Node, NPM and A Million Little Packages

Hunter Jorgensen

Background of Node.js

Node.js and NPM were both created in 2009 (Express.js, which we'll see soon, was created in 2010.) It was created by Ryan Dahl and was managed for a number of years by the Node.js Foundation, before being taken over by the OpenJS Foundation.

Node was created in response to Apache HTTP Server, and had the benefit of being able to handle many concurrent, non-blocking requests at a single time.

Why Node.js?

Node is open-source, works on most platforms and lets us code in JavaScript in a server-side environment.

Node performs very well in straightforward web applications (non-blocking, etc.), but struggles with more complicated tasks.

We can code in JS: minimizes context switching between front and backend

You can also use TypeScript, CoffeeScript, LiveScript, etc. and it can convert in JS.

Lots of great 3rd party libraries through NPM

First Node Server

Let’s create an extremely simple server with built in Node libraries

[Demo]

Unfortunately, Node can’t do much more than this. If we want to do more complex stuff, more easily, we need to bring in backup!

Let’s take a look at NPM and some libraries/frameworks for Node.js

Making Requests to Servers

Before getting too far ahead, it's important to get comfortable making requests without the use of a browser (browsers can only do GET requests.)

- cURL (client/URL): if you're comfortable with the command line, you can make do a curl command:

  -
    ```
    curl -X POST --header "Content-Type: application/json" -d "{\"value\":\"node JS\"}" http://localhost:3000/test
    ```
- You can also use a browser plugin (I recommend Postman for Chrome and Rested for Firefox), which I find a bit easier to use (and will use throughout these demos)

Hello (again), NPM

Before we get started, we’ll want to create a new npm package:

```
npm init
```

(You can just press enter on all of the questions, or answer them as you please).

npm init basically sets us up *package.json*, which manages all of the libraries we’ll be importing and can do simple commands (we used it a lot when we’ve been using for both of our projects.)

You might also see a *package-lock.json* file. This manages the dependencies of our dependencies, and is a relatively new addition to NPM.

All Aboard the Express.js

Express is one of the most popular Node libraries.

It allows us write handlers for different HTTP verbs (POST, GET, etc.)

Render server side (which we won’t do here)

Add middleware, to do common request interception (to check for valid requests, cookies, security, etc.)

Adding Express

Now we can install Express using npm:

```
npm install -s express
```

The -s here means save the library and add it to the package.json (this way if other people download your code or your put it in Heroku, it’ll know how to set up everything up exactly as you want it.)

Express

Now let’s go back to our first code and update it with express:

Demo

Express is built upon the Node HTTP module, but is abstracting out a lot of logic so we can do more interesting and powerful things.

Tip: Nodemon

It can get a little tiring to constantly start and restart node, but there are tools that will make this easier.

```
npm install -g nodemon
```

Then you can run **nodemon server.js**, modify the code and watch it update much faster!

Modules

In React, we so the more modern **import X from “Y”;** syntax. This is a relatively new addition to JavaScript, and can only be used with new versions of Node.js, so you may often see the older version: **const x = require(‘y’)**;

A module is a JavaScript library or file that you can import elsewhere, which is done using **require(...)**, for example Express is a module that we’re importing.

In the previous example, we invoked the required(...) function, specifying the module name (this could also be a relative file path), and using the returned object to create an Express application (similar to creating a new object out of a class in Java, for instance.)

Modules

Demo

Alternate helper.js

The second approach is more common!

export is a shortcut to module.exports (Node.js does some behind the scenes on initializing it)

HTTP Verbs

Express allows us to different verbs to specify what can of HTTP action we're looking for on a specific endpoint. For instance, we can cleanly specify different behavior on for a GET, PUT, etc. request on a single endpoint:

Demo

Some import app methods are **get()**, **put()**, **post()**, **delete()**, **all()** (all is when you when to capture every HTTP verb) and **use()** (which is used to handle every request regardless of the URL path.)

Note: you can also always change the URL path (we'll see this more later)

Express Order Matters

One of the tricky things about Express, is that the order of your code matters. For instance, if you have multiple requests to identical URL paths with the same HTTP action, the first one will take precedence:

Demo

UNLESS you use the next method:

Demo

We’ll be using this more in the next module when we want to do things like validation (next is mainly used to intercept a request for whatever reason)

Express Order Matters: Middleware

In Node/Express, middleware is any logic that happens between your server receiving your request and the meat of your own code.

For instance, you'll often see the following lines in our code:

```
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
```

This means that any information that is passed through this parts of the code is modified by this logic. In the case of the above middleware, it is simply parsing the request object for us and turning it into an easy to use object.

Organizing Express into Modules

As our application grows, we will want to organize our node backend into a way that makes it easy to manage. Typically, similar features will live together in the same file.

Express has a function called Routers that make this extremely simple to implement.

Demo

Putting the M and C into MVC

A note on organization: any of the router logic in Node/Express is typically considered part of the controller and should be separated into its own unique *controller* folder.

Likewise, if model (temporary, or in a database) should be organized into a *model* folder.

If your app is big enough, you can further organize helper code into smaller feature directories.

Query Params

Query Params are usually optional parts of a URL that are represented by key/value mappings. ExL

- Amazon.com: https://www.amazon.com/s?k=bananas
- Google Search: https://www.google.com/search?q=where+to+buy+bananas

If you actually go to these websites and make these searches, what do you think the other query params do?

Would these URL's still work if you removed the query params?

Query Params in Node.js

We can access query params by doing req.query.*attribute* (where attribute is the key that you're looking for.)

Demo

http://localhost:8000/food/exists?keyword=bread

http://localhost:8000/food/exists?keyword=banana

Path Params

Path parameters are required parts of a URL to work, and URLs are designed around them. Example:

- https://www.amazon.com/dp/B07885W5GF

(dp stands for detail page)

- https://www.redfin.com/WA/Seattle/5301-1st-Ave-NW-98107/home/298819

What is the relationship of the path params to the other parts of the URL?

Path Params in Node

Path params can be accessed by declaring them in the URL in Node, and then accessing them through the req.params.*attribute*

Demo

Side note: see how we can also send back an error response if the response is bad?

http://localhost:8000/food/bread

http://localhost:8000/food/banana

Request Body

In the previous example, we created a simple local database (that would totally get overwritten!

Demo

Request w/ Postman or Rested: Content-type: application/json

What happens if you don’t include all the parts of the body? (You may need to verify data on the backend, unless you don’t care!)

Debugging Node

While you can always through a bunch of console.logs at your code, node has a pretty simple debugger. Do debug, add the *debugger* line somewhere in your code. Then you can use the following commands to navigate your code:

First you do **node inspect server.js,** then

- **next**: go to the next line
- **cont**: continue code execution until the next breakpoint, if any
- **repl**: enter interactive mode, so you can see what variables look like

We can try it out on the code we just ran!

Node also has fully featured debuggers on tools like WebStorm, Intellij, VSCode, etc.

Lab: Create Put/Delete Operations For Food App

Using the code we started in class, implement both PUT and DELETE functions.

- Typically, PUT and DELETE only require a single unique ID/key to figure out what needs to be deleted. What would that key be in our food app?
- How will you design your URL?

Submit this code to the TA’s for 1 bonus point on the next mini assignment