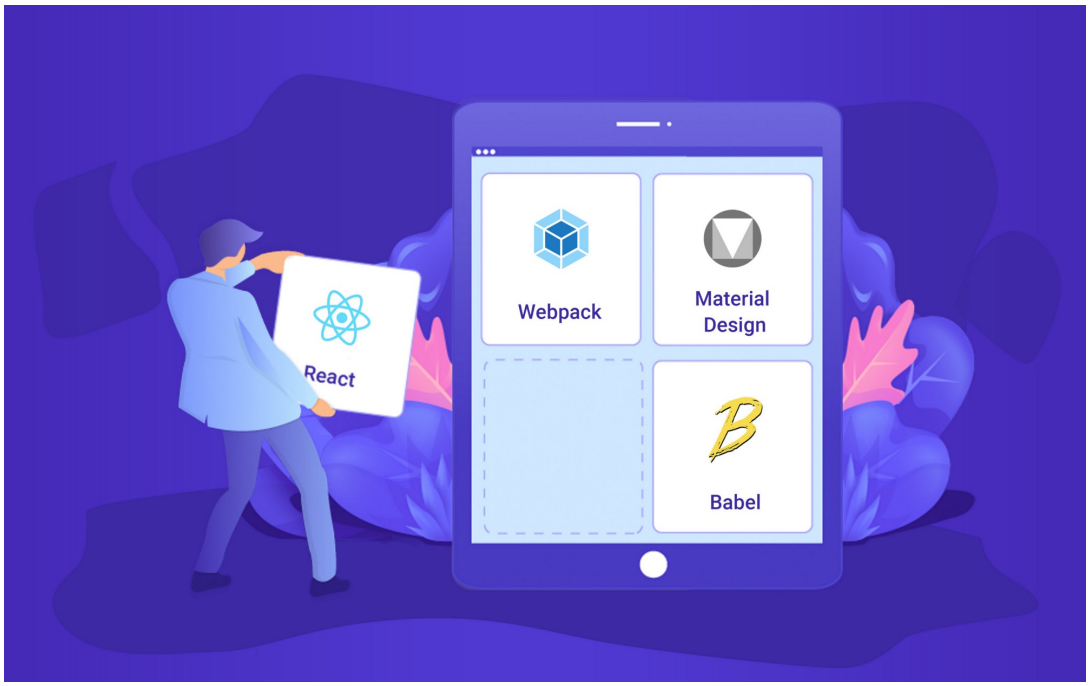
How to use ReactJS with Webpack 4, Babel 7, and Material Design

Nazare Emanuel Ioan
Oct 10



For the past year and some, I have been working with React at Creative Tim. I have been using create-react-app for developing some nice products. There have been a lot of clients asking how can someone migrate our product templates on Webpack.

So after a number of requests, we created this little tutorial about how to start using React with Webpack 4 and Babel 7. At the end of the tutorial, I am going to show you guys how to add Material Dashboard React on top of the newly created app.

Before we get started please make sure you have the latest versions of npm and Nodejs installed globally on your machine. At the time of writing this post, the latest versions were 6.4.1 for npm and 8.12.0 (lts) for Nodejs on my machine.

Creating a new project folder with package.json

First things first, let's create a **new folder** for our new **app** and enter it:

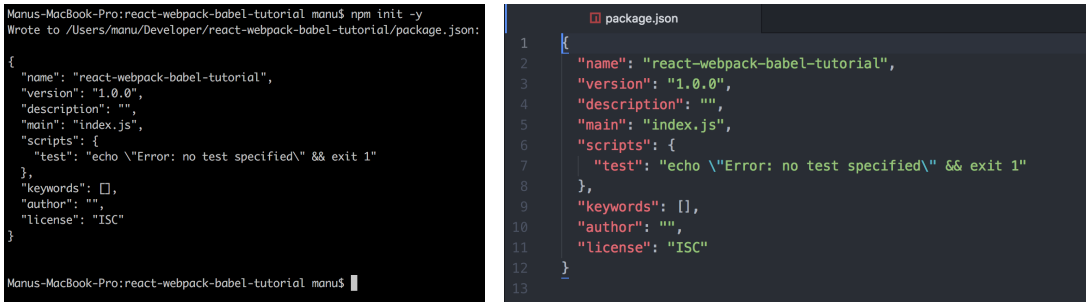
```
mkdir react-webpack-babel-tutorial
cd react-webpack-babel-tutorial
```

Now that we have created **the folder** in which we are going to develop the **app**, we need to add a **package.json** file to it. We can do this two ways and you should choose one of them:

> just create the **package.json** file without any other configuration:

```
npm init -y
```

As you can see, the **package.json** file has been created with some very basic information in it.

