JANUARY 23, 2020 / #JAVASCRIPT

How to enable ES6 (and beyond) syntax with Node and Express

Jonathan Cunanan

Have you ever tried to write front-end apps using ES6 syntax, but then when you decided to learn `om` and `export default`? If so, you came to the right place! This is step by step guide on how to configure your dev and prod environments, setup scripts, and as a bonus we'll learn how to add tests!

Table of Contents / Summary of topics

How does it work? A high level view of what we need

To enable a front-end development-like experience while developing back-end apps, here's a high level view of the processes happening to your project.

Code Transpiler from ES6+ to ES5

We need a package that translates ES6 and above syntax to ES5 in Node.js. This is where the package called *babel* shines.

Babel takes a js file, converts the code in it, and outputs into a new file.

Script that removes files

Whenever we change something in our code, we feed it to the transpiler, and it outputs a fresh copy every-time. That's why we need a script that removes files before the fresh transpiled copy enters. And for that, there's an existing package called rimraf. Rimraf deletes files. We'll demonstrate that later.

Watcher of file changes

When coding in Node.js, automatic restart of our server doesn't come out of the box just like when doing a project made on-top of create-react-app or vue-cli. That's why we'll install a package called nodemon, that executes something whenever we change a file in our code. We can leverage nodemon to restart our server every-time a file is changed.

So that's the high-level view of how it works under the hood. With that, let's start on how should we setup or project.

Prerequisites

recommend installing their latest LTS or current stable version. You can install it via Node.js Source or NVM (Node Version Manager)

2. Basic knowledge of terminal commands. Most of the commands are in the tutorial anyway so you don't have to worry about them.
3. Make sure you have your terminal open and your favorite text editor installed.

That's it, we're good to go!

Installing Express

Using the Express generator, we will create a new project with generated code, move some files, and convert some code to ES6 syntax. We need to convert it at this early stage because we need a way to verify if our ES6 code works.

Project Setup

Run this command in your terminal. You can name `your-project-name` with the name you like. `--no-view` flag means that we won't be using any templating engine such as handlebars, ejs, or pug, for our skeleton Express app.

Windows Powershell and Linux terminals, use:

```
cd your-project-name
```

Next, open the text editor you like. For me, I just use VSCode so I just have my terminal and text editor open at the same time. But you can use any text editor you want.

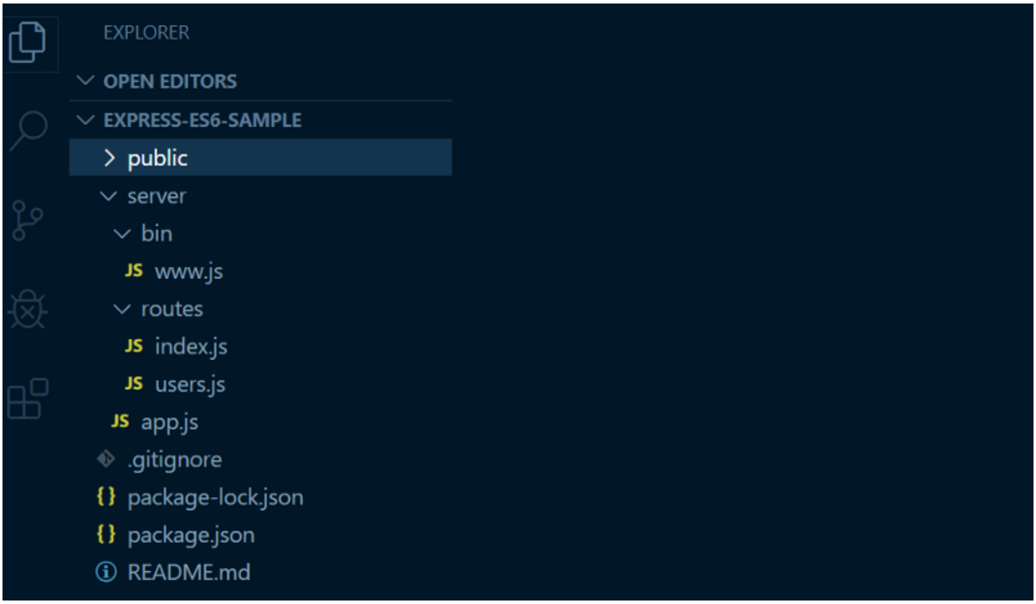
Installing Packages and Moving and Deleting Files

After we have the generated project ready, we need to `install` the dependencies and move some folders. Run this command to install Express and other packages.

npm install

While you're waiting for the dependencies to install, follow these steps.

