

CS 4220 - NODE.JS & ANGULAR.JS

NODE MODULES

AGENDA

- ▶ http - Core node.js module
 - ▶ request - Simplified http requests
 - ▶ cheerio - Web Scraping
- ▶ crypto - Core node.js module
 - ▶ Generating hashes
 - ▶ Identifying duplicate files

HTTP

- ▶ The core *http* module can be used to:
 - ▶ Create a local server to service incoming requests
 - ▶ Submit http requests to other servers
 - ▶ This will be our focus
- ▶ Documentation: <https://nodejs.org/api/http.html>

SUBMITTING AN HTTP REQUEST

HTTP.REQUEST()

- ▶ Require *http* module
- ▶ Create *options* object
- ▶ Submit http request
- ▶ Read status
- ▶ Read headers

```
1  const http = require('http')
2
3  const options = {
4    hostname: 'albertcervantes.com',
5    port: 80,
6    path: '/hello.html',
7    method: 'GET',
8    headers: {
9      'Content-Type': 'text/html'
10   }
11 };
12
13 const req = http.request(options, (res) => {
14   console.log(`STATUS: ${res.statusCode}`);
15   console.log(`HEADERS: ${JSON.stringify(res.headers)}`);
16   res.setEncoding('utf8');
```

HTTP.REQUEST()

- ▶ Read response body *in chunks*
- ▶ Handle errors
- ▶ Call `.end()`

```
17     res.on('data', (chunk) => {
18         console.log(`BODY: ${chunk}`);
19     });
20     res.on('end', () => {
21         console.log('No more data in response.');
```

One must always call `req.end()` to signify that you're done with the request - even if there is no data being written to the request body.

.REQUEST VS .GET

HTTP - .GET()

- ▶ Connect to external server
- ▶ Request *hello.html*
- ▶ Handle errors

```
1  const http = require('http')
2
3  http.get('http://albertcervantes.com/hello.html', (res) => {
4
5      // Read some information about the response
6      const statusCode = res.statusCode;
7      const contentType = res.headers['content-type'];
8
9      let error;
10     if (statusCode !== 200) {
11         error = new Error(`Request Failed.\n` +
12             `Status Code: ${statusCode}`);
13     }
14     if (error) {
15         console.log(error.message);
16
17         // consume response data to free up memory
18         res.resume();
19
20         return;
21     }
```


HTTP - .GET()

- ▶ Read response
- ▶ Display on console

```
22
23     // Read the contents of the response in chunks
24     res.setEncoding('utf8');
25     let html = '';
26     res.on('data', (chunk) => html += chunk);
27
28     res.on('end', () => {
29         console.log(html)
30     });
31
32 }).on('error', (e) => {
33     console.log(`Got error: ${e.message}`);
34 });
```

Since most requests are GET requests without bodies, Node.js provides this convenience method.

The only difference between this method and **http.request()** is that it sets the method to GET and calls `req.end()` automatically.

**THERE HAS TO BE AN EASIER
WAY!**

Every Developer

REQUEST

- ▶ The *request* module is a "Simplified HTTP request client."
- ▶ Github: <https://github.com/request/request>
- ▶ Install: `npm install request`
- ▶ Our goal is to simplify making http requests, submitting form-data, and processing results.

MODULE: REQUEST

- ▶ Require *request* module
- ▶ Make request
- ▶ *Error(s), Response, and Body* handled automatically

```
1  var request = require('request');
2  request('http://albertcervantes.com/hello.html', (error, response, body) => {
3      // Print the error if one occurred
4      console.log('error:', error);
5
6      // Print the response status code if a response was received
7      console.log('statusCode:', response && response.statusCode);
8
9      // Print the HTML for the hello.html doc
10     console.log('body:', body);
11 });
```

WEB SCRAPING

WEB SCRAPING

- ▶ Our goal is to work with HTML on the server in the same way we would work with it in the browser.
 - ▶ jQuery is a typical client-side library used for this purpose
 - ▶ <http://jquery.com>
- ▶ How do we interrogate the HTML that is returned from an HTTP request?
 - ▶ In the browser we rely on the Document Object Model (DOM) to ask questions about the structure of the document.
 - ▶ Node.js does not provide a DOM to interrogate, and many off-the-shelf client-side libraries fail when you try to use them in node.

TEACH YOUR SERVER HTML.

Cheerio.js

CHEERIO

- ▶ The cheerio module is a “[f]ast, flexible, and lean implementation of core jQuery designed specifically for the server”.
- ▶ Github: <https://cheerio.js.org>
- ▶ Install: `npm install cheerio`

CHEERIO

- ▶ Require *cheerio* module
- ▶ Load the *HTML* into cheerio
 - ▶ Assign result to variable, typically *\$*
- ▶ Use *\$* like jQuery

```
1  const cheerio = require('cheerio'),
2      $ = cheerio.load('<h2 class = "title">Hello world</h2>');
3
4  console.log('Before manipulation: ', $.html());
5
6  $('h2.title').text('Hello there!');
7  $('h2').addClass('welcome');
8
9  console.log('After manipulation: ', $.html());
```

CHEERIO

- ▶ Answer the following questions using cheerio and the following html document: <http://albertcervantes.com/hello.html>
 - ▶ How many paragraphs are present on the page?
 - ▶ How many images?
 - ▶ What are the URLs of all images on the page?
 - ▶ What is the average length of all paragraphs?
 - ▶ What is the average word count of all paragraphs?

HASHING

A HASH FUNCTION IS ANY **FUNCTION**
THAT CAN BE USED TO MAP **DATA** OF
ARBITRARY SIZE TO DATA OF FIXED
SIZE.

Wikipedia

HASHING & CRYPTO

▶ SHA256

- ▶ The SHA (Secure Hash Algorithm) is one of a number of cryptographic hash functions.
- ▶ A cryptographic hash is like a signature for a text or a data file.
- ▶ SHA-256 algorithm generates an almost-unique, fixed size 256-bit (32-byte) hash.
- ▶ Hash is a one way function – it cannot be decrypted back.
- ▶ This makes it suitable for password validation, challenge hash authentication, anti-tamper, digital signatures.

Source: <http://www.xorbin.com/tools/sha256-hash-calculator>

CRYPTO

- ▶ The crypto module provides cryptographic functionality that includes a set of wrappers for OpenSSL's hash, HMAC, cipher, decipher, sign and verify functions.
- ▶ We can use a SHA256 hash to generate a unique identifier based on the *contents* of a file.
- ▶ Documentation: https://nodejs.org/api/crypto.html#crypto_class_hash

LAB

- ▶ Write a node.js application that:
 - ▶ Takes a directory as a command-line-argument
 - ▶ Traverses the directory
 - ▶ Traverses all sub-directories
 - ▶ Prints out all filenames (full-path) that are duplicates of each other.

LAB - HINTS

▶ node-dir

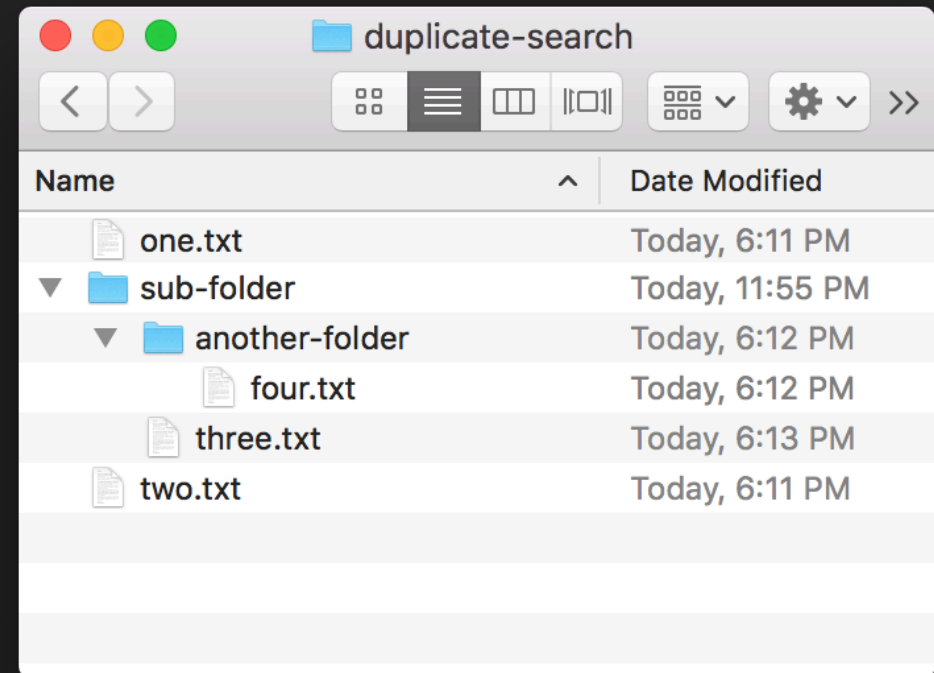
- ▶ Consider using a module like *node-dir* to traverse a folder structure
- ▶ The *readFiles* method is particularly useful
- ▶ Install: `node install node-dir`
- ▶ URL: <https://www.npmjs.com/package/node-dir>

▶ treeify

- ▶ Consider nicely formatted output as a nice-to-have. However, consider using a module like *treeify* to format the output in a meaningful way.
- ▶ Install: `node install treeify`
- ▶ URL: <https://www.npmjs.com/package/treeify>

LAB - SAMPLE RUN

- ▶ My *duplicate-search* folder contains files and a directory.
- ▶ The directory contains additional files and a directory.
- ▶ I run my *duplicate-search.js* application from the *parent directory*, and I pass in the *./duplicate-search* path.
- ▶ The output is displayed in a tree-like fashion.



```
$ node duplicate-search ./duplicate-search
```

The following duplicates were found:

```
├ duplicate-search/one.txt
├   duplicate-search/sub-folder/three.txt
├ duplicate-search/sub-folder/another-folder/four.txt
├   duplicate-search/two.txt
```