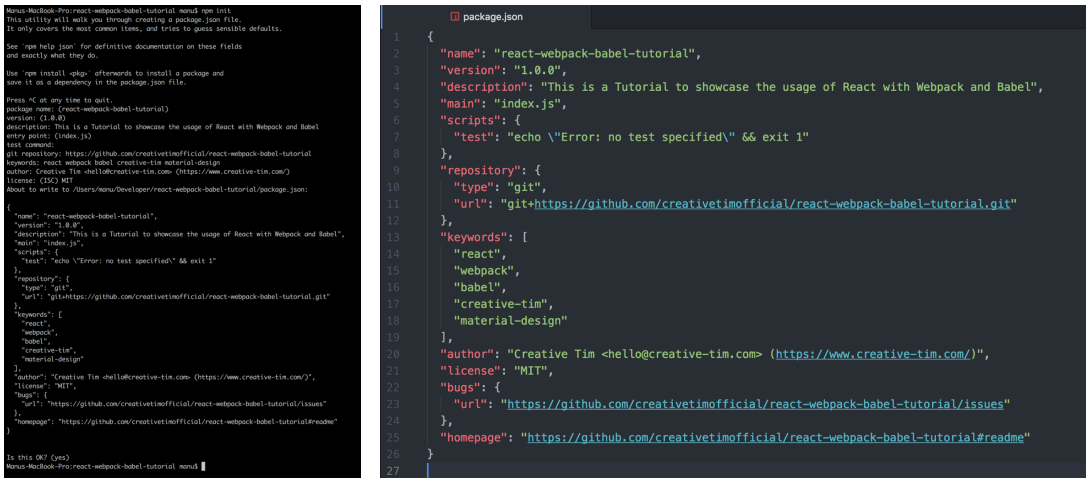


npm init -y output

2. create the **package.json** file with some extra config settings

```
npm init
```

I've added some stuff to our newly created **package.json** file, such as some nice **keywords**, **a repo** and so on...



npm init output

After this, let's add an **index.html** and **index.js** files to our new project folder, inside an **src** folder.

Linux/MacOS commands

```
mkdir src
touch src/index.html
touch src/index.js
```

2. Windows commands

```
mkdir src
echo "" > src\index.html
echo "" > src\index.js
```

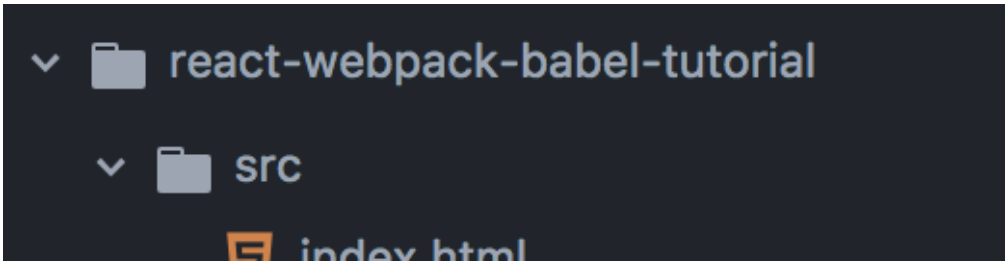
After this, let's add the following template inside the **index.html.**

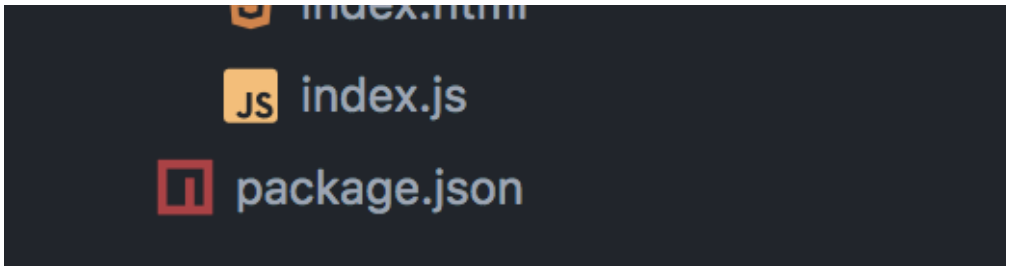
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1, shrink-to-fit=no">
    <meta name="theme-color" content="#000000">
    <title>React Tutorial</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will
see an empty page.

      You can add webfonts, meta tags, or analytics to
this file.
      The build step will place the bundled scripts
into the <body> tag.
    -->
  </body>
</html>
```

Let's add something inside the **index.js** just for the sake of some showcase that we are going to see a bit further down.

```
(function () {
  console.log("hey mister");
}());
```

And this is what we've got so far:





folder project structure

Adding Webpack to the project

Let's start adding all the **Webpack** packages that we are going to need. We are going to install them as **devDependencies** since they will be only used in development mode.

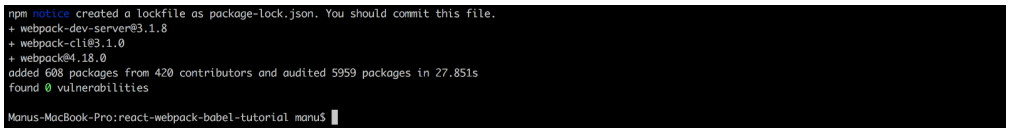
```
npm install --save-dev webpack webpack-cli webpack-dev-server
```

webpack
- used to configure our new app
- at the time of this post, the version was ***4.19.0***

webpack-cli
- used so that we can use Webpack in the command line
- at the time of this post, the version was ***3.1.0***

webpack-dev-server
- used so that when we make a change to a file inside our new app, we won't need to refresh the page. It refreshes the browser page automatically every time we change a file in our app
- as its name says, it's a server that is working non-stop
- at the time of this post, the version was ***3.1.8***

```
Manus-MacBook-Pro:react-webpack-babel-tutorial manu$ npm install --save-dev webpack webpack-cli webpack-dev-server

> fsevents@1.2.4 install /Users/manu/Developer/react-webpack-babel-tutorial/node_modules/fsevents
> node install

