# Set up React from Zero

Tutorial about how to create from zero a simple development setup for React applications using Webpack and Babel. ***You do not have to submit anything for this exercise.***



## Create React setup

### Create new project

Create new project and initialize a **package.json** file.

npm init

### Install Webpack

**Webpack** is a module **bundler** for modern JavaScript applications. When **webpack** processes your application, it **recursively** **builds** a dependency graph that includes every module your application needs, then packages all of those modules **into** a small number of **bundles** - often only one - to be loaded by the browser.

**Webpack** installation

npm i webpack -S

In your project create a 'webpack.config.js' file, and place the following code inside.

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;

In the cofig file, you are configuring the src folder (from which you are taking your resources) and **dist**, folder where you will save, your processed files.

### Test Your Setup

In your project folder create **'src/client/app'** folder, and inside of it create index.jsx file, inside of it write the following line of code.

console.log('You have successfully setup webpack')

In **'src/client'** directory, create index.html file with the following code.

<html>  
 <head>  
 <meta charset="utf-8">  
 <title>React.js using NPM, Babel6 and Webpack</title>  
 </head>  
 <body>  
 <h1>Press F12 and see whats going on in the console</h1>

<div id="app" />  
 <script src="public/bundle.js" type="text/javascript"></script>  
 </body>  
</html>

Run the following command in your project directory, and open the index.html file

node\_modules\.bin\webpack -d

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

### Install and Config Babel

Babel is essentially an ESNext to ES”current” transpiler. In short - it allows you to use language features “from the future”. It does so by transpiling your code to the currently supported version of JavaScript.

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

Create a '.babelrc' file which will store your transpiration configuration, and should hold the following code.

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

Now we should update the **Webpack** configuration in such way, that it will **transpile** our code when creating our 'bundle.js' file. Add the following code in your **Webpack** **config** file.

// Existing Code ....  
var config = {  
 // Existing Code ....  
 module : {  
 loaders : [  
 {  
 test : /\.jsx?/,  
 include : APP\_DIR,  
 loader : 'babel-loader'  
 }  
 ]  
 }  
}

### Install React and React-Dom

Install React and React-dom, so that we can use the library.

npm i react react-dom -S

Switch the index.jsx with the following code.

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'));

### 1.6 Enable Webpack Watch

Run the following command

node\_modules\.bin\webpack -d --watch

by doing so **Webpack**, will start to watch for any changes in your code and will re-run itself, whenever it is needed.

Congratulations you now have a working project

## Extend Your Project

As an introduction to React, convert the provided HTML and CSS files to a React application. **There is no need to use components yet**, just play around with the import/export functionality that JSX provides. Try to find any parts that of the markup that are similar and use functions returning HTML to reduce code repetition.

For instance, the majority of the HTML might be placed inside a variable named app, which you pass to ReactDOM.render(), while the contact-card of each user should be a function makeContact(), that receives a user object and returns HTML based on the data. Import the list of contacts from the provided JSON file and iterate over the array, passing each object to the makeContact() function. Store the results in a variable and place that inside the HTML of app.