COMP2068 JavaScript Frameworks

Lesson 3
Node “Style Guide” & Routing w/Connect

Lesson Objectives

In this Lesson you will learn about:

1. Examine some Node.js use cases
2. Explore Node’s code style conventions
3. Review a basic node http file
4. Introduce routing with the Connect middleware

Why use Node.js?

- https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js

Unofficial Node.js Style Guide

- https://github.com/felixge/node-style-guide

Review Exercise: Using the HTTP module

- Create a new file called loop.js
- Include Node's built-in http module
- Start the web server on port 3000, logging out a message to the console that the server is running
- Inside the callback of the createServer method, loop from 1 to 20 and output these numbers to the browser, each time on a separate line

Meet the Connect module

- **Connect** is a module built to support interception of requests in a more modular approach.
- In our previous web server examples, we built a simple web server using the `http` module that ships with Node.
- If you wish to extend these examples, you'd have to write code that manages the different HTTP requests sent to your server, handles them properly, and responds to each request with the correct response.

Meet the Connect module (cont'd)

- **Connect** creates an API exactly for that purpose – it uses a modular component called **middleware**, which allows you to simply register your application logic to **predefined** HTTP request scenarios.
- Connect **middleware** are basically callback functions, which get executed when an HTTP request occurs.
- The **middleware** can then **perform some logic**, **return a response**, or call the next registered middleware.
- While you will mostly write **custom middleware** to support your application needs, Connect also includes some common middleware to support **logging**, **static file serving**, and more.

Meet the Connect module (cont’d)

- The way a Connect application works is by using an object called **`dispatcher`**.
- The **`dispatcher`** object handles each HTTP request received by the server and then decides, in a cascading way, the order of middleware execution.

Meet the Connect module (cont'd)

- Next week, we'll create our first **Express** application, but **Express** is based on **Connect's** approach, so in order to understand how **Express** works, we'll begin with creating a **Connect** application.
- In your working folder, create a file named **`server.js`** that contains the following code snippet: (we'll run this after the install of connect on the next slide)

```
let connect = require('connect');
let app = connect();
app.listen(3000);
console.log('Server running at
http://localhost:3000/');
```

Meet the Connect module (cont'd)

- As you can see, your application file is using the **`connect`** module to create a new **web server**.
- However, **Connect** isn't a core module, so you'll have to install it using **NPM**.
- To do so, use your command-line tool, and navigate to your working folder. Then execute the following command:
- First, let's initialize npm (and hit enter to build the package.json file):

```
$ npm init
$ npm i connect
```

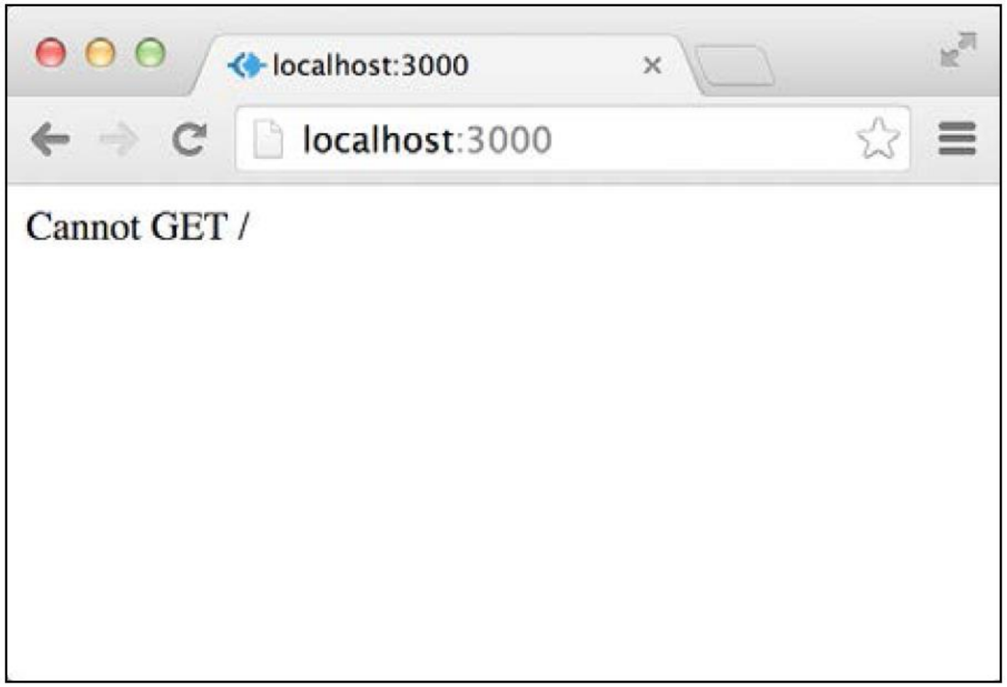
- **NPM** will install the connect module inside a **node_modules** folder, which will enable you to **require** it in your application file. The **--save** option adds connect the list of dependencies in package.json
- To run your Connect web server, just use Node's CLI and execute the following command:

```
$ node server
```

Meet the Connect module (cont'd)

