Node JS, Express & MongoDB

NodeJs –

* Runtime Application.
* Server javascript
* Asynchronous events (non blocking)
* Non-blocking I/O

**Require**: function – use to import other nodejs modules

**Path**: module comes with node – gives us some tools to work with paths

* path.basename(\_\_filename) - methods returns the last portion of a path [path.basename(path[, ext])](https://nodejs.org/api/path.html#path_path_basename_path_ext)
* [path.dirname(path)](https://nodejs.org/api/path.html#path_path_dirname_path) - The path.dirname() method returns the directory name of a path, similar to the Unix dirnamecommand.
* path.join([...paths])[#](https://nodejs.org/api/path.html#path_path_join_paths)
* ...paths [<String>](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#String_type) A sequence of path segments
* The path.join() method joins all given path segments together using the platform specific separator as a delimiter, then normalizes the resulting path.

**Util:** The util module is primarily designed to support the needs of Node.js' own internal APIs.  However, many of the utilities are useful for application and module developers as well. It can be accessed using: const util = require('util'); Used to log with date stamp 🡪var util = require('util'); util.log(path.basename(\_\_filename));

util.inhertits(Person, EventEmitter) - The utilities module has an inherits function, and it's a way that we can add a object to the prototype of an existing object.

**V8**: module to get information about memory – uses chromes v8 processor - var v8 = require('v8'); util.log(v8.getHeapStatistics());

**ReadLine:** module ask question about terminal usage var readline =require('readline'); var rl = readline.createInterface(process.stdin, process.stdout);

**Event: eventEmitter is part of the events module** The Event Emitter is Node.js's implementationof the pub/sub design pattern, and it allows us to create listeners for an emit custom Events.  In fact, every time we've used that on function to listen for a new Event, we've already been using an implementation of the EventEmitter.

**Process**: object – interact with current process instance –

* process.argv.indexOf(flag);
* process.stdout.write(‘hello’); - to write it to stdout
* process.stdin.on('data', function(data){})
* process.stdout.clearLine();
* process.stdout.cursorTo(0);
* process.exit();
* process.on('exit', function() {})

**Timer:** The timer module exposes a global API for scheduling functions to be called at some future period of time.

* [Class: Immediate](https://nodejs.org/api/timers.html#timers_class_immediate)
* [Class: Timeout](https://nodejs.org/api/timers.html#timers_class_timeout)
* [timeout.ref()](https://nodejs.org/api/timers.html#timers_timeout_ref)
* [timeout.unref()](https://nodejs.org/api/timers.html#timers_timeout_unref)
* [Scheduling Timers](https://nodejs.org/api/timers.html#timers_scheduling_timers)
* [setImmediate(callback[, ...args])](https://nodejs.org/api/timers.html#timers_setimmediate_callback_args)
* [setInterval(callback, delay[, ...args])](https://nodejs.org/api/timers.html#timers_setinterval_callback_delay_args)
* [setTimeout(callback, delay[, ...args])](https://nodejs.org/api/timers.html#timers_settimeout_callback_delay_args)
* [Cancelling Timers](https://nodejs.org/api/timers.html#timers_cancelling_timers)
* [clearImmediate(immediate)](https://nodejs.org/api/timers.html#timers_clearimmediate_immediate)
* [clearInterval(timeout)](https://nodejs.org/api/timers.html#timers_clearinterval_timeout)
* [clearTimeout(timeout)](https://nodejs.org/api/timers.html#timers_cleartimeout_timeout)

**Http**:  designed to be the simplest way possible to make http calls

We need to add meta data for header and response

// Configure our HTTP server to respond with Hello World to all requests.

var server = http.createServer(function (request, response) {

response.writeHead(200, {"Content-Type": "text/plain"});

response.end("Hello World\n");

});

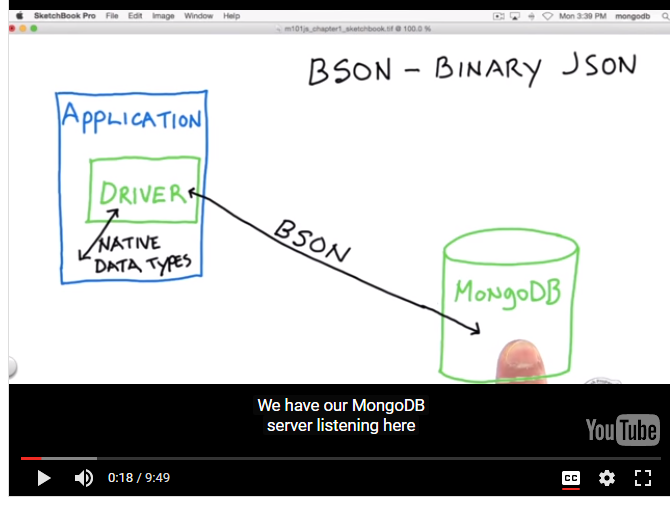
Module.exports is the object that is returned by the require statement.

Child\_process: this module allows you to execute external processes in your environment

* Exec:  if we have these processes that have small bits of data, they're perfect for execute.
* Spawn: is made for longer, ongoing processes with large amounts of data

**Fs :** module - can be used to list files and directories, create files and directories, stream files, write files, read files, modify file permissions or just about anything that you need to be able to do with the file system. Can read content of text and binary files

Npm: package manager for nodejs:

* Require – includes the external packages in nodejs application
* Driver: library in a specific language. Here it communicates with the MongoDB server
* 

Express:

Frame work. It handles routing, request param and other details of HTTP requests, and provides API to work with requests more easily to handle requests.

* Streamlined NodeJS - make things faster
* Adds server methods – popular modules
* Routing mechanism – different type of request
* Easier APIs – easier to create an API
* Middleware friendly –other modules
* Easy access to public files

Creating Routes:

* App.get();
* Pass variables res.params

**router.get**('/speakers/:speakerid', function(**req, res**) {

var **dataFile** = req.app.get('appData');

var speaker = dataFile.speakers[req.params.speakerid];

**res.send**(`

<h1>${speaker.title}</h1>

<h2>with ${speaker.name}</h2>

<p>${speaker.summary}</p>

`);

});

One of the best features of Express is how easy it is to work with routing. Routing allows you to give users access to different types of data, and we can pass along information to the routes through the params variable of the request object.

Modularization

* NodeJS makes it super simple to build modules with built-in methods that let you import and output content.
* Require() - lets you import one document into another

var express = require('express'); - express library into the module.

var router = express.Router(); -- to use the router object of the Express library so that I can work with these external routes.

* module.exports - allows you to export the contents of a module.

module.exports = router ; We have to use this router object, and in order for our app.js file to get everything in this route, we're gonna need to export the router object.

* app.use() - Within Express, the use method allows our application to use external routes as part of the application.

Ex : ----- app.use(require('./routes/index'));

* Express.Router() - router object provided by the Express library to create an instance of the route. This makes the route mountable by our original application,
* app.set() - The set method of your app can create variables that then can be passed to your routes.

app.set('port', process.env.PORT || 3000 );

app.set('appData', dataFile); -- app information is going to be available through our request

* req.app.get() - That information can be passed along to your modules through the request object that is going to be generated when you call the application.

var dataFile = req.app.get('appData'); the app information and then use a get command to get this variable that I created in my main application called appData.

Public Folders:

Express Middleware

* Middleware is any library that can be called as part of the process of working with a site that then returns control back to Express.
* express.static() -  it allows you to designate a folder for holding any sort of document or file that we want our users or our routes to have access to

app.use(express.static('app/public')); use an app.use command and then I'll use express.static and that is a method that Express has that will allow us to designate a folder.

Workflow Automation

* watch for changes
* reload browser
* use package.json script -- Nodemon & Reload – npm install –g nodemon

nodemonitor – Nodemon

package.json

"start": "nodemon -e css,ejs,js,json --watch app"

So for example, you may not want to do anything if somebody updates any markdown documents, like their readme file, it doesn't really belong to the server and so you can just specify certain extensions, so for example, just the CSS files,EJS documents, JavaScript files, or JSON documents.

**Reload**

So what about that reloading of the page? It's sort of annoying that it is noticing the changes, but it doesn't do anything else with the browser. So to get that working, we need to install this other npm plugin, called Restart. So I'm gonna cancel this. So I'm gonna issue an npm install -g to install this locally. And the package I need is called reload.

Npm install –g reload --- globally

Npm install –save reload --- to install this as a dependency for the project.

App.js

Var reload =require(‘reload’);

Reload(server,app);

Index.js

<script src=”/reload/reload.js”></script>

Template libraries

**Consolidate:** template engine

app.engine('html', engines.nunjucks); 🡪 registering nunjucks template engine with html extension

app.set('view engine', 'html'); 🡪 to render our html file by using above engine

app.set('views', \_\_dirname + '/views'); 🡪 where our templates are located.

\_\_dirname 🡪 nodejs environment variable - where to look for views

res.render('hello', { name : 'Templates' });🡪 use render the template to use and give an object in which we will pass the value to the variable in the template

**Bodyparser:** middleware with express. Telling express to do extra processing the body of route

**MongoDB connection from NodeJS**

var MongoClient = require('mongodb').MongoClient,

assert = require('assert');

MongoClient.connect('mongodb://localhost:27017/crunchbase', function(err, db) {})

We can replace localhost with name or ip address of that host

//to get all the records at once from DB

db.collection('companies').find(query).toArray(function(err, docs) {})

//to get one records at a time from DB

var cursor = db.collection('companies').find(query);

cursor.forEach(function(doc) {})