an *index.html* file in the *src/client* directory and modify it to use this *bundle.js* file

<html>  
 <head>  
 <meta charset="utf-8">  
 <title>React.js using NPM, Babel6 and Webpack</title>  
 </head>  
 <body>  
 <div id="app" />  
 <script src="public/bundle.js" type="text/javascript"></script>  
 </body>  
</html>

If you open the browser, you can see the *Hello World!* in the **console log**.

**Setting Up Babel-Loader**

By using JSX and ES6 we can be more productive while working with React, so we need to install the following npm packages.

npm i babel-core babel-loader babel-preset-es2015 babel-preset-react -S

Now we need to create .babelrc file

**Linux**  
touch .babelrc

**Windows**Create a b.babelrc file then in CMD

ren b.babelrc .babelrc (Just rename the file manually)

Open file and add

{  
 "presets" : ["es2015", "react"]  
}

Now we need to tell Webpack to use the babel-loader while bundling the files, open *webpack.config.js* file and update it as below

var webpack = require('webpack');

var path = require('path');

var BUILD\_DIR = path.resolve(\_\_dirname, 'src/client/public');

var APP\_DIR = path.resolve(\_\_dirname, 'src/client/app');

var config = {

entry: APP\_DIR + '/index.jsx',

output: {

path: BUILD\_DIR,

filename: 'bundle.js'

},

module: {

rules: [

{

test: /\.jsx?/,

use: [

{ loader: 'babel-loader' }

]

}

]

}

};

module.exports = config;

**Lets get some text out**

Use npm to install react and react-dom

npm i react react-dom –S

Replace the existing console.log statement in the *index.jsx* with the following content

import React from 'react';  
import {render} from 'react-dom';  
  
class App extends React.Component {  
 render () {  
 return <p> Hello React project</p>;  
 }  
}  
  
render(<App/>, document.getElementById('app'));

Run the following command to update the bundle file with the new changes

**Linux**  
./node\_modules/.bin/webpack -d

**Windows**  
node\_modules\.bin\webpack -d

Open the *index.html* in the browser, you can see *Hello React, be proud ;)*

**Add component…**

Create a new file *AwesomeComponent.jsx* and update it as below

import React from 'react';  
  
class AwesomeComponent extends React.Component {  
  
 constructor(props) {  
 super(props);  
 this.state = {likesCount : 0};  
 this.onLike = this.onLike.bind(this);  
 }  
  
 onLike () {  
 let newLikesCount = this.state.likesCount + 1;  
 this.setState({likesCount: newLikesCount});  
 }  
  
 render() {  
 return (  
 <div>  
 Likes : <span>{this.state.likesCount}</span>  
 <div><button onClick={this.onLike}>Like Me</button></div>  
 </div>  
 );  
 }  
  
}  
  
export default AwesomeComponent;

Then include it in the *index.jsx* file

// ...  
import AwesomeComponent from './AwesomeComponent.jsx';  
// ...  
class App extends React.Component {  
 render () {  
 return (  
 <div>  
 <p> Hello React Project</p>  
 <AwesomeComponent />  
 </div>  
 );  
 }  
}  
  
// ...

If your Webpack is already running in watch mode or you have updated, then refresh the browser to see the AwesomeComponent in action.

otherwise run

**Linux**  
./node\_modules/.bin/webpack -d --watch

**Windows**  
node\_modules\.bin\webpack -d –watch

Setting up Git to exclude node\_modules to be pushed to github  
  
**git rm -r --cached --ignore-unmatch node\_modules**

Setting up the app to be run via Webpack devserver.

1)

