# The Minimal React + Webpack 4 + Babel Setup



JANUARY 18, 2018

Personally I bootstrapped a lot of React projects in the recent time. I always had to setup the project from scratch. Eventually I have created my own boilerplate project on GitHub (https://github.com/rwieruch/minimal-react-webpack-babel-setup). As you might know, uncountable React boilerplate projects and repositories were created that way.

But the article is not my attempt to advertise yet another React boilerplate project. I had several reasons why I extracted the setup process from another article of mine.

First, I can reuse it for all my other articles of my website whenever there is a React project setup involved. You might be reading the article right now, because you are in the middle of another article.

Second, it helps me to maintain the React setup at one place. It is my single source of truth. Whenever there are updates regarding React, Webpack, Babel or Hot Reloading, I can come back to this one article to keep all other articles updated.

Third, a single source of truth has to be well maintained. When several of my articles reference this one article to bootstrap an React application with Webpack and Babel, I am enforced to maintain it well. People, who search about setting up their React, Webpack and Babel environment, will hopefully always find an up to date version of the article. I really appreciate any feedback, issue reports and improvements for the article.

Fourth, the article is not about the boilerplate project itself. The article is more about teaching people how to setup their own project without a boilerplate project. At some point, you will start to use the tools around your library or framework of choice. In JavaScript you will have to deal with Webpack, Babel et al. at some point.

Last but not least, there is already a great official way introduced by Facebook to bootstrap a boilerplate React project. create-react-app (https://github.com/facebookincubator/create-react-app) comes without any build configuration. I can recommend it for everyone who is getting started in React. If you are a beginner, you probably shouldn't bother with a setup of Webpack, Babel and Hot Reloading. I use create-react-app to teach plain React in my book the Road to learn React (https://www.robinwieruch.de/the-road-to-learn-react/). I can recommend to read it before you get started with the tooling around React.

That's enough about my motivation behind the article. Let's dive into my personal minimal setup for a React project. This tutorial is up to the recent Webpack 4 version. It should work for Webpack 3 as well.

# Table of Contents


- React Project Setup
- Webpack Setup
- Babel Setup
- React Setup in a Webpack + Babel Project
- Hot Module Replacement


---

# React Project Setup

You need some requirements before you can start. First you should have an editor and terminal (https://www.robinwieruch.de/developer-setup/) on your machine. Second you will need an installed version of node with npm (https://nodejs.org/en/).

The first chapter concentrates on setting up the project. Let's create a new folder and initialize it as a npm project.

*In your terminal type:*

```
mkdir minimal-react-boilerplate
cd minimal-react-boilerplate
npm init -y
```

The last command should have generated a *package.json* file. The `-y` indicates that all default configurations should be used. In the *package.json* file you will find configurations, installed node packages and scripts later on.

The next step is to create a distribution folder. The folder will be used to serve the single page application (SPA). Serving the app makes it possible to view it in the browser or host it on an external server to make it accessible for everyone.

The whole served SPA contains only of two files: a *.html* and a *.js* file. While the *.js* file will be generated automatically from all of your JavaScript source files (via Webpack) later, you can already create the *.html* file manually as an entry point for your application.

*From root folder (minimal-react-boilerplate):*

```
mkdir dist
cd dist
touch index.html
```

As a side note: The distribution folder will be everything you need to publish your web app to a hosting server. It will only need the HTML and JS file to serve your app.

The created file should have the following content:

*dist/index.html*

```
<!DOCTYPE html>
<html>
  <head>
    <title>The Minimal React Webpack Babel Setup</title>
  </head>
  <body>
    <div id="app"></div>
    <script src="/bundle.js"></script>
  </body>
</html>
```

Two important facts about the content:

- the bundle.js file will be a generated file by Webpack (1)
- the id="app" attribute will help our root React component to find its entry point (2)

Therefore our next possible steps are:

- (1) setup Webpack to bundle our source files in one file as bundle.js
- (2) build our first React root component which uses the entry point id="app"

Let's continue with the first step followed by the latter one.

