NodeJS

EXPRESS 101

In the last lesson, we set the stage by explaining quite a bit of the background information you'll need to really understand what's going on as we start to dive into express. This lesson will actually start you on the project that you'll be completing as you follow the tutorial.

Learning Outcomes

By the end of this lesson, you should be able to do the following:

- Use `express-generator` to generate a basic express site.
- Understand the basic parts of an express project.
- Understand what a Templating Language is and be able to list a couple of popular ones.
- Understand what Middleware is.
- Understand `req`, `res` and `next` in the context of middleware.

Templating Engines

A templating engine is a tool that allows you to insert variables and simple logic into your views. For instance, you could have a header that updates with the actual user's name once they've logged in, something that is not possible with plain HTML. As the lesson mentions, there are several templating languages available for JavaScript. The tutorial uses Pug (formerly known as Jade) which has a bit of a learning curve because it looks and feels dramatically different from regular HTML. If you've ever worked with Ruby on Rails you might be more comfortable with `ejs`, which is *very* similar to `erb` or `hbs` (handlebars), which also looks and feels a lot like HTML, and inserts the dynamic bits inside of double curly brackets.

It's up to you which you choose! If you choose not to use Pug you will still be able to follow the tutorial just fine. Most of the Odin staff prefer ejs or handlebars to Pug simply because we like working with HTML, but in the end, there is nothing wrong with Pug if you like the look of it or want to learn something new.

Middleware

This step of the MDN tutorial mentions middleware, but does not clearly define it. Middleware is a complicated word for a simple concept. A middleware is just a plain JavaScript function that Express will call for you between the time it receives a network request and the time it fires off a response (i.e. it's a function that sits in the *middle*). You will eventually be using several of these functions that will run in a specific sequence for every request.

For example, you might have a logger (that prints details of the request to the console), an authenticator (that checks to see if the user is logged in, or otherwise has permission to access whatever they're requesting) and a static-file server (if the user is requesting a static file then it will send it to them). All of these functions will be called in the order you specify every time there's a request on the way to your `app.get("/")` function.

It is possible and common to write your own middleware functions (you'll be doing that later) so let's take a minute to demystify what they're actually doing. Middleware functions are just plain javascript functions with a specific function signature (that is, it takes a specific set of arguments in a specific order). You've actually already seen it!

The three middleware function arguments are: `req`, `res`, and `next`. Technically, these are just variables, so you could call them anything, but convention (and the express documentation) almost always give them these names.

A middleware function:

```
function(req, res, next) {
  // do stuff!
}
```

When someone visits your site, their web-browser sends a request to your server. Express takes that request and passes it through all of the middleware functions that you have defined and used in your project. Each function is defined with these parameters which might seem familiar to you from the plain Node tutorial that you went through in the 'Getting Started' lesson. Technically, `req` and `res` are *almost* the same here as they are in vanilla Node, but Express enhances them by adding a few useful properties and methods to them.

`req` or `request` is an object that has data about the incoming request such as the exact URL that was visited, any parameters in the URL, the `body` of the request (useful if the user is submitting a form with some data in it) and many other things.

- You can see everything it includes in the express docs.

`res` or `response` is an object that represents the response that Express is going to send back to the user. Typically, you use the information in the `req` to determine what you're going to do with the `res` by calling `res.send()` or another method on the object.

- Check out the documentation for the response object here!

`next` is a function that you see a little less often, but is *very* important to the functioning of your app. If you are writing or using some middleware that does not send a response back to the user's client then you *must* call the `next` function at the end of your middleware function. The next function simply tells express to move to the next middleware in the stack, but if you forget to call it then your app will pause and nothing will happen!

An example middleware

An example middleware

As a quick example, if you wanted to create a simple logging middleware you could write a function like this:

```
const myLogger = function(req, res, next) {
  console.log("Request IP: " + req.ip);
  console.log("Request Method: " + req.method);
  console.log("Request date: " + new Date());

  next(); // THIS IS IMPORTANT!
}

app.use(myLogger)
```

`app.use` is how you load your middleware function into Express so that it knows to use it. If you stick this bit of code in any express application near the beginning of your `app.js` (after the part where you define `app = express()`) then it will write all of those details to your console every time you get a network request. When the logging is complete we call the `next()` function so that our app can continue.

As a final detail, the order that middleware gets executed in your app matters! Middleware functions will always run in the order that they are instantiated using `app.use()`.

Using git

if you choose to use git when completing this tutorial (you should!) then you will want to add a `.gitignore` file to make sure you do not commit/upload your `node_modules` folder to GitHub. `node_modules` is the directory where all of your project's dependencies are installed (it's where the code for express is downloaded) and it can get quite large. References to all of these dependencies are stored in the `package.json` file anyway, so anyone that wants to clone and work on the project simply has to run `npm install` to download and install all those dependencies anyway, so uploading them to GitHub is a waste of time and space.

- This article explains the process. You just need to create a file called `.gitignore` and put `node_modules` on a line inside that file.

Assignment

1. Read this intro article on MDN.
2. Begin the project by following this lesson. Be sure to read everything carefully! There's quite a bit of important information in this article. You only have to do part 2 for now. We will continue where we leave off later.
3. For a little more detail on the nature of middleware read the official documentation here.

View Course

Login to track progress

Next Lesson

Improve this lesson on GitHub

Have a question?

Chat with our friendly Odin community in our Discord chatrooms!

Open Discord

Are you interested in accelerating your web development learning experience?

Get started