- Node will run your application, reporting the server status using the **`console.log()`** method.
- You can try reaching your application in the browser by visiting **`http:// localhost:3000`**.
- However, you should get a response similar to what is shown in the following screenshot:

❖ What this response means is that there isn't any **middleware** registered to handle the **GET HTTP** request.



```
Cannot GET /
```

Connect Middleware

- Connect **middleware** is just **JavaScript function** with a unique signature.
- Each middleware function is defined with the following three arguments:
  - **req:** This is an object that holds the HTTP **request** information
  - **res:** This is an object that holds the HTTP **response** information and allows you to set the response properties
  - **next:** This is the next middleware function defined in the ordered set of Connect middleware (this is often unused in reality)

Connect Middleware (cont’d)

- When you have a **middleware** defined, you'll just have to register it with the **Connect** application using the **`app.use()`** method.
- Let's revise the previous example to include your first **middleware**.
- Change your **`server.js`** file to look like the following code snippet:

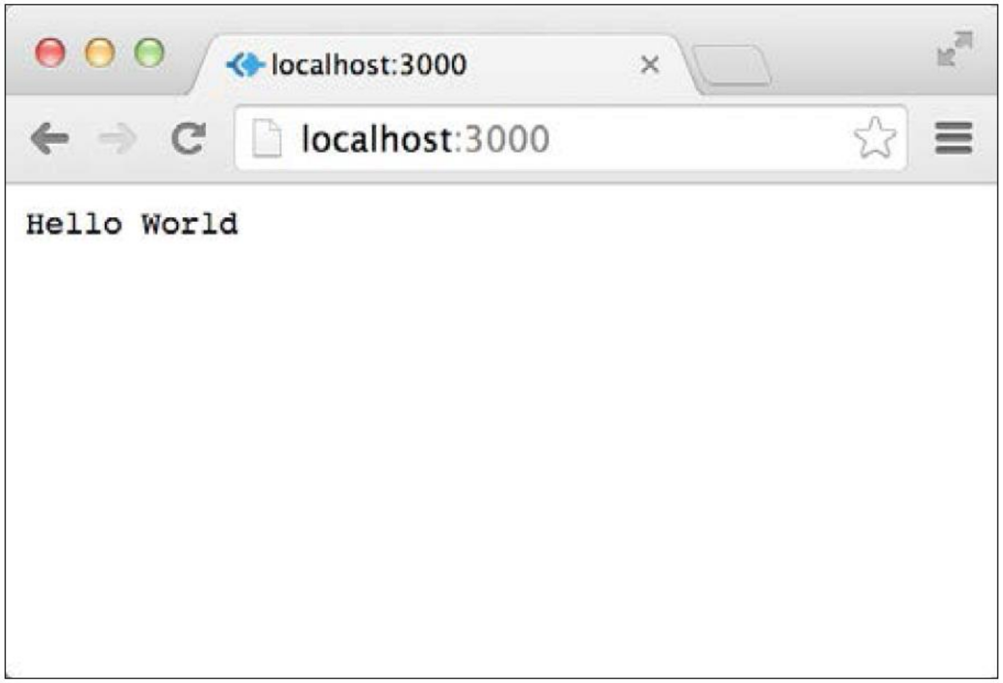
```
let connect = require('connect');
let app = connect();
let helloWorld = (req, res, next) => {
   res.writeHead(200);
   res.end('Hello World');
};
app.use(helloWorld);
app.listen(3000);
console.log('Server running at http://localhost:3000/');
```

Connect Middleware (cont'd)

- Then, start your **connect** server again by issuing the following command in your command-line tool:

```
$ node server
```

- Try visiting **`http://localhost:3000`** again.
- You will now get a response similar to that in the following screenshot:



```
Hello World
```

Mounting Connect middleware

- As you may have noticed, the **middleware** you registered responds to **any request** regardless of the request path.
- This does not comply with modern web application development because responding to different paths is an integral part of all web applications.
- Fortunately, Connect middleware supports a feature called **mounting**, which enables you to determine which request path is required for the middleware function to get executed.
- **Mounting** is done by adding the **`path`** argument to the **`app.use()`** method.
- To understand this better, let's revisit our previous example.

Mounting Connect middleware (cont'd)

```
let connect = require('connect')
let app = connect();
let helloWorld = (req, res, next) => {
    res.writeHead(200, { 'Content-Type': 'text/html; charset=UTF-8' })
    res.write('<h1>Hello World</h1>');
    res.end();
};

let goodbyeWorld = (req, res, next) => {
  res.writeHead(200, { 'Content-Type': 'text/html; charset=UTF-8' })
  res.write('<h1>Goodbye World</h1>');
  res.end();
};

let index = (req, res, next) => {
    if (req.url == '/') {
        res.writeHead(200, { 'Content-Type': 'text/html; charset=UTF-8' })
        res.write('<h1>Connect Home Page</h1>');
        res.end();
    }
    else {
        res.writeHead(404);
        res.write('<h1>Page Not Found</h1>');

    }
    res.end();
};

app.use('/hello', helloWorld);
app.use('/goodbye', goodbyeWorld);
app.use('/', index); // must be last, or else it overrides ALL other paths

app.listen(3000);
```