**npm install webpack-dev-server --save-dev**  
  
2)  
Add these to start array in package.json  
  
**"start:dev": "webpack-dev-server --mode development --watch-content-base --open",**

**3)**

Add these to webpackconfig just above watch  
  
These are the devserver settings ->

**devServer: {**

**contentBase: path.resolve(\_\_dirname, 'src/client'),**

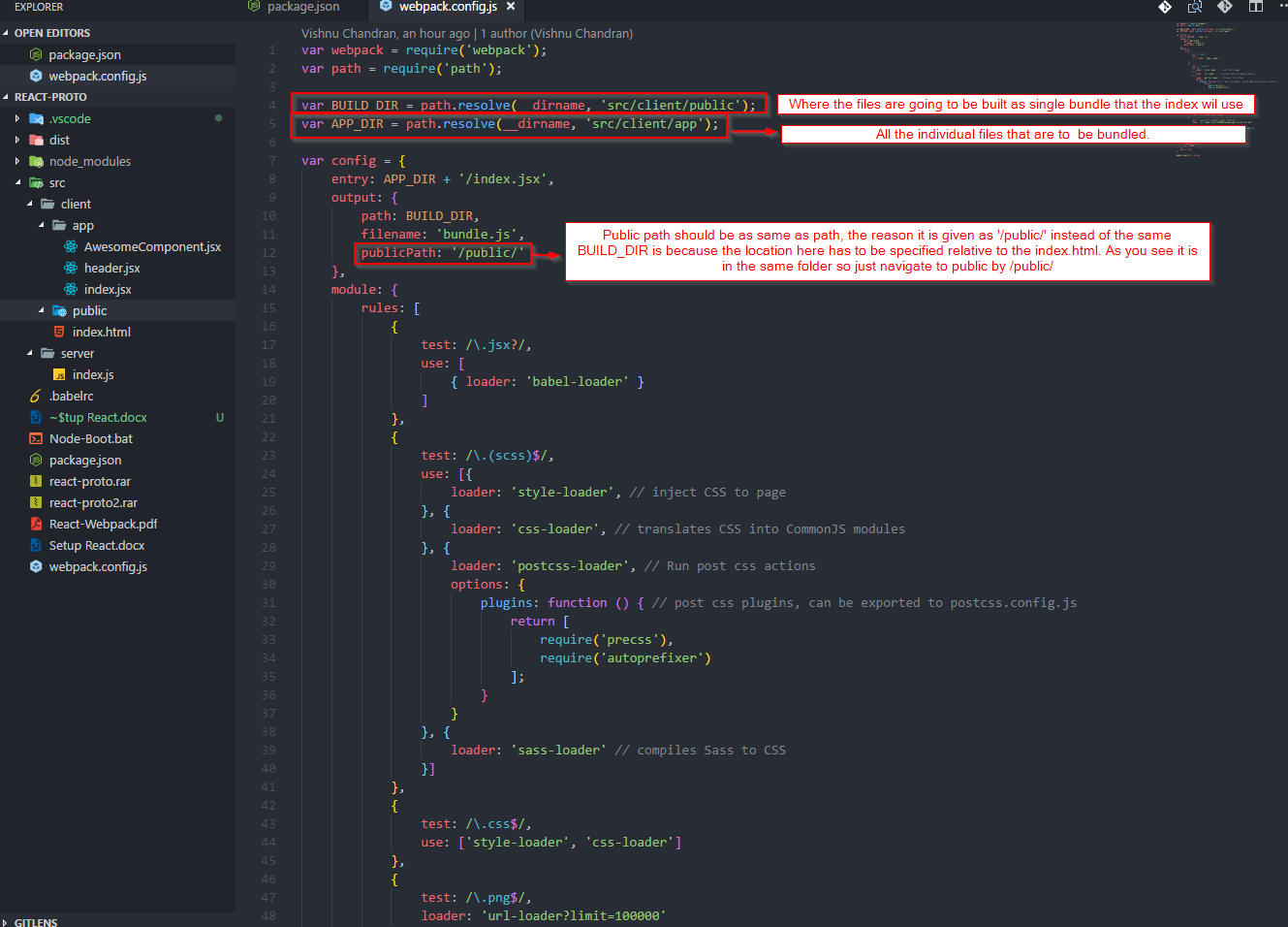
**watchContentBase: true,**

**compress: true,**

**port: 9000**

**}**

**Also add publicPath output as shown below.**



**4) Now delete all files in the client/public folder (if at all they were created by previous webpack build)   
Webpack dev server makes those build.js files into memory , not on disk.  
  
5) Run via console - npm run start:dev**

**And open** [**http://localhost:9000**](http://localhost:9000) **in browser.**

**Make changes to index.jsx , like paragraph renaming etc , the console should recompile fast and page will be reloaded with changes.**

Stateful vs Stateless components.

* **Stateless Component** — Only *props*, no *state.* There's not much going on besides the render() function and all their logic revolves around the *props* they receive. This makes them very easy to follow (and test for that matter). We sometimes call these *dumb-as-f\*ck Components* (which [turns out](http://www.urbandictionary.com/define.php?term=dumb%20as%20fuck) to be the only way to misuse the F-word in the English language).
* **Stateful Component** — Both *props* and *state.* We also call these *state managers*. They are in charge of client-server communication (XHR, web sockets, etc.), processing data and responding to user events. These sort of logistics should be encapsulated in a moderate number of *Stateful Components*, while all visualization and formatting logic should move downstream into as many *Stateless Components* as possible.