REACT JS: Steps to Create First React “Hello World” application

Ref: <https://www.kirupa.com/react/setting_up_react_environment.htm>

You can download the sample code from my github repo [here](https://github.com/sinhajyoti/Tutorials-Misc/tree/master/REACT%20JS-HelloWorld%20code)

# Step1: Create the folder structure

mkdir *helloworld /// folder to contain the app code*

*cd helloworld /// make this a working directory*

*mkdir dev ///create Dev code that would have jsx file*

*mkdir output /// create folder where output javascript will be generated*

# Step 2: Get the UI ready

Inside *helloworld* folder, create *index.html* as presentation UI. File should look like following:

<!DOCTYPE html>

<html>

<head>

<title>Hello World</title>

</head>

<body>

<div id="container"></div>

<script src="output/runtimeCode.js"></script>

</body>

</html>

In the above code, *runtimeCode.js* is being referenced which will be generated post-compilation. No need to worry about that for now.

# Step 3: NodeJS initialization

Make sure nodeJS is installed, to check that go to command prompt and check version:

Node –v

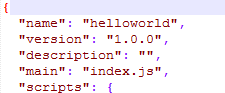
Make sure on command prompt, you are inside *Helloworld* directory. Initlaize the *npm*

npm init

This will take a while and prompt for series of questions. For Name, just enter *helloworld*. Make sure it is in smaller cap. For rest of the questions, just press *<enter>*, for now.

At the end of it, a *package.json* file shall be generated inside root “*Helloworld*” folder.

*Package.json* should look something like following:



# Step 4: Install React Dependencies

On command prompt, make sure you are in “*Helloworld*” directory

npm install react react-dom –save

It should create another folder named node\_module inside your helloworld directory.

# Step 5: Create React component

Inside folder *Dev*, create a new file and name it as *index.jsx*. Now, edit the index.jsx and put following content therein:

import React from "react";

import ReactDOM from "react-dom";

var HelloWorld = React.createClass({

render: function() {

return (

<p>Welcome {this.props.name}!</p>

);

}

});

ReactDOM.render(

<div>

<HelloWorld name="Jyoti"/>

<HelloWorld name="Sinha"/>

</div>,

document.querySelector("#container")

)*;*

First, it imports node.js packages for *React* and R*eactDom.*

Then, It’s quite simple component named *Helloworld*, which on render, will use attribute “name” of tag <*HelloWorld*> and prefix that with text “Welcome ”.

# Step 6: *Webpack* configuration

The conversion of *jsx* to browser understandable javascript is facilitated by *webpack* and *Babel*. First, let’s configure Webpack

npm install webpack –save

Let’s install *webpack dev server*, so that, app be run using browser

npm install webpack-dev-server –save

Both these commands will download plenty of dependencies in *node\_modules* folder. Also, there would be few config additions in package.json. That’s expected.

Now, in order to make *webpack* aware of *index.jsx* file and output file, let’s create a file *webpack.config.js* in *Helloworld* folder. Open the *webpack.config.js* in text editor and place following configrations:

var webpack = require("webpack");

var path = require("path");

var DEV = path.resolve(\_\_dirname, "dev");

var OUTPUT = path.resolve(\_\_dirname, "output");

var config = {

entry: DEV + "/index.jsx",

output: {

path: OUTPUT,

filename: "runtimeCode.js"

},

devServer: {

inline: true,

port: 8085

}

};

module.exports = config;

In the above config, *\_\_dirname* gives current directory path and “dev” and “output” is respective folder to be used to map *index.jsx* file to *runtimeCode.js*. Again, don’t worry about not having *runtimeCode.js* file yet. That will be generated when we compile the code.

*devServer* tells us about what port(*8085* in above config) the *webpack* server is going to listen at.

Now, let’s tell *npm* to start the *webpack dev server* on start. Open the *package.json* file in text editor and inside the scripts object make the following changes

**Old setting**:

“scripts”:{

"test" "echo \"Error: no test specified\" && exit 1"

}

**New setting:**

"scripts": {

"start": "webpack-dev-server --hot"

}

# Step 7: Babel configuration

Make sure, on command prompt, current directory is “HelloWorld”. Let’s install Babel using following command:

npm install babel-core babel-loader babel-preset-es2015 babel-preset-react --save

Again, it will download dependencies and makes changes to *package.json* file.

