# How to Webpack 5 - Setup Tutorial

OCTOBER 30, 2020 BY ROBIN WIERUCH - EDIT THIS POST

Follow on Twitter 17k Follow on Facebook

This tutorial is part 2 of 3 in this series.

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

Webpack is a JavaScript bundler for your web application. In the past, you had to link JavaScript files manually in HTML files. Nowadays, Webpack takes care about it. In this tutorial, I want to walk you through a minimal Webpack setup for a JavaScript + HTML application. Afterward, you should be able to extend Webpacks features, advance your JavaScript application, or structure the HTML of your website yourself.

## PROJECT FILE/FOLDER STRUCTURE

We will start with creating a distribution folder. The folder will be used to serve your application from a local or remote web server later on. Serving the app makes it possible to view it in the browser or to host it on an external hosting server which makes it accessible for everyone via HTTP. After all, the distribution folder will be everything you need to publish your web application. To get you started, create the distribution folder and a HTML file as entry point to your application from your project's root directory on the command line:

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

The naming of the folder is up to you. Often you will find it named as *dist*, *public*, or *build*. Also you can give the HTML file a name yourself. However, calling it *index.html* is a widely used naming convention again. So you may want to keep it this way for now. As next step, give the created *dist/index.html* file the following content:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello Webpack</title>
  </head>
  <body>
    <div>
      <h1>Hello Webpack</h1>
    </div>
  </body>
</html>
```

The file contains all the HTML to display your website in a browser. No JavaScript is involved yet. In order to get you started with JavaScript, put the following logging in a *src/index.js* file:

```
console.log('Hello Webpack Project.');
```

As for now, you should have a *dist/* and *src/* folder for your project:

```
- dist/
-- index.html
- src/
-- index.js
- package.json
```

You may wonder: Why don't we put all files into one folder? That's because in the next steps we will make sure that all JavaScript source code files from the *src/* folder will get bundled into a single JavaScript file which will be placed automatically into the *dist/* folder. By keeping this separation of folders, you can be sure that everything you need to take your application to production on a web server sits in the *dist/* folder and everything to implement your application sits in the *src/* folder.

Now, the most straightforward approach to link JavaScript to your HTML file would be the following way:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello Webpack</title>
  </head>
  <body>
    <div>
      <h1>Hello Webpack</h1>
    </div>
    <script src="../src/index.js"></script>
  </body>
</html>
```

However, this may become a tedious task over time, because you have to keep track of what JavaScript you link to your HTML file. That's why we are using Webpack to generate **one JavaScript bundle** from all our source code in the *src/* folder, which will be automatically put into your *dist/* folder afterward. Then, it can be used in our entry point HTML file the following way:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello Webpack</title>
  </head>
  <body>
    <div>
      <h1>Hello Webpack</h1>
    </div>
    <script src="./bundle.js"></script>
  </body>
</html>
```

Therefore, we will need to set up Webpack to bundle our source code for us. Let's get into Webpack.

## WEBPACK SETUP

You will use Webpack to bundle your JavaScript source code, but also to add advanced features to *build* your project with further build steps later on. Moreover, you will use the Webpack Dev Server to serve your project in a local environment with a local web server for development purposes. Last but not least, you need the Webpack CLI as well. Let's install all three dependencies as libraries (node packages) by using npm for your project. From your project's root directory, type the following command to install all libraries as development dependencies:

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

*Note: Development dependencies (short: dev dependencies, indicated with `--save-dev`) are not bundled in your production code later on. This way, you keep the bundle for your production code lightweight. For instance, if you want to test your source code later on, you can do this in your local development mode with all the testing libraries installed as dev dependencies, but later on these libraries will not be included in the actual application for production. Only the dependencies for running the source code are needed as production ready dependencies. They can be installed without the `--save-dev` flag.*

As a side effect, you should find a *node_modules/* folder with all your third party dependencies. Usually you don't need to worry about them in there. Also the dependencies will be listed in the *package.json* file. Check it yourself to see the dependencies with their version numbers. By now, your folder structure should look like the following:

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

In your *package.json* file, change the start script to the following script for using the Webpack Dev Server:

```
{
  ...
  "scripts": {
    "start": "webpack serve --mode development",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  ...
}
```

You can run your *npm start* script on the command line again, however, this time you have Webpack as local web server to serve your files in development mode. Navigate to your *dist/index.html* file to see its output in the browser. You may also see in the developer tools of your browser that our *bundle.js* cannot be found. That's because we didn't tell Webpack yet to generate it for us. Next, let's see how we can bundle our source code files from the *src/* folder into the *dist/* folder with Webpack. Therefore, add the following addition to your npm start script in your *package.json* file:

```
{
  ...
  "scripts": {
    "start": "webpack serve --config ./webpack.config.js --mode development",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  ...
}
```

The script defines that you want to use the Webpack Dev Server with a configuration file called *webpack.config.js*. Let's create the required *webpack.config.js* file in your project's root directory:

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

Your folder structure should look like the following now:

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

Finish the Webpack setup by providing the following configuration for the new *webpack.config.js* file:

```
module.exports = {
  // 1
  entry: './src/index.js',
  // 2
  output: {
    path: '/dist',
    filename: 'bundle.js'
  },
  // 3
  devServer: {
    contentBase: './dist'
  }
};
```

The Webpack configuration file gives the following instructions:

- (1) Use the *src/index.js* file as entry point to bundle it. If the *src/index.js* file imports other JavaScript files, bundle them as well.
- (2) The bundled source code files shall result in a *bundle.js* file in the */dist* folder.
- (3) The */dist* folder will be used to serve our application to the browser.

In order to be more correct about these paths across operating systems, we should resolve them properly:

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

module.exports = {
  entry: path.resolve(__dirname, './src/index.js'),
  output: {
    path: path.resolve(__dirname, './dist'),
    filename: 'bundle.js',
  },
  devServer: {
    contentBase: path.resolve(__dirname, './dist'),
  },
};
```

After all, you should be able to start the Webpack Dev Server again. Before, make sure that your *dist/index.html* includes the generated *bundle.js* file as already shown previously:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello Webpack</title>
  </head>
  <body>
    <div>
      <h1>Hello Webpack</h1>
    </div>
    <script src="./bundle.js"></script>
  </body>
</html>
```

After starting the application with `npm start`, open the application in your browser. Note: The URL for the browser should be visible on the command line too. Once you opened your application in the browser, you should be able to see the output of your *dist/index.html* file and the logging from your *src/index.js* file. If more JavaScript source code files get imported from the *src/index.js* file, they will be bundled in the *bundle.js* by Webpack as well.

**Exercises:**

- Confirm your source code for the last section
- Add a button element to your *dist/index.html* file
- Add a click handler for this new button in your *src/index.js* file

This tutorial is part 2 of 3 in this series.

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

Show Comments

## KEEP READING ABOUT BABEL

### HOW TO USE IMAGES WITH WEBPACK 5 - SETUP TUTORIAL

In this tutorial, you will learn how to set up Webpack to use images as assets for your application. Essentially, there is not much in Webpack to include your desired images for your web application...

### HOW TO ADVANCED WEBPACK 5 - SETUP TUTORIAL

The previous tutorials have shown you how to set up a basic web application with Webpack 5. So far, Webpack is only used to bundle all your JavaScript files, to transpile new JavaScript features via...

TAKE PART

NEVER MISS AN ARTICLE ABOUT WEB DEVELOPMENT AND JAVASCRIPT.

✔ Join 50.000+ Developers

✔ Learn Web Development with JavaScript

✔ Tips and Tricks

✔ Access Tutorials, eBooks and Courses

✔ Personal Development as a Software Engineer

Your email address SUBSCRIBE

View our Privacy Policy.

**PORTFOLIO**

Online Courses

Open Source

Tutorials

**ABOUT**

About me

What I use

How to work with me

How to support me

Contact Me Privacy & Terms