How to JavaScript - Setup Tutorial

OCTOBER 29, 2020 BY ROBIN WIERUCH - EDIT THIS POST

This tutorial is part 1 of 2 in the 'Backend Setup'-series.

Part 2: The minimal Node.js with Babel Setup

This tutorial is part 1 of 3 in the 'Frontend Setup'-series.

Part 2: How to set up Webpack 5
Part 3: How to set up Webpack 5 with Babel

In this tutorial, I want to show you how to set up a JavaScript project from scratch.

Afterward, you can continue by advancing the project to a frontend (e.g. React.js) or backend (e.g. Node.js with Express) application. In case you decide to go with the frontend application, you will set it up the modern way by not linking JavaScript files in HTML files, but by using a project bundler that will automate this process for you. It's one of the most popular ways to start with a JavaScript project nowadays.

Let's start with our JavaScript application. For any new JavaScript project, there has to be a folder to allocate the project's configuration but most importantly all of its source code. This folder usually sits in another folder where all your other JavaScript projects can be found. That's at least how I do it for my projects. In order to get started with your new project, create its project folder on the command line or via your favorite folder/file explorer (e.g. MacOS finder, Windows explorer, editor/IDE side bar) and navigate into it. In this tutorial, we will do it on the command line:

```
mkdir my-project
cd my-project
```

Now you have got the project's folder. In the next step, we will initialize the project as npm project which comes with two benefits for the project: First, you can install libraries (node packages) from npm to your project. And second, you can add npm scripts to start, test, or deploy your project in a later stage of your project's lifecycle. Before you can use npm on the command line, you have to install Node.js which comes with npm. Afterward, you can initialize the npm project on the command line:

```
npm init -y
```

By giving it the `-y` shorthand flag, you are telling npm that it should take all the defaults. If you leave the flag out, you are in charge to specify the information about your project manually. You can try it yourself with a second project.

Now, since you have used npm to initialize it, your project should have a *package.json* file which should be filled with your defaults. If you want to change your defaults, you can see and change them with the following commands on the command line:

```
npm config list

npm set init.author.name "<Your Name>"
npm set init.author.email "you@example.com"
npm set init.author.url "example.com"
npm set init.license "MIT"
```

After setting up your npm project, you can install libraries (alias: node packages) to your project with npm (node package manager). Once you have installed your first library via npm to your project, it should show up in your *package.json* file as dependency. Also you will see a *node_modules/* folder showing up, where all your installed libraries will be kept as actual source code. More about this later.

Next, on the command line or in your editor/IDE/explorer, create a *src/* folder for your project's source code. In this folder, create a *src/index.js* file as an entry point to your project:

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

The name is based on a Node.js naming convention. It's up to you to follow it or to come up with a naming yourself.

Now it's time to write your first JavaScript. To begin, introduce a logging statement in the *src/index.js* file to make sure your setup is running:

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

As you navigate on the command line again, you can run this file with Node.js from your project's root folder:

```
node src/index.js
```

The logging should appear in the command line after the script is executed. Next, move this script into your *package.json* file, because that's where all your project's scripts (start, test, deploy) will end up eventually. As mentioned earlier, it's one benefit of an npm project to define these npm scripts in the *package.json* file:

```
{
  ...
  "scripts": {
    "start": "node src/index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  }
```

```
  }
  ...
}
```

On the command line, run the script again, but this time with a `npm start`. Every time you change the underlying start script in the *package.json* file's npm scripts, you only need to type `npm start` on the command line without the specifics of the underlying script.

Congratulations, you have set up your first JavaScript project with Node and npm. From here it's up to you to advance it as backend application, frontend application or anything else. If you are into open source, you can also open source this project as library (node package). Therefore, follow the backend application series to find out how to open source it.

This tutorial is part 1 of 2 in the 'Backend Setup'-series.

Part 2: The minimal Node.js with Babel Setup

This tutorial is part 1 of 3 in the 'Frontend Setup'-series.

Part 2: How to set up Webpack 5
Part 3: How to set up Webpack 5 with Babel

KEEP READING ABOUT STARTER

HOW TO GET STARTED WITH DENO TUTORIAL

Deno is a new runtime for JavaScript and TypeScript. If this doesn't tell you much and you don't know what to expect, then take this statement as secondary introduction: Ryan Dahl, inventor of Node.js...

HOW TO SETUP EXPRESS.JS IN NODE.JS

Express.js is the most popular choice when it comes to building web applications with Node.js. However, when saying web applications with Node.js, it's often not for anything visible in the browser...

TAKE PART

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