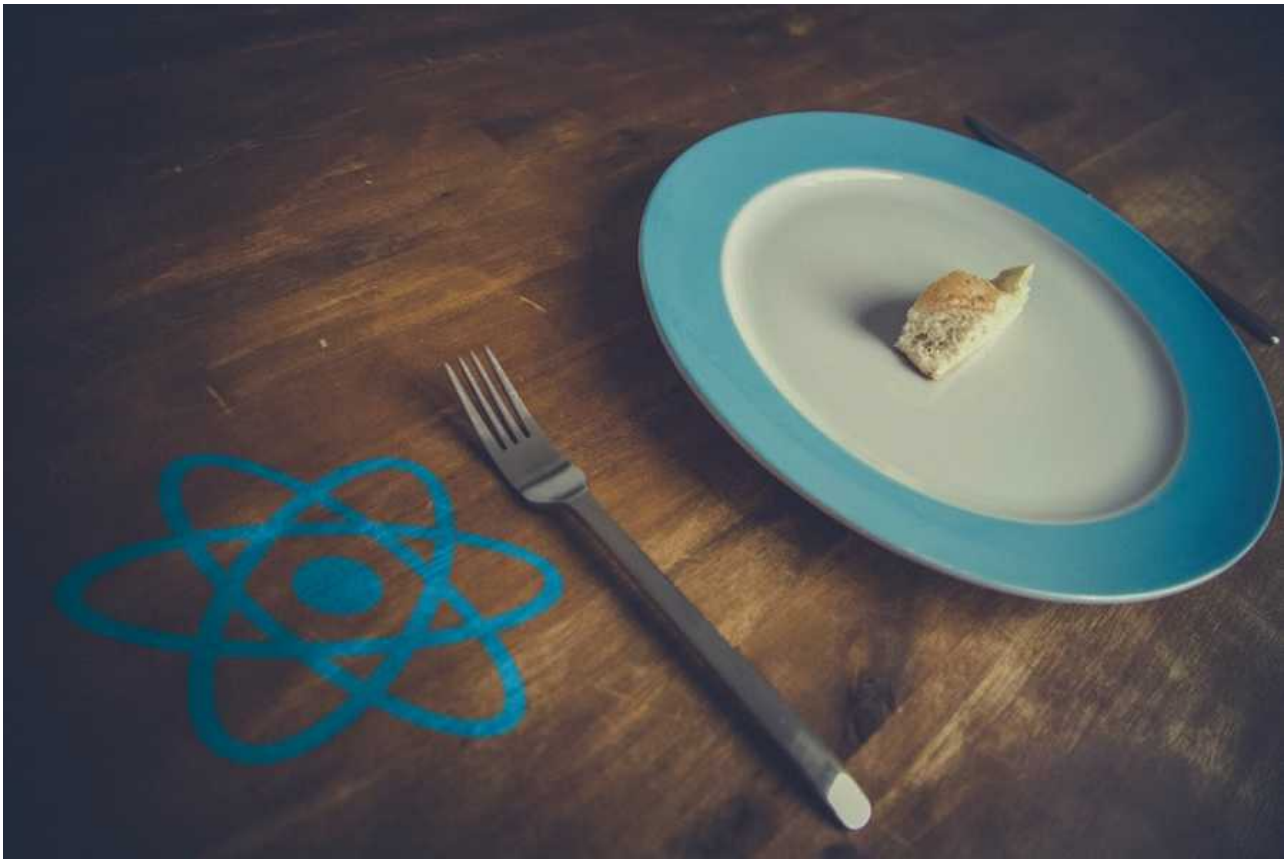
How to React with Webpack 5 - Setup Tutorial

OCTOBER 30, 2020 BY ROBIN WIERUCH - EDIT THIS POST



This tutorial is part 4 of 4 in the 'React Setup'-series.

Part 1: How to set up a modern JavaScript project
Part 2: How to set up Webpack 5
Part 3: How to set up Webpack 5 with Babel

Personally I bootstrapped a lot of React projects over the last years. I always had to setup the project from scratch, however, eventually I have created my own boilerplate project on GitHub for it. 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. Instead, I had several reasons why I extracted the setup process from another article of mine.

First, I can reuse it for all my other tutorials on my website whenever there is a React project setup involved. Also people from other websites started to use this tutorial as guide for getting started with React and Webpack.

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 Module Replacement, I can come back to this one tutorial to keep all other tutorials updated.

Third, a single source of truth has to be well maintained. If several of my tutorials reference this one tutorial to set up a React application with Webpack, I am forced to maintain it well. People, who search about a React with Webpack setup, will hopefully always find an up to date version of this tutorial. I really appreciate any feedback, issue reports and improvements for it.

Fourth, the tutorial is not about the boilerplate project itself. The tutorial is more about teaching people how to setup their own project without a third-party boilerplate project. At some point, you will start to use the tools (e.g. Webpack, Babel) around your library or framework of choice. In JavaScript you will have to deal with Webpack, Babel et al. and thus it makes sense to learn about them. I hope this tutorial helps you with this adventure.

Last but not least, there is already a great official way introduced by Facebook to start a React project: create-react-app comes without any build configuration which I can only recommend for anyone who is getting started with React. If you are a beginner, you probably shouldn't bother with a setup of Webpack and Babel yourself. I use create-react-app to teach plain React in my book the Road to learn React too. You should take the time to read it before you get started with the tooling around React with this tutorial.