[fsevents] Success: "/Users/manu/Developer/react-webpack-babel-tutorial/node_modules/fsevents/lib/binding/Release/node-v57-darwin-x64/fse.node" already installed
Pass --update-binary to reinstall or --build-from-source to recompile
```



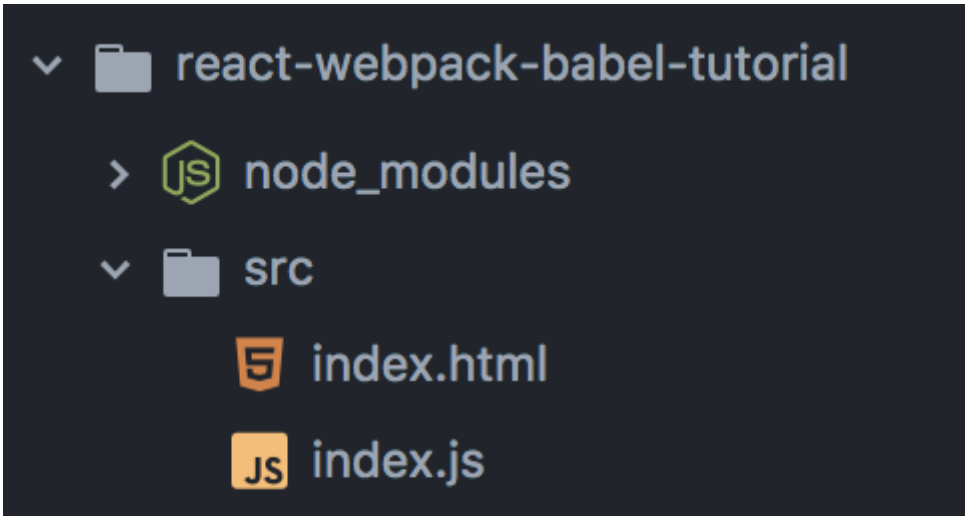
npm install — save-dev webpack webpack-cli webpack-dev-server output

If we take a look inside the **package.json** file, we are going to see that these three packages were added to this file like so:

```
"devDependencies": {
  "webpack": "^4.19.0",
  "webpack-cli": "^3.1.0",
  "webpack-dev-server": "^3.1.8"
}
```

I'm going to go ahead and delete the ^ (caret) from each version. This is because I can't tell whether the next version of these plugins is still going to work with what I am building. I think this is something that should be common sense. When creating a new app, use the available versions and then, maybe make some updates to newer versions. You might not know what a new version will break in your app.

As you will see, the installation of these plugins made some changes to our project folder. It added **node_modules** folder and **package-lock.json** to it.



project folder after installing **webpack**

Now, we need to add a new file to our project, the config file for **Webpack** called **webpack.config.js**:

Linux/MacOS command

```
touch webpack.config.js
```

2. Windows command

```
echo "" > webpack.config.js
```

Or you can simply manually create the new file if you do not want to use the command line.

Before we go ahead and start messing with the **Webpack config**file, let's first install some stuff that we are going to need in our app.

First, we are going to work with some paths inside the Webpack config file. Let's install **path** in our project as a **devDependency.**

```
npm install --save-dev path
```

Also, since we don't want to manually inject the **index.js** file inside the HTML one, we are going to need a plugin called **html-webpack-plugin. This plugin** will inject the **index.js** inside the HTML file without any manual operation.

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

Once again, I am going to edit my **package.json** file and delete all the ^ (caret) occurrences from it.

One more edit that we are going to make to our **package.json** is to add some new scripts inside the **scripts** object, after the **test**script (See the second example below).

```
"webpack": "webpack",
"start": "webpack-dev-server --open"
```

This is what your **package.json** should look like at this point:

```
{
  "name": "react-webpack-babel-tutorial",
  "version": "1.0.0",
  "description": "This is a Tutorial to showcase the
usage of React with Webpack and Babel",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit
1",
    "webpack": "webpack",
    "start": "webpack-dev-server --open"
  },
  "repository": {
    "type": "git",
    "url":
"git+https://github.com/creativetimofficial/react-
webpack-babel-tutorial.git"
  },
  "keywords": [
```

```
    "react",
    "webpack",
    "babel",
    "creative-tim",
    "material-design"
  ],
  "author": "Creative Tim <hello@creative-tim.com>
(https://www.creative-tim.com/)",
  "license": "MIT",
  "bugs": {
    "url":
"https://github.com/creativetimofficial/react-webpack-
babel-tutorial/issues"
  },
  "homepage":
"https://github.com/creativetimofficial/react-webpack-
babel-tutorial#readme",
  "devDependencies": {
    "html-webpack-plugin": "3.2.0",
    "path": "0.12.7",
    "webpack": "4.19.0",
    "webpack-cli": "3.1.0",
    "webpack-dev-server": "3.1.8"
  }
}
```

Let's go ahead and run these commands one by one and see what happens.