- create a `server/` folder
- Put `bin/` , `app.js` , and `routes/` inside the `server/` folder.
- Rename `www` , found in `bin` to `www.js`
- Leave `public/` folder at your project root.


EXPLORER
OPEN EDITORS
EXPRESS-ES6-SAMPLE
```
> public
v server
  v bin
    www.js
  v routes
    index.js
    users.js
  app.js
.gitignore
package-lock.json
package.json
README.md
```

This is how our file structure looks like. `public/` folder is at the root, and all the `.js` files are inside `server/` folder.

Now, because we modified the file structure, our start server script won't work. But we'll fix it along the way.

just post the code here and feel free to copy and paste it.

Code for `bin/www.js`:

Now, because we modified the file structure, our start server script won't work. Here's what we'll do to fix it. On your package.json file, rename start script to `server` found in a JSON Object called `"scripts"`

```
// package.json
{
  "name": "your-project-name",
  // ....other details
  "scripts": {
    "server": "node ./server/bin/www"
  }
}
```

You'll see that we changed the file path from `./bin/www` to `./server/bin/www` because we moved files to `server/`. We'll use start script later on.

Try it! Try running the server by typing `npm run server` on your terminal, and go to `localhost:3000` on your browser.

just post the code here and feel free to copy and paste it.

Code for `bin/www.js`:

```
// bin/www.js
/**
 * Module dependencies.
 */
import app from '../app';
import debugLib from 'debug';
import http from 'http';
const debug = debugLib('your-project-name:server');
// ..generated code below.
```

Almost all of our modifications are only at the top and bottom of the files. We are leaving other generated code as is.

Code for `routes/index.js` and `routes/users.js`:

```
// routes/index.js and users.js
import express from 'express';
var router = express.Router();
// ..stuff below
export default router;
```

```
// app.js
import express from 'express';
import path from 'path';
import cookieParser from 'cookie-parser';
import logger from 'morgan';
import indexRouter from './routes/index';
import usersRouter from './routes/users';
var app = express();
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, '../public')));
app.use('/', indexRouter);
app.use('/users', usersRouter);
export default app;
```

In `app.js` , because we left `public/` at the project root , we need to change the Express static path one folder up. Notice that the path `'public'` became `'../public'` .

`app.use(express.static(path.join(__dirname, '../public')));`

Okay we're done with converting the code! Let's setup our scripts now.

production environments, they have a different configuration. (Almost identical, you'll see later) That's why we need to compose our scripts so we can use them without repeatedly typing the same stuff all over again.

Install `npm-run-all`

Since some terminal commands won't work on windows cmd, we need to install a package called `npm-run-all` so this script will work for any environment. Run this command in your terminal project root.

```
npm install --save npm-run-all
```

Install babel, nodemon, and rimraf

Babel is modern JavaScript transpiler. A transpiler means your modern JavaScript code will be transformed to an older format that Node.js can understand. Run this command in your terminal project root. We will be using the latest version of babel (Babel 7+).

Note that Nodemon is our file watcher and Rimraf is our file remover packages.

```
npm install --save @babel/core @babel/cli @babel/preset-
```

Before babel starts converting code, we need to tell it which parts of the code to translate. Note that there are a lots of configuration available, because babel can convert a lot of JS Syntaxes for every different kinds of purpose. Luckily we don't need to think about that because there's an available default for that. We are using default config called as preset-env (the one we installed earlier) in our package.json file to tell Babel in which format we are transpiling the code.

Inside your `package.json` file, create a `"babel"` object and put this setting.

```
// package.json
{
  // .. contents above
  "babel": {
    "presets": ["@babel/preset-env"]
  },
}
```

After this setup we are now ready to test if babel really converts code. Add a script named transpile in your `package.json`:

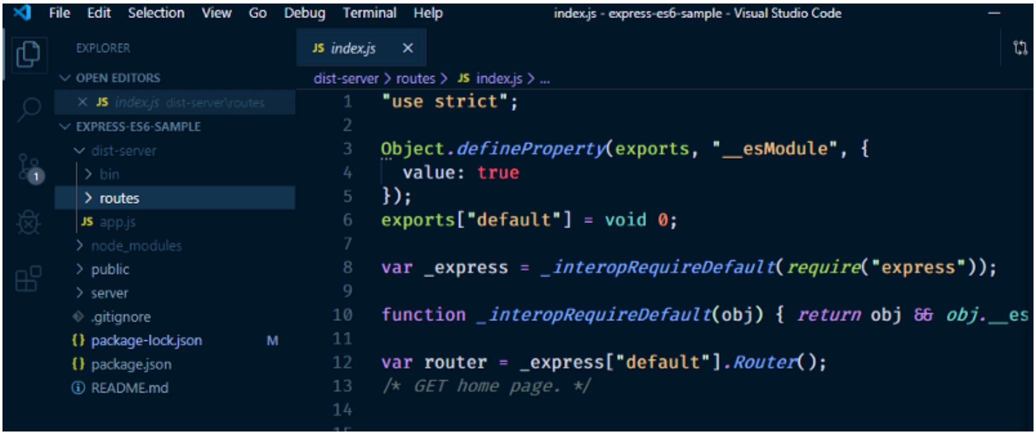
```
      "start": "node ./server/bin/www",
      "transpile": "babel ./server --out-dir dist-server",
  }
```

Now what happened here? First we need to run the cli command `babel`, specify the files to convert, in this case, the files in `server/` and put the transpiled contents in a different folder called `dist-server` in our project root.

You can test it by running this command

`npm run transpile`

You'll see a new folder pop up.



```
"use strict";

Object.defineProperty(exports, "__esModule", {
  value: true
});
exports["default"] = void 0;

var _express = _interopRequireDefault(require("express"));

function _interopRequireDefault(obj) { return obj && obj.__es

var router = _express["default"].Router();
/* GET home page. */

```

New folder popped up called dist-server because of the script we ran.

Yay it worked! ✔️ As you can see, there's a folder that has the same folder structure as our server folder but with converted code inside. Pretty cool right? Next step is to run try if our server is running!

Clean script

To have a fresh copy every-time we transpile code into new files, we need a script that removes old files. Add this script to your package.json

```
"scripts": {
  "server": "node ./dist-server/bin/www",
  "transpile": "babel ./server --out-dir dist-server",
  "clean": "rimraf dist-server"
}
```

This npm script that we made means it removes the folder `dist-server/`

```
// scripts
"build": "npm-run-all clean transpile"
```

Running dev script

Now we have a build script, we need to run our dev server. We'll add a script called `dev` in our package.json. This takes care of setting our Node Environment to "development", removing old transpiled code, and replacing it with a new one.

```
"scripts": {
  "build": "npm-run-all clean transpile"
  "server": "node ./dist-server/bin/www",
  "dev": "NODE_ENV=development npm-run-all build server",
  "transpile": "babel ./server --out-dir dist-server",
  "clean": "rimraf dist-server"
}
```

Note here that we've changed again the file we are running on our server script. We're running the file-path with the transpiled code, found in `dist-server/`.

development, we have a `prod` script that sets it to "production." We use this configuration when we are deploying. (Heroku, AWS, DigitalOcean, etc..) We're now adding again our start script and prod script in our package.json again.

```
"scripts": {
  "start": "npm run prod"
  "build": "npm-run-all clean transpile"
  "server": "node ./dist-server/bin/www",
  "dev": "NODE_ENV=development npm-run-all build server",
  "prod": "NODE_ENV=production npm-run-all build server",
  "transpile": "babel ./server --out-dir dist-server",
  "clean": "rimraf dist-server"
}
```

We set `start` script default to prod because start script is being used always by deployment platforms like AWS or Heroku to start a server.

Try either by running `npm start` or `npm run prod`.

```
// package.json
...
"nodemonConfig": {
  "exec": "npm run dev",
  // ... other scripts
  "watch:dev": "nodemon"
}
```

How about auto-restarting the server whenever a file change?

One final script, in order to complete our development setup. We need to add a file watcher script that runs a command whenever a change is made in a file. Add a JSON Object named “nodemonConfig” in your package.json. This is where we store what we tell the watcher what to do when a file changes.

Also, add a script called `watch:dev` in your package.json

```
  // package.json
  ...
  "nodemonConfig": {
    "exec": "npm run dev",
    "watch": ["server/*", "public/*"],
    "ignore": ["**/__tests__/**", "*.test.js", "*.spec.js"]
  },
  "scripts": {
    // ... other scripts
    "watch:dev": "nodemon"
  }
```

- Which command to run whenever a file changes, in our case `npm run dev`
- What folders and files to watch
- And which files to ignore

More about configuration of nodemon here.