---

## Webpack Setup

You will use Webpack (https://github.com/webpack/webpack) as module bundler and build tool. Moreover you will use webpack-dev-server (https://github.com/webpack/webpack-dev-server) to serve your bundled app in a local environment. Otherwise you couldn't see it in the browser to develop it. Last but not least, you need the webpack-cli (https://github.com/webpack/webpack-cli) node package to configure your Webpack setup in a configuration file later on. Let's install all three node packages by using npm.

*From root folder:*

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

Now you should have a *node_modules* folder where you can find your third party dependencies. The dependencies will be listed in the *package.json* file as well, since you used the *–save-dev* flag. Your folder structure should look like the following by now:

*Folder structure:*

```
- dist
-- index.html
- node_modules
- package.json
```

In the *package.json* file you can add a start script additionally to the default given scripts to run the webpack-dev-server.

*package.json*

```
...
"scripts": {
  "start": "webpack-dev-server --config ./webpack.config.js --mode development",
  ...
},
...
```

The script defines that you want to use the webpack-dev-server with a configuration file called *wepback.config.js*. The `--mode development` flag just adds default Webpack configurations which came with Webpack 4. You wouldn't need the flag for Webpack 3.

Let's create the required *wepback.config.js* file.

*From root folder:*

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

You can continue by providing the following content:

*webpack.config.js*

```
module.exports = {
  entry: [
    './src/index.js'
  ],
  output: {
    path: __dirname + '/dist',
    publicPath: '/',
    filename: 'bundle.js'
  },
  devServer: {
    contentBase: './dist'
  }
};
```

Roughly the configuration file says that (1) we want to use the *src/index.js* file as entry point to bundle all of its imported files. (2) The bundled files will result in a *bundle.js* file which (3) will be generated in our already set up */dist* folder. The */dist* folder will be used to serve our app.

What is missing in our project is the *src/index.js* file.

*From root folder:*

```
mkdir src
cd src
touch index.js
```

*src/index.js*

```
console.log('My Minimal React Webpack Babel Setup');
```

*Folder structure:*

```
- dist
-- index.html
- node_modules
- src
-- index.js
- package.json
- webpack.config.js
```

Now you should be able to start your webpack-dev-server.

*From root folder:*

```
npm start
```

You can open the app in a browser (http://localhost:8080/). Additionally you should see the `console.log()` in the developer console.

You are serving your app via Webpack now. You bundle your entry point file *src/index.js* as *bundle.js*, use it in *dist/index.html* and can see the `console.log()` in the developer console. For now it is only the *src/index.js* file. But you will import more JS files later on in that file, which will get bundled automatically by Webpack in the *bundle.js* file.

---

## Babel Setup

Babel (https://babeljs.io/) enables you writing your code in ES6 (ES2015) (https://babeljs.io/docs/learn-es2015/). With Babel the code will get transpiled back to ES5 so that every browser, without having all ES6 features implemented, can interpret it. Babel even takes it one step further. You can not only use ES6 features, but also the next generations of ES.

*From root folder:*

```
npm install --save-dev babel-core babel-loader babel-preset-env
```

Additionally you might want to use some more experimental features in ES6 (e.g. object spread (https://github.com/sebmarkbage/ecmascript-rest-spread)) which can get activated via stages (https://babeljs.io/docs/plugins/preset-stage-0/). No worries, even though it is experimental, it is already used in create-react-app by Facebook too.

*From root folder:*

```
npm install --save-dev babel-preset-stage-2
```

As last step, since you want to use React, you need one more configuration to transform the natural React *.jsx* files to *.js* files. It is for the sake of convenience.

*From root folder:*

```
npm install --save-dev babel-preset-react
```

Now, with all node packages in place, you need to adjust your *package.json* and *webpack.config.js* to respect the Babel changes. These changes include all packages you have installed.