```
npm run webpack
```

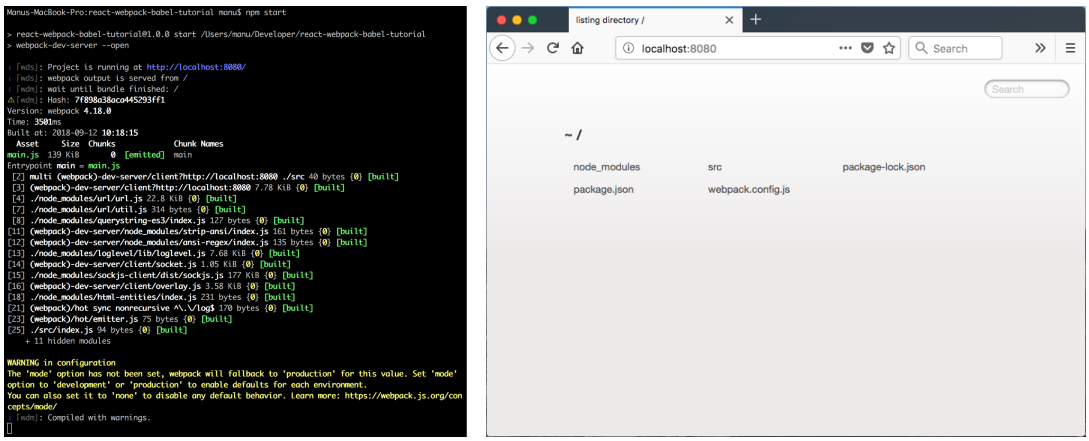
Webpack will automatically take the **src/index.js** file, compile it, and output it inside **dist/main.js** and will minify that code. This is because we haven't yet configured the **Webpack config**file. Also, since we haven't configured the file, we are going to have some warnings in our console.

npm run webpack output

If we run the other command

```
npm start
```

webpack-dev-server will automatically start a server and open the default browser with this server. But once again, since we do not have our **webpack.config.js** file configured, the output will not be the expected one.



npm start output

If you want to stop the server, just press at the same time the **CTRL** + **C** keys while in the command line.

Let's add the following template inside our **Webpack config** file:

```
const path = require('path');

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

module.exports = {
  entry: path.join(__dirname,'src','index.js'),
  output: {
    path: path.join(__dirname,'build'),
    filename: 'index.bundle.js'
  },
  mode: process.env.NODE_ENV || 'development',
  resolve: {
    modules: [path.resolve(__dirname, 'src'),
'node_modules']
  },
  devServer: {
    contentBase: path.join(__dirname,'src')
```

```
  },
  plugins: [
    new HtmlWebpackPlugin({
      template:
path.join(__dirname,'src','index.html')
    })
  ]
};
```

entry and **output**
—these are used to tell our server what has to be compiled and from where (*entry: path.join(__dirname,'src','index.js'),*). It also tells where to put the outputted compiled version (*output*—the folder and the filename)

mode
—this is the mode of our output. We are setting it to '*development*'. If in the scripts we specify the **NODE_ENV**variable, it will take that one instead. See the below example on how to use **NODE_ENV** *(note that the below changes will not be made inside the* ***package.json*** *file in this tutorial, it is just an example for you to see how it works)*

```
"webpack": "NODE_ENV=production webpack",
```

resolve
—this is used so that we can import anything from **src** folder in relative paths instead of absolute ones. It is the same for the **node_modules.** We can import anything from node_modules directly without absolute paths

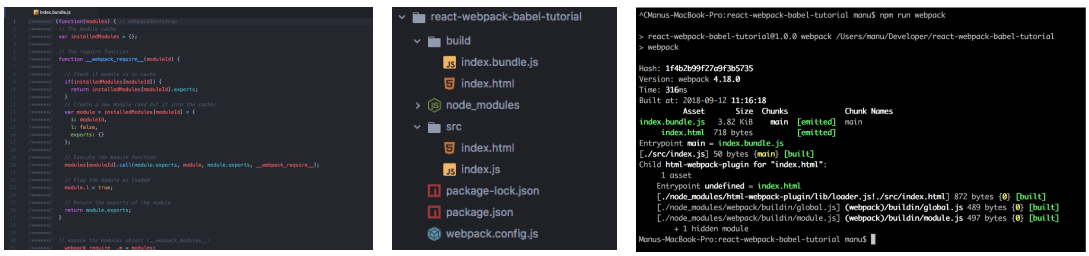
devServer
—this tells the **webpack-dev-server** what files are

needed to be served. Everything from our **src** folder needs to be served (outputted) in the browser

plugins
—here we set what plugins we need in our app. As of this moment we only need the **html-webpack-plugin** which tells the server that the **index.bundle.js** should be injected (or added if you will) to our **index.html** file

If we now run the earlier commands we will see some differences.

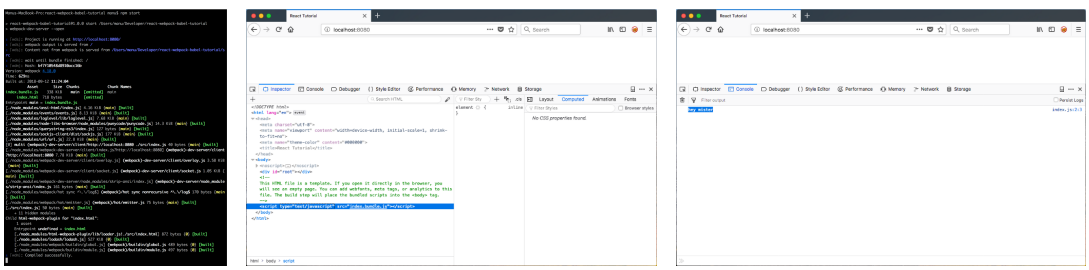
```
npm run webpack
```



npm run webpack output with **webpack.config.js**

We've changed where the output should be (from **dist** folder to **build** folder). By changing the **mode** of **Webpack**, now the code has a different look. It is not **minified** as the last time with no **config**.

