The minimal Node.js with Babel Setup

APRIL 22, 2020 BY ROBIN WIERUCH - EDIT THIS POST

This tutorial is part 2 of 2 in the series.

Part 1: How to set up a modern JavaScript project

I have always been of the understanding there are no common sense rules about how to create a minimal Node.js application with Babel. In my search for these answers, it seemed that every tutorial I came across showed something different. As a result, I wanted to streamline this project setup for my readers and myself, so I developed a common approach for Node.js applications with Babel.

I strongly believe it is a good foundation for learning JavaScript on the command line, building sophisticated Node.js projects on top of it, releasing it as node package (library) on npm as an open source project, or to build a RESTful or GraphQL server on top of it. The final project you are going to implement here can be found in this GitHub repository.

TABLE OF CONTENTS

NODE.JS WITH NODEMON

So far, you are able to start your application by running the npm start script. The only remaining concern is that you have to start the script every time you want to try your source code. You can change this behavior with an always-running node process. To remedy this, install the commonly used nodemon library on the command line as development dependency to your project.

```
npm install nodemon --save-dev
```

Next, exchange node with nodemon in your npm start script:

```
{
  ...
  "main": "index.js",
  "scripts": {
    "start": "nodemon src/index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  ...
}
```

When you run your application with `npm start` from the command line, it should keep running. The best part is that the script will execute again once you change the source code. Try adjusting your source code in the *src/index.js* file and see what happens in your command line.

```
console.log('Hello ever running Node.js project.');
```

This little adjustment to the environment gives developers a powerful tool, because the node process executes again once you change your code. If you introduce a bug, you

node process executes again once you change your code. If you introduce a bug, you will see a stack trace in the command line, and the script runs again without any flaws.

NODE.JS WITH BABEL

You should be able to develop a Node.js application by now, but there is more to setting up a sophisticated Node.js project that is capable of using recent JavaScript language features (ECMAScript) that are not included in the recent Node.js versions. That's where Babel becomes useful. You can install it from the command line for your project's development dependencies.

```
npm install @babel/core @babel/node --save-dev
```

Next, add it to your npm start script:

```
{
  ...
  "main": "index.js",
  "scripts": {
    "start": "nodemon --exec babel-node src/index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  ...
}
```

Nothing should change when you run the application again, though that's just the surface. Under the hood, Babel transpiles your code to vanilla JavaScript. When you use an upcoming JavaScript language feature, which hasn't been introduced in Node.js, you can still use the feature in your source code. Babel makes sure that Node.js understands it. However, there is still one crucial step to include upcoming language features with Babel. You can activate different upcoming JavaScript features by adding them as presets to Babel. Let's add the most common used Babel preset to your application:

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

Now, in the project's root folder, create a *.babelrc* file in the command line:

```
touch .babelrc
```

In this configuration file for Babel, you can include the recently installed dependency for unlocking the upcoming JavaScript language features.

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

Now you can include upcoming JavaScript features in your *src/index.js* file. If you run into problems because your desired feature is not working, check whether there exists a dedicated Babel preset for it.

ENVIRONMENT VARIABLES IN NODE.JS

It is important to set data like private API keys and user credentials like password, username, and email as environmental variables, but without exposing them in the source code. For this, we put environmental variables in a dedicated file that is safe from external access. The *.env* file lets you set Node.js environment variables as accessible in your project's source code. On the command line, in your project's root folder, create a *.env* file:

```
touch .env
```

Now you can place any key value pair that you don't want in your source code in this new file.

```
MY_SECRET=mysupersecretpassword
```

dotenv is another helpful library to make environmental variables accessible in the source code. First, install it on the command line as a normal dependency:

```
npm install dotenv --save
```

Second, import it into your *src/index.js* file to initialize it. The environment variable from your *.env* file is now accessible in your source code.

```
import 'dotenv/config';

console.log('Hello Node.js project.');
```

```
console.log(process.env.MY_SECRET);
```

Start the npm script again, and you should see the environmental variable in the command line. Now you are able to store sensitive data separate from the source code.

Now, consider the following code for your *src/index.js* file, where a function is imported from another file from this project.

```
import saySomething from './my-other-file.js'

import 'dotenv/config';
```

If you use an environment variable in your *src/my-other-file.js*, it is undefined because the initialization of the dotenv package happens after the import in your *src/index.js* file. To fix it, put the dotenv initialization before your local file imports:

```
import 'dotenv/config';

import saySomething from './my-other-file.js'
```

That's a basic understanding of Node.js environment variables. They should be used to keep sensitive data secure in JavaScript applications, but shouldn't be shared on public GitHub repositories when using git.

Exercises:

- Confirm your source code.
- Ask yourself:
  - What's `npm init` doing when you setup your Node.js project?
  - What benefit is Nodemon giving us?
  - Why do we need Babel?
  - Why do we need Environment Variables?

. . .

This guide has shown you how to create a Node.js project from scratch, and how you can introduce upcoming JavaScript features in your Node.js environment using Babel. You have seen how npm scripts are used to start, test, and deploy applications, and how environment variables secure sensitive data like private API keys and user credentials. The finished product is a node package that can be open sourced on npm, another rewarding aspect of working with the Node.js ecosystem.

This tutorial is part 1 of 2 in the series.

Part 2: How to setup Express.js in Node.js

This tutorial is part 1 of 2 in the series.

Part 2: GraphQL Server Tutorial with Apollo Server and Express

This tutorial is part 1 of 2 in the series.

Part 2: Node Testing Setup with Mocha and Chai

This tutorial is part 1 of 2 in the series.

Part 2: How to publish a npm package?

Show Comments

KEEP READING ABOUT DOCKER›

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 DOCKER WITH NODE.JS

Just recently I had to use Docker for my Node.js web application development. Here I want to give you a brief walkthrough on how to achieve it. First of all, we need a Node.js application. Either take...

THE ROAD TO REACT

Learn React by building real world applications. No setup configuration. No tooling. Plain React in 200+ pages of learning material. Learn React like **50.000+ readers**.

GET THE BOOK >

Get it on Amazon.

TAKE PART

NEVER MISS AN ARTICLE ABOUT WEB DEVELOPMENT AND JAVASCRIPT.

- ✔ Join 50.000+ Developers
- ✔ Learn Web Development
- ✔ Learn JavaScript