Let’s complete the Babel configuration now. Open the *Package.json* file in your text editor and add following highlighted text

{

"name": "helloworld",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"start": "webpack-dev-server --hot"

},

"author": "",

"license": "ISC",

"dependencies": {

"babel-core": "^6.24.0",

"babel-loader": "^6.4.1",

"babel-preset-es2015": "^6.24.0",

"babel-preset-react": "^6.23.0",

"react": "^15.4.2",

"react-dom": "^15.4.2",

"webpack": "^2.2.1",

"webpack-dev-server": "^2.4.2"

},

"babel": {

"presets": [

"es2015",

"react"

]

}

}

Also, open the *webpack.config.js* and ensure it has babel-loader configured as module as highlighted in following code. This directs *webpack* to pass the *index.jsx* file defined in entry property through babel-loader for transformation into javascript for browser engine.

var webpack = require("webpack");

var path = require("path");

var DEV = path.resolve(\_\_dirname, "dev");

var OUTPUT = path.resolve(\_\_dirname, "output");

var config = {

entry: DEV + "/index.jsx",

output: {

path: OUTPUT,

filename: "runtimeCode.js"

},

devServer: {

inline: true,

port: 8085

},

module: {

loaders: [{

include: DEV,

loader: "babel-loader",

}]

}

};

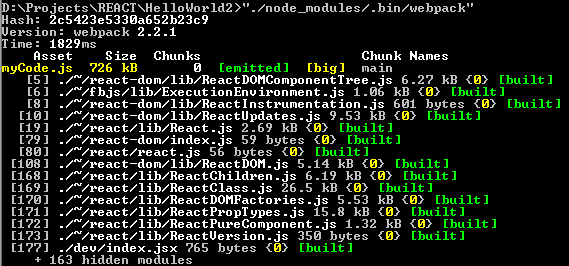
module.exports = config;

# Step -8: Build and Run the *Helloworld*

Make sure, on command prompt, your current directory is “*Helloworld*”.

“./node\_modules/.bin/webpack”

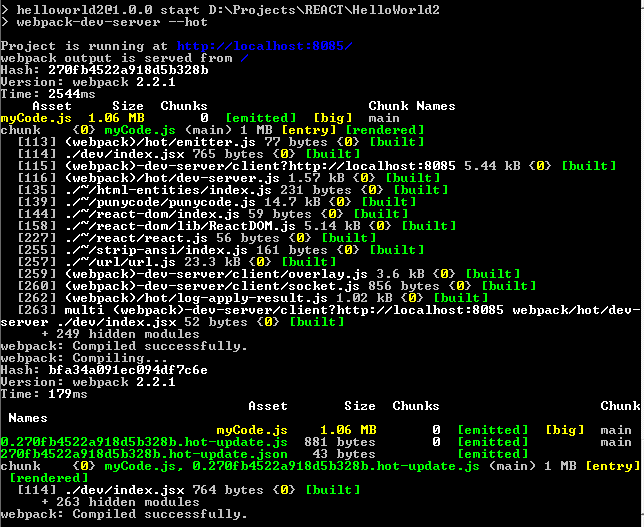
That should compile the code and present the output similar to following screenshot:



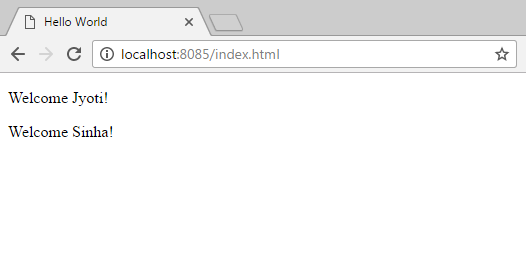
Let’s run the *webpack dev server*

npm start

It will also perform compilation and start the server which listens at port configured in *package.json* in previous steps.



Let’s go to favorite browser try *helloworld:*



Here you go to your first successful REACT program!

Cheers!!!