React setup with Webpack

# Folder Setup

Firstly just create your project folder and then a *public* and *src* folder inside it. *public* folder will be used to serve the application and this folder will be everything to publish your application.

All the javascript files will be in the *src* folder and this folder will be bundled into a single javascript file and will be placed automatically in the public folder.

Next just create an *index.js* file in the *src* folder and an *index.html* file in the *public* folder.

# Initializing app

Set up your package.json file by running *npm init*

## Installing the necessary packages

### Installing React

Run *npm install react react-dom*

Now add the following code to your index.html file

<!DOCTYPE *html*>

<html *lang*="en">

<head>

<meta *charset*="utf-8" />

<meta *name*="viewport" *content*="width=device-width, initial-scale=1" />

<title>React App</title>

</head>

<body>

<noscript>You need to enable JavaScript to run this app.</noscript>

<div *id*="root"></div>

</body>

</html>

Create an App.js component in the src directory

import React from 'react'

const App = () => {

return (

<div>React App</div>

)

}

export default App

And and this to your index.js file

import React from "react";

import ReactDOM from "react-dom/client";

import App from "./app";

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(

<*React.StrictMode*>

<*App* />

</*React.StrictMode*>

);

### Installing Webpack

* Run the following command to install all the required packages :

*npm i --save-dev webpack webpack-cli webpack-dev-server*

By running this command you’ve installed 3 packages   
  
 *webpack* : used to bundle all js code and build the application

*webpack-dev-server* : used to server the application on your local server for development  
*webpack-cli* : provides a set of commands to set up the webpack project.

# Basic config

Now you need to create a webpack.config.js file in the root folder.  
After that you can add the following script to your package.json

"scripts": {

"start": "webpack serve --config ./webpack.config.js --mode development",

}

Now we need to specify the index.js file as the entry point. Any file that the index.js file imports will be bundled too. Also we will add an output location for the bundled file. Here we tell webpack to create the final bundled file (bundle.js) in the *public* folder in the root of the project.

const path = require('path');

*module*.*exports* = {

entry: './src/index.js',

output: {

path: path.join(\_\_dirname, 'public'),

filename: 'bundle.js',

}

Once the bundled javascript file is created we need to tell webpack to inject it as a script tag to the HTML file. To do that we first need to install a webpack plugin that will help us do it.

*npm install --save-dev html-webpack-plugin*

Use the following code to add this plugin

const path = require('path');

const HtmlWebpackPlugin = require('html-webpack-plugin');

*module*.*exports* = {

entry: './src/index.js',

output: {

path: path.join(\_\_dirname, 'public'),

filename: 'bundle.js',

},

plugins: [

**new** HtmlWebpackPlugin({

template: './public/index.html',

}),

],

}

# Installing Babel

Now we will add Babel to transpile your ES6 Javascript code to vanilla javascript. Run the following command for installing the required packages:

*npm i --save-dev @babel/core @babel/preset-env @babel/preset-react babel-loader*

*@babel/preset-env* is a smart preset that allows you to use the latest JavaScript without needing to micromanage which syntax transforms (and optionally, browser polyfills) are needed by your target environment(s)

*@babel/preset-react* will help in transpiling react code

*babel-loader* Loaders tell webpack how to interpret and translate files. The transformation happens on a per-file basis before adding to the dependency graph.

Now you need to update the webpack config file for using Babel with the presets

const path = require('path');

const HtmlWebpackPlugin = require('html-webpack-plugin');

*module*.*exports* = {

entry: './src/index.js',

output: {

path: path.join(\_\_dirname, 'public'),

filename: 'bundle.js',

},

module: {

rules: [

{

test: /\.?js|jsx$/,

exclude: /node\_modules/,

use: {

loader: "babel-loader",

options: {

presets: ['@babel/preset-env', '@babel/preset-react']

}

}

},

]

},

plugins: [

**new** HtmlWebpackPlugin({

template: './public/index.html',

}),

],

}

After all of these steps are done, create a .babelrc file with the following code

{

"presets": ["@babel/preset-env"]

}

# Adding Typescript support

We are now ready to run our React application using webpack configs but this project will only support Javascript/Jsx code. To configure typescript we need to go over a few more steps.

Firstly we need to get the required packages, we can get those by running the following command

*npm install --save-dev ts-loader typescript @types/react @types/react-dom*

All we did was add a loader and type support for react by running this command.

In order to work with typescript, we need to create a tsconfig.json file. In our case, we can create the file manually as the configuration we need is very simple.

So just create the *tsconfig.json* file in your root folder and add the following code:

{

"exclude": [

"node\_modules",

"node\_modules/\*",

"./packages/\*\*/node\_modules",

],

"compilerOptions": {

"outDir": "public",

"sourceMap": true,

"jsx": "react",

"target": "ES6",

"allowJs": true,

"esModuleInterop": true,

}

There are just the basic items we need to get things rolling, you can find additional compiler options according to your needs [here](https://www.typescriptlang.org/tsconfig).

Now let’s modify our webpack.config.js file to add the typescript loader and we’re done.

const path = require('path');

const HtmlWebpackPlugin = require('html-webpack-plugin');

*module*.*exports* = {

entry: './src/index.js',

output: {

path: path.join(\_\_dirname, 'public'),

filename: 'bundle.js',

},

module: {

rules: [

{

test: /\.?js|jsx$/,

exclude: /node\_modules/,

use: {

loader: "babel-loader",

options: {

presets: ['@babel/preset-env', '@babel/preset-react']

}

}

},

{ test: /\.tsx?$/, loader: 'ts-loader' },

]

},

resolve: {

extensions: ['\*', '.js', '.jsx', '.ts', '.tsx'],

},

plugins: [

**new** HtmlWebpackPlugin({

template: './public/index.html',

}),

],

}

# Build and run

Now that we have already added all the necessary configuration for Webpack and added the *start* script to the package.json file we can run npm start to see the server running on localhost. Additionally we can add a *build* script to build the code for deployement.

"scripts": {

"start": "webpack serve --config ./webpack.config.js --mode development",

"build": "webpack --mode production"

}

**NOTE: If you get any ‘module not found’ or ‘cannot resolve’ errors after starting the app then try running the installation commands for the respective packages again.**