```
npm start
```



npm start output with **webpack.config.js**

The **webpack-dev-server** took everything from the **src** folder and outputted it to our browser.

We are on the right path, but we've only added Webpack to our project. Where are React and Babel? Well, that is what we are going to do next.

React, Babel and some nice loaders for styles

Add **React** and **ReactDOM** to our project as **normal dependencies**.

```
npm install --save react react-dom
```

At this moment in our development, if we were to add **React**code inside our JS file, **Webpack** will give us an error. It doesn't know how to compile **React** inside the **bundle.js** file.

Let's modify the **index.js** file as follows:

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

let HelloWorld = () => {
  return <h1>Hello there World!</h1>
}

ReactDOM.render(
  <HelloWorld/>,
  document.getElementById("root")
);
```

And after that let's start the server again.

```
npm start
```

And this is the error:

webpack error for not having appropriate **loaders** for **react**

So this is where **Babel** comes to our aid. **Babel** will tell **Webpack**how to compile our **React** code.

Let's go ahead and add a bunch of Babel packages to our app as **devDependencies**.

```
npm install --save-dev @babel/core @babel/node
@babel/preset-env @babel/preset-react babel-loader
```

@babel/core
—this is used to compile **ES6** and above into **ES5**

@babel/node
—this is used so that we can **import** our plugins and packages inside the **webpack.config.js** rather than **require**them (it's just something that I like, and maybe you'll like it too)

@babel/preset-env
—this will determinate which transformations or plugins to

use and polyfills (i.e it provides modern functionality on older browsers that do not natively support it) based on the browser matrix you want to support

@babel/preset-react
—this is going to compile the **React** code into **ES5** code

babel-loader
—this is a **Webpack** helper that transforms your **JavaScript**dependencies with **Babel** (i.e. will transform the **import**statements into **require** ones)

Since you are probably going to need to add some styles to your project (I know that I need them), we are going to add a loader that will let us **import** and use **CSS** files and **SCSS** files.

```
npm install --save-dev style-loader css-loader sass-
loader node-sass
```

style-loader
—this will add to the **DOM** the styles (will inject a **<style>**tag inside the **HTML** file)

css-loader
—will let us import **CSS** files into our project

sass-loader
—will let us import **SCSS** files into our project

node-sass
—will compile the **SCSS** files into normal **CSS** files

We are going to create a new **SCSS** file and add it to our folders.

Linux/MacOS command

```
touch src/index.scss
```

2. Windows command

```
echo "" > src/index.scss
```

And also add some nice styles to it.

```
body {
  div#root{
    background-color: #222;
    color: #8EE4AF;
  }
}
```

And change our **index.js** by adding an import for the **SCSS** file.

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

// this line is new
// we now have some nice styles on our react app
import "index.scss";

let HelloWorld = () => {
  return <h1>Hello there World!</h1>
}

ReactDOM.render(
  <HelloWorld/>,
  document.getElementById("root")
);
```

Don’t forget to delete the carets (^) from **package.json**.

This is how your **package.json** should look like:

```
{
  "name": "react-webpack-babel-tutorial",
  "version": "1.0.0",
  "description": "This is a Tutorial to showcase the
usage of React with Webpack and Babel",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit
1",
    "webpack": "webpack",
    "start": "webpack-dev-server --open"
  },
  "repository": {
    "type": "git",
    "url":
"git+https://github.com/creativetimofficial/react-
webpack-babel-tutorial.git"
  },
  "keywords": [
    "react",
    "webpack",
    "babel",
    "creative-tim",
    "material-design"
  ],
  "author": "Creative Tim <hello@creative-tim.com>
(https://www.creative-tim.com/)",
  "license": "MIT",
  "bugs": {
    "url":
"https://github.com/creativetimofficial/react-webpack-
babel-tutorial/issues"
  },
  "homepage":
"https://github.com/creativetimofficial/react-webpack-
babel-tutorial#readme",
  "devDependencies": {
    "@babel/core": "7.0.1",
    "@babel/node": "7.0.0",
    "@babel/preset-env": "7.0.0",
    "@babel/preset-react": "7.0.0",
    "babel-loader": "8.0.2",
    "css-loader": "1.0.0",
    "html-webpack-plugin": "3.2.0",
    "node-sass": "4.9.3",
    "path": "0.12.7",
    "sass-loader": "7.1.0",
    "style-loader": "0.23.0",
    "webpack": "4.19.0",
    "webpack-cli": "3.1.0",
    "webpack-dev-server": "3.1.8"
  },
  "dependencies": {
    "react": "16.5.1",
    "react-dom": "16.5.1"
  }
}
```

If we run any of the above commands again, the error will still persist. We haven't yet told **Webpack** that it should use **Babel**and the style loaders to compile our **React** and **SCSS** code.

Next thing to do is add a configuration file for **Babel**. For this we need to create a file named **.babelrc** in which we will configure **Babel**.

I've heard that you can add the configuration for **Babel** directly in the **webpack.config.js** file. For this, you can take a look at the official babel-loader docs. As far as I am concerned, I think it's best to have the **Babel** config in its own file. That way you do not overcrowd your **Webpack** config.

So, let's run in the command line the following:

Linux/MacOS command

```
touch .babelrc
```

2. Windows command

```
echo "" > .babelrc
```

And add the following code inside **.babelrc** so that **babel-loader**will know what to use to compile the code:

```
{
  "presets": [
    "@babel/env",
    "@babel/react"
  ]
}
```

After these steps, we will need to add something to our project so we can import all sorts of files such as images. We will also need to add a plugin that will let us work with classes and much more. Let us add class properties in our classes. Basically, it will let us work with Object Oriented Programming—nice.