*package.json*

```
...
"keywords": [],
"author": "",
"license": "ISC",
"babel": {
  "presets": [
    "env",
    "react",
    "stage-2"
  ]
},
"devDependencies": {
...
```

*webpack.config.js*

```
module.exports = {
  entry: [
    './src/index.js'
  ],
  module: {
    rules: [
      {
        test: /\.(js|jsx)$/,
        exclude: /node_modules/,
        use: ['babel-loader']
      }
    ]
  },
  resolve: {
    extensions: ['*', '.js', '.jsx']
  },
  output: {
    path: __dirname + '/dist',
    publicPath: '/',
    filename: 'bundle.js'
  },
  devServer: {
    contentBase: './dist'
  }
};
```

You can start your application again. Nothing should have changed except for that you can use upcoming ECMAScript functionalities for JavaScript now. An optional step would be to extract your Babel configuration in a separate *.babelrc* configuration file.

*From root folder:*

```
touch .babelrc
```

Now you can add the configuration for Babel, which you have previously added in your *package.json*, in the *.babelrc* file. Don't forget to remove the configuration in the *package.json* afterward. It should be configured at only one place.

*.babelrc*

```
{
  "presets": [
    "env",
    "react",
    "stage-2"
  ]
}
```

You are set up to build your first React component now.

---

## React Setup in a Webpack + Babel Project

In order to use React, you need two more node packages. The react and react-dom packages should fix your npm start.

*From root folder:*

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

In your *src/index.js* you can implement your first hook into the React world.

*src/index.js*

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

const title = 'My Minimal React Webpack Babel Setup';

ReactDOM.render(
  <div>{title}</div>,
  document.getElementById('app')
);
```

You should be able to see the output in your browser rather than in a developer console now.

`ReactDOM.render` needs two parameters. The first parameter is your JSX. It has to have always one root node. The second parameter is the node where your output should be appended. Remember when we used `<div id="app"></div>` in the *dist/index.html* file? The same id is your entry point for React now.

---

## Hot Module Replacement in React

A huge development boost will give you react-hot-loader (https://github.com/gaearon/react-hot-loader) (Hot Module Replacement). It will shorten your feedback loop during development. Basically whenever you change something in your source code, the change will apply in your app running in the browser without reloading the entire page (https://www.youtube.com/watch?v=xsSnOQynTHs).

*From root folder:*

```
npm install --save-dev react-hot-loader
```

You have to add some more configuration to your Webpack configuration file.

*webpack.config.js*

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

module.exports = {
  entry: [
    'react-hot-loader/patch',
    './src/index.js'
  ],
  module: {
    rules: [
      {
        test: /\.(js|jsx)$/,
        exclude: /node_modules/,
        use: ['babel-loader']
      }
    ]
  },
  resolve: {
    extensions: ['*', '.js', '.jsx']
  },
  output: {
    path: __dirname + '/dist',
    publicPath: '/',
    filename: 'bundle.js'
  },
  plugins: [
    new webpack.HotModuleReplacementPlugin()
  ],
  devServer: {
    contentBase: './dist',
    hot: true
  }
};
```

Additionally in the *src/index.js* file you have to define that hot reloading is available and should be used.

*src/index.js*

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

const title = 'My Minimal React Webpack Babel Setup';

ReactDOM.render(
  <div>{title}</div>,
  document.getElementById('app')
);

module.hot.accept();
```

Now you can start your app again.

*From root folder:*

```
npm start
```

When you change your `title` for the React component in the *src/index.js* file, you should see the updated output in the browser without any browser reloading. If you would remove the `module.hot.accept();` line, the browser would perform a reload if something has changed in the code.

. . .

That's it for a minimal React setup with Babel and Webpack. Let me know your thoughts. Again, you can find the repository on GitHub (https://github.com/rwieruch/minimal-react-webpack-babel-setup). You can contribute by creating issues when new versions introduce breaking changes. Even more you can have a direct impact on this article on GitHub as well.

**Read more: React Code Style with ESLint + Babel + Webpack (https://www.robinwieruch.de/react-eslint-webpack-babel/)**