Now that we have our file watcher, you can now just run `npm run watch:dev`, code, and save your file. and whenever you go to `localhost:3000`, you'll see the changes. Try it out!

Bonus: Add tests!

To add tests in our project, simply install Jest from npm, add a few config, and add a script called `test` in our package.json

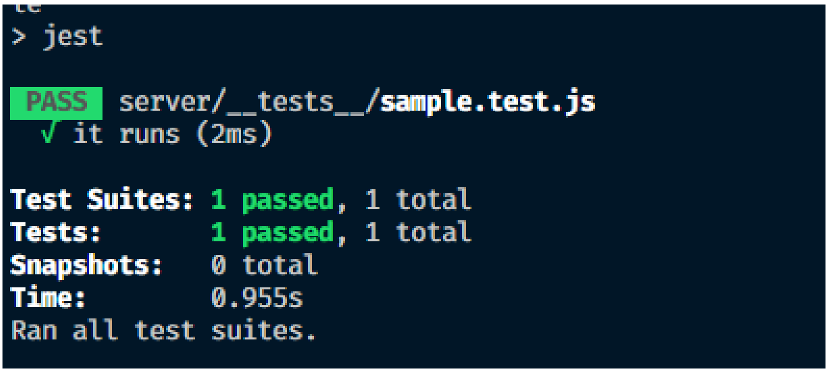
```
npm i -D jest
```

Add an object called "jest", and a test script in your package.json

```
// package.json
...
"jest": {
  "testEnvironment": "node"
},
```

Try it out, make a file sample.test.js, write any tests, and run the script!

```
npm run test
```



```
> jest

 PASS  server/__tests__/sample.test.js
  √ it runs (2ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.955s
Ran all test suites.
```

Sample Screenshot of running npm run test.

TL;DR

Here are the simplified steps for how to enable ES6 in Node.js. I'll also include the repo so you can copy and inspect the whole code.

- Make a new project using `express your-project-name` terminal command.

forget to rename `bin/www` to `www.js`

- Install all the dependencies and devDependencies

```
npm i npm-run-all @babel/cli @babel/core @babel/preset-env noc
npm i -D jest
```

- Add these scripts to your package.json

```
"scripts": {
  "start": "npm run prod",
  "build": "npm-run-all clean transpile",
  "server": "node ./dist-server/bin/www",
  "dev": "NODE_ENV=development npm-run-all build server",
  "prod": "NODE_ENV=production npm-run-all build server",
  "transpile": "babel ./server --out-dir dist-server",
  "clean": "rimraf dist-server",
  "watch:dev": "nodemon",
  "test": "jest"
}
```

- Put configurations for babel, nodemon, and jest in your package.json

```
    "watch": [ "server/*", "public/*" ],
    "ignore": [ "**/__tests__/**", "*.test.js", "*.spec.js" ]
  },
  "babel": {
    "presets": [ "@babel/preset-env" ]
  },
  "jest": {
    "testEnvironment": "node"
  },
```

- Test your scripts by running `npm run your-script-here`
- You'll see the complete repo at my github

Notes and disclaimers

Note that this setup may not be proved ideal for all situations, specially for big projects. (like 1k files of code). Transpiling step and deleting might slow down your development environment. Plus, ES Modules, is almost coming to node. But, nevertheless, this is a good eductational material to understand how transipiling runs under the hood like when we are developing front-end apps :)

Conclusion

All right! I hope you learned a lot. Thank you for reading this far.

This article is published in freeCodecamp news.

? Twitter - ? freeCodeCamp - ? Portfolio - ⚛ Github

Jonathan Cunanan

Javascript Engineer / React Dev with AWS Experience. Passionate in #100DaysOfCode

Trending Guides

10 to the Power of 0

Git Reset to Remote

R Value in Statistics

What is Economics?

Module Exports

Python VS JavaScript

Model View Controller

React Testing Library

ASCII Table Chart

Data Validation

Recursion

ISO File

ADB

MBR VS GPT

Debounce

OSI Model

HTML Link Code

SDLC

Inductive VS Deductive

JavaScript Empty Array

Best Instagram Post Time

Garbage Collection in Java

Auto-Numbering in Excel

JavaScript Keycode List

JavaScript Reverse Array

How to Screenshot on Mac

How to Reverse Image Search

Ternary Operator JavaScript