```
npm install --save-dev file-loader @babel/plugin-
proposal-class-properties
```

Now that we have done this, we need to make some changes inside **webpack.config.js** so that **Webpack** will now use **Babel**. We'll also configure **Webpack** to listen for **style** files and we are going to change the **require** statements to **import** ones.

So this being said, let's change our **webpack.config.js** to the following (I've also added some comments, maybe they will help you):

```
// old
// const path = require('path');
// const HtmlWebpackPlugin = require('html-webpack-
plugin');

// new
import path from 'path';

import HtmlWebpackPlugin from 'html-webpack-plugin';

module.exports = {
  entry: path.join(__dirname,'src','index.js'),
  output: {
    path: path.join(__dirname,'build'),
    filename: 'index.bundle.js'
  },
  mode: process.env.NODE_ENV || 'development',
  resolve: {
    modules: [path.resolve(__dirname, 'src'),
'node_modules']
  },
  devServer: {
    contentBase: path.join(__dirname,'src')
  },
  module: {
```

```
    rules: [
      {
        // this is so that we can compile any React,
        // ES6 and above into normal ES5 syntax
        test: /\.(js|jsx)$/,
        // we do not want anything from node_modules
to be compiled
        exclude: /node_modules/,
        use: ['babel-loader']
      },
      {
        test: /\.(css|scss)$/,
        use: [
          "style-loader", // creates style nodes from
JS strings
          "css-loader", // translates CSS into
CommonJS
          "sass-loader" // compiles Sass to CSS, using
Node Sass by default
        ]
      },
      {
        test: /\.(jpg|jpeg|png|gif|mp3|svg)$/,
        loaders: ['file-loader']
      }
    ]
  },
  plugins: [
    new HtmlWebpackPlugin({
      template:
path.join(__dirname,'src','index.html')
    })
  ]
};
```

There's one more change we need to do to the **package.json** file. We need to tell our scripts that inside the config files of **Webpack**, we use **import** instead of **require** statements. Else it will give us an error that it doesn't know what **import** stands for.

```
{
  "name": "react-webpack-babel-tutorial",
  "version": "1.0.0",
  "description": "This is a Tutorial to showcase the
usage of React with Webpack and Babel",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit
1",
    "webpack": "babel-node
./node_modules/webpack/bin/webpack",
```

```
    "start": "babel-node ./node_modules/webpack-dev-
server/bin/webpack-dev-server --open"
  },
  "repository": {
    "type": "git",
    "url":
"git+https://github.com/creativetimofficial/react-
webpack-babel-tutorial.git"
  },
  "keywords": [
    "react",
    "webpack",
    "babel",
    "creative-tim",
    "material-design"
  ],
  "author": "Creative Tim <hello@creative-tim.com>
(https://www.creative-tim.com/)",
  "license": "MIT",
  "bugs": {
    "url":
"https://github.com/creativetimofficial/react-webpack-
babel-tutorial/issues"
  },
  "homepage":
"https://github.com/creativetimofficial/react-webpack-
babel-tutorial#readme",
  "devDependencies": {
    "@babel/core": "7.0.1",
    "@babel/node": "7.0.0",
    "@babel/plugin-proposal-class-properties":
"7.0.0",
    "@babel/preset-env": "7.0.0",
    "@babel/preset-react": "7.0.0",
    "babel-loader": "8.0.2",
    "css-loader": "1.0.0",
    "file-loader": "2.0.0",
    "html-webpack-plugin": "3.2.0",
    "node-sass": "4.9.3",
    "path": "0.12.7",
    "sass-loader": "7.1.0",
    "style-loader": "0.23.0",
    "webpack": "4.19.0",
    "webpack-cli": "3.1.0",
    "webpack-dev-server": "3.1.8"
  },
  "dependencies": {
    "react": "16.5.1",
    "react-dom": "16.5.1"
  }
}
```

Another thing that we will have to still add is the **@babel/plugin-proposal-class-properties** to

the **.babelrc** file. Babel will know how to deal with class properties.

```
{
  "presets": [
    "@babel/env",
    "@babel/react"
  ],
  "plugins": [
    "@babel/plugin-proposal-class-properties"
  ]
}
```

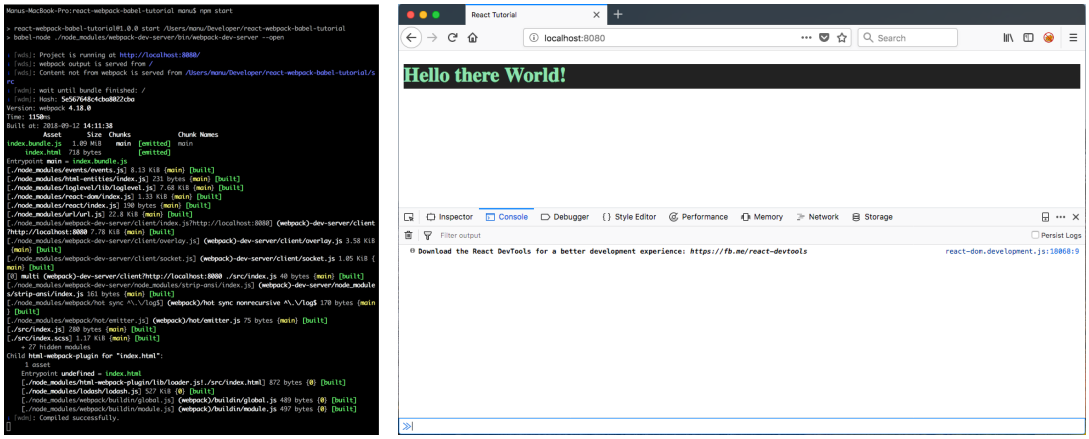
Now we are done. We can run either one of the above commands and it should not give us any errors. Let's see them in action.