That should be enough said about my motivation behind this tutorial. Let's dive into my personal minimal setup for a React project. This tutorial supports the latest versions of React, Webpack 5, and Babel 7.

TABLE OF CONTENTS

REACT WITH BABEL

The application we have built so far enables you to write JavaScript applications with Webpack and Babel. Whereas Webpack bundles all our JavaScript source code files into one bundle (including custom configured build steps), Babel enables us to use recent JavaScript features that are not supported by many browsers yet. That's why Babel is also needed for React, because JSX -- React's syntax -- and its file extension *.jsx*, aren't natively supported. Babel makes sure to transpile our React code to vanilla JavaScript. Therefore, you have to install the following Babel Preset for React on your command line:

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

In your *.babelrc* (or *package.json*) file -- depending on where you have your Babel configuration for presets and plugins -- add the new preset. In this tutorial, we will add it in our *.babelrc* file:

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

Next, let's inform Webpack in our *webpack.config.js* file about files with the JSX extension to make sure that they run through the transpiling step as well:

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

module.exports = {
  entry: path.resolve(__dirname, './src/index.js'),
  module: {
    rules: [
      {
        test: /\.(js|jsx)$/,
        exclude: /node_modules/,
        use: ['babel-loader'],
      },
    ],
  },
  resolve: {
```

```
    resolve: {
      extensions: ['*', '.js', '.jsx'],
    },
    output: {
      path: path.resolve(__dirname, './dist'),
      filename: 'bundle.js',
    },
    devServer: {
      contentBase: path.resolve(__dirname, './dist'),
    },
  };
```

That's it for enabling React in JavaScript by using Babel and Webpack. We are allowed to write React with JSX now.

REACT WITH WEBPACK

So far, you should have the following folder structure for your JavaScript application that uses Webpack and Babel:

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

In order to use React, you need two libraries (node packages): react and react-dom. Install them on the command line from your project's root folder:

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

In your *src/index.js*, you can implement your entry point into the React world:

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

const title = 'React with Webpack and Babel';

ReactDOM.render(
  <div>{title}</div>,
```

```
  document.getElementById('app')
);
```

The React DOM API takes two arguments. Whereas the first argument is the rendered JSX from React, the second argument is the HTML element where it should be integrated into the DOM. Since it expects an HTML element identified by an id attribute, we need to add this element in our *dist/index.html* file:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello React</title>
  </head>
  <body>
    <div id="app"></div>
    <script src="./bundle.js"></script>
  </body>
</html>
```

Start your application with `npm start` again. You should be able to see the output of your React implementation in your browser.

HOT MODULE REPLACEMENT IN REACT

A huge development boost will give you 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. First, install it from your project's root directory on the command line:

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

Second, add the configuration to your *webpack.config.js* file:

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

module.exports = {
  entry: path.resolve(__dirname, './src/index.js'),
  module: {
    rules: [
```

```
      {
        test: /\.(js|jsx)$/,
        exclude: /node_modules/,
        use: ['babel-loader'],
      },
    ],
  },
  resolve: {
    extensions: ['*', '.js', '.jsx'],
  },
  output: {
    path: path.resolve(__dirname, './dist'),
    filename: 'bundle.js',
  },
  plugins: [new webpack.HotModuleReplacementPlugin()],
  devServer: {
    contentBase: path.resolve(__dirname, './dist'),
    hot: true,
  },
};
```

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

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

const title = 'React with Webpack and Babel';

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

module.hot.accept();
```

Now you can start your app again. Once 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 source code.

Last but not least, create your first React component. In your *src/index.js* file, import a not yet defined App component:

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

```
import App from './App';

const title = 'React with Webpack and Babel';

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

module.hot.accept();
```

Next, create a this new file in your *src/* folder:

```
cd src/
touch App.js
```

And add the following content in it:

```
import React from 'react';

const App = ({ title }) =>
  <div>{title}</div>;

export default App;
```

Congratulations, you have created your first function component and passed props to it. That's it for a minimal React setup with Babel and Webpack. Let me know your thoughts and whether there are things to improve the setup. You can find the repository on GitHub.

Exercises:

- Check out the advanced Webpack setup
  - Put it into your minimal React with Webpack application
  - Confirm your final result with the official advanced React with Webpack setup

This tutorial is part 1 of 3 in the series.

Part 2: How to use ESLint in Webpack
Part 3: How to use ESLint in React

Continue Reading: How to use Prettier in VS Code

Continue Reading: How to use CSS Modules in React?

Continue Reading: How to use SVG Icons as React Components?

Continue Reading: How to use Fonts in React?

Continue Reading: How to use Images in React?

Show Comments

KEEP READING ABOUT GRAPHQL

GETTING STARTED WITH REACT AND PARCEL

The following article is a short guide on how to setup Parcel with React. Parcel is a bundler that got popular because of its zero-configuration setup for JavaScript applications. That's why this...

A MINIMAL APOLLO CLIENT IN REACT EXAMPLE

It's time to get you started with a minimal Apollo Client in React application that can be used as boilerplate project. There will be sections later on where you can use this application as starter...