Running a Node Server

First question you might have: **What is a Server**?

Great question! It sounds complicated, however, a server is simply a computer that provides data to other computers!

When you go to a URL, such as *www.google.com*, you are directed to a server from which you are requesting a website. The website should respond with an HTML page that your web-browser can view!

We can spin up a server in Node.js quite simply using the built-in `http` module:

```
const http = require('http');

const server = http.createServer(function(request, response) {
  response.statusCode = 200;
  response.setHeader('Content-Type', 'text/plain');
  response.end('Hello World');
});

server.listen({ port: 3000, host: 'localhost' }, function() {
  console.log('Server is running!');
});
```

Take this code, save it in a file somewhere called `index.js` .

Next, run `node index` from a terminal located in the directory. You should see the message `"Server is running!"` logged.

Finally, you can go in your browser and type `http://localhost:3000/` in the browser bar to see the response `"Hello World"` in text. Cool!

We can do better than that though! Let's break this down a bit.

> When you start the server it will continue to run, awaiting new requests. If you want to quit the process you can hit `CTRL + C` in your terminal. You'll need to restart the server if you want to make any changes. Alternatively you can use nodemon to restart the server automatically if there are any changes to your code!

Creating the Server

```
const server = http.createServer(function(request, response) {
  response.statusCode = 200;
  response.setHeader('Content-Type', 'text/plain');
  response.end('Hello World');
});
```

Alright, so it looks like this code is creating a server. What else is going on here?

The `function` in the above code is run anytime a new machine connects to our server. When a machine connects to our server, it said to make a **request**. The `request` object will contain all sorts of information about the connecting machine and what data or action is being requested.

> If you're curious what your browser is saying about you, go ahead and `console.log(request.headers)` to see what a server sees when you connect to it! You'll see your connection information and localhost cookies

The **response** is what the server sends back to the machine that made the request (the requesting machine is commonly referred to as the **client**).

The first thing we're doing is setting a `statusCode` :

```
response.statusCode = 200;
```

This status code means "OK", indicating a successful HTTP request. The server sends this back to the client so they know everything is working the way it ought to. Any status code in the 200 range indicates a type of successful request.

> You can find all full list of HTTP status codes and their descriptions on Wikipedia.

Next we're setting a header:

```js
response.setHeader('Content-Type', 'text/plain');
```

Headers are used to define requests and responses. There are many common types of headers expected by both the web-browser and front-end developer. The Content-Type header specifically tells the browser how to render the information coming back.

> Just like status codes, you can also find a list of standard HTTP header fields on Wikipedia.

Finally, we add the information we're sending back:

```js
response.end('Hello World');
```

The `end` method on the `response` indicates we are done adding information to the response. At this point the server can go ahead and respond to the client's request. In this method we can also add to the **body** of the response, which is the main part of the response. When we serve an HTML page, the body will be the HTML page itself. The headers simply describe what is coming back.

Listening

Now that we've created the server, we need to tell it **how to listen** to requests:

```js
server.listen({ port: 3000, host: 'localhost' }, function() {
  console.log('Server is running!');
});
```

In this code we are telling the server to listen on **port** 3000 on our **localhost**. When this connection is ready, it will fire the function which logs `"Server is running!"` .

The **port** allows us to create connections to several different processes on our machine. We could run two different node servers on our computer simply by specifying different ports. For instance we could use 3000 for one and 3001 for another.

> If you use a port that's already in use in Node.js you'll get back a EADDRINUSE error when you try to listen to the port.

Ports can be any unsigned integer from 0 to 65535. Some ports are commonly reserved for certain processes. It's become a standard web-development practice to use 3000 as the server port and incrementing from there. Some build processes will look for the nearest open port after 3000.

> You can visit this article for a list of well-known ports.

Wrap Up

Well that's it for this lesson! We went over the basics of servers, specifically in Node.js in good detail!

Mark Complete