```
npm run webpack
```

npm run webpack with no errors

And now let's see the main script of our app.

```
npm start
```

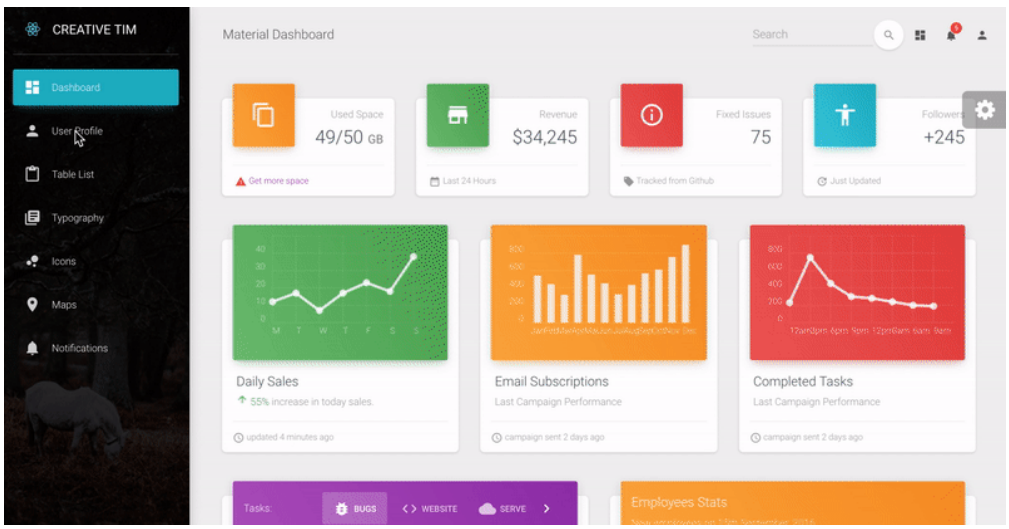


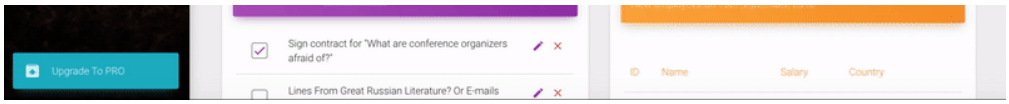
npm start output

Add Material Design to our new React with Webpack and Babel project

As I've told you at the beginning of this post, we are not going to create from scratch styles for Material Design. That would require a lot of work. We don't have time for that.

Instead, we are going to add a nice product that implements Google's Material Design with some minor touches from the Creative Tim staff. We are going to add Material Dashboard React to it.





First things first, you need to get the product. Here are a few ways of getting the product:

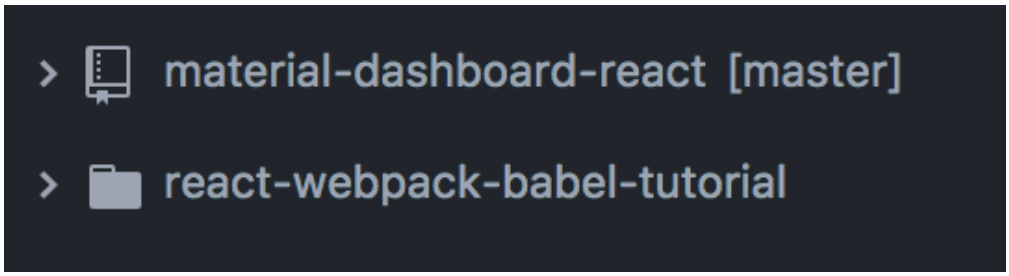
Clone the repo inside another folder:

```
git clone
https://github.com/creativetimofficial/material-
dashboard-react.git
```

Download from Github

Download from Creative Tim

Ok, so now we have both projects—Material Dashboard React and our newly created one with **Webpack** and **Babel**—with **React**.



material-dashboard-react and **react-webpack-babel-tutorial**

Now, we can't simply copy the src folder from **Material Dashboard React** into our new project. That will give us a lot of errors. Such as errors for missing dependencies, module not found, you get the point, a lot of errors.

So, I suggest that we start with adding the dependencies from **Material Dashboard React**'s **package.json** to

our **package.json**. We do not need all the dependencies from **Material Dashboard React's** packages, since we have built our own server using **Webpack.** We have added other style loaders beyond what the product has.

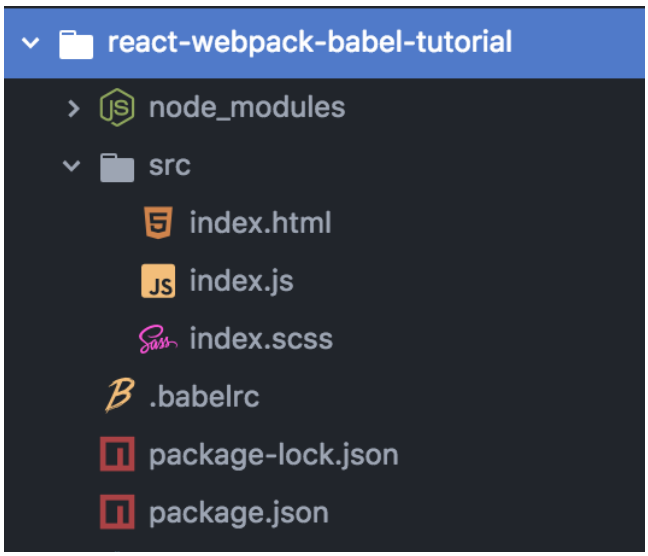
So this being said, we need the following:

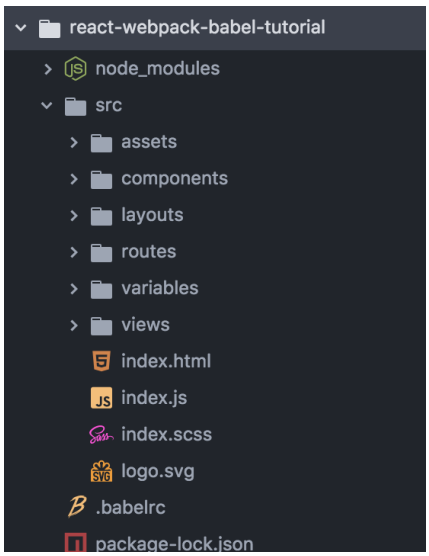
```
npm install --save @material-ui/core@3.1.0 @material-
ui/icons@3.0.1 @types/googlemaps@3.30.11
@types/markerclustererplus@2.1.33 chartist@0.10.1
classnames@2.2.6 perfect-scrollbar@1.4.0 react-
chartist@0.13.1 react-google-maps@9.4.5 react-router-
dom@4.3.1 react-swipeable-views@0.12.15
```

We are not going through all of them. They can be found on npmjs.com with all the details and their own documentation.

Once again, we go inside the **package.json** file and delete the carets (^) from the packages that we just installed.

Ok, we are almost done. We are going to copy all the contents of the **src** folder from **Material Dashboard React** inside our project's **src** folder and override the **index.js** file. But keep it in the **index.html** file.





Folder structure before and after adding the Material Dashboard React **src** folder

Now we need to add some styles and fonts cdns inside our **index.html**.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1, shrink-to-fit=no">
    <meta name="theme-color" content="#000000">
    <link rel="stylesheet"
href="//cdn.jsdelivr.net/chartist.js/latest/chartist.min.
    <script
src="//cdn.jsdelivr.net/chartist.js/latest/chartist.min.j
</script>
    <link rel="stylesheet"
href="https://fonts.googleapis.com/css?
family=Roboto:300,400,500,700|Material+Icons">
    <link href="https://fonts.googleapis.com/icon?
family=Material+Icons" rel="stylesheet">
    <title>React Tutorial</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will
see an empty page.

      You can add webfonts, meta tags, or analytics to
this file.
      The build step will place the bundled scripts
into the <body> tag.
    -->
  </body>
</html>
```

And we are almost there. We still have a small problem. When we refresh the page, we have an error **Cannot GET /dashboard.** If we navigate to another page we will get, for example, **Cannot GET /user** and so on. So basically, our

routes do not work. We need to make some changes inside either **src/index.js** or inside our **webpack.config.js**.

I will choose the first option since it is pretty straightforward and easy to understand.

We navigate inside the new index.js and we change the history type. We put **createHashHistory()** instead of **createBrowserHistory()**.

This will allow us to refresh the page without any other errors. Now we are done.

```
import React from "react";
import ReactDOM from "react-dom";
import { createHashHistory } from "history";
import { Router, Route, Switch } from "react-router-
dom";

import "assets/css/material-dashboard-react.css?
v=1.5.0";

import indexRoutes from "routes/index.jsx";

const hist = createHashHistory();

ReactDOM.render(
  <Router history={hist}>
    <Switch>
      {indexRoutes.map((prop, key) => {
        return <Route path={prop.path} component=
{prop.component} key={key} />;
      })}
    </Switch>
  </Router>,
  document.getElementById("root")
);
```

I really hope you've liked this tutorial and I am very keen on hearing your thoughts about it. Just give this thread a comment and I'll be more than happy to reply.

Special thanks should also go to Linh Nguyen My for her tutorialwhich has given me some much needed

understanding on **Webpack**.

Useful links:

- Get the code for this tutorial from Github
- Read more about ReactJS on their official website
- Read more about Webpack here
- Read more about Babel on this link here
- Read more about Material Design
- Check out our platform to see what we are doing and who we are
- Get Material Dashboard React from www.creative-tim.comor from Github
- Read more about Material-UI, the core of Material Dashboard React

Find me on:

- Email: manu@creative-tim.com
- Facebook: https://www.facebook.com/NazareEmanuel
- Instagram: https://www.instagram.com